

Volume
2

M I C R O S O F T®

OS/2

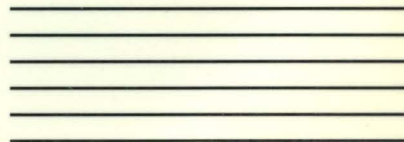
Programmer's Reference



Including Presentation Manager

Microsoft
OS/2

PROGRAMMER'S
REFERENCE
LIBRARY

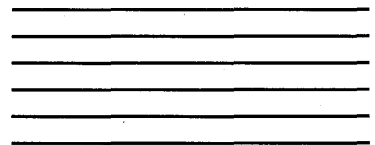
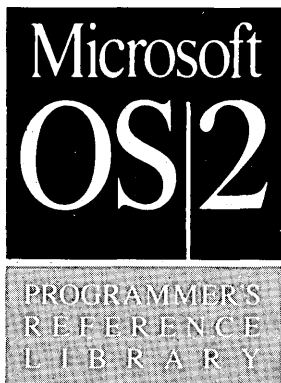


Volume
2

Microsoft[®] Operating System/2 Programmer's Reference

Version 1.1

Written, edited, and produced
by Microsoft Corporation
Distributed by Microsoft Press



Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft Corporation. The software and/or databases described in this document are furnished under a license agreement or nondisclosure agreement. The software and/or databases may be used or copied only in accordance with the terms of the agreement. The purchaser may make one copy of the software for backup purposes. No part of this manual and/or database may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than the purchaser's personal use, without the written permission of Microsoft Corporation.

PUBLISHED BY

Microsoft Press

A Division of Microsoft Corporation

16011 NE 36th Way, Box 97017, Redmond, Washington 98073-9717

© Copyright Microsoft Corporation, 1989. All rights reserved.

Library of Congress Cataloging in Publication Data

Microsoft OS/2 programmer's reference.

Includes index.

1. Microsoft OS/2 (Computer operating system) I. Microsoft Press

QA76.76.063078 1989 005.4'469 89-2817

ISBN 1-55615-221-3(Vol. 2)

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 FGFG 3 2 1 0 9

Distributed to the book trade in the United States by Harper & Row.

Distributed to the book trade in Canada by General Publishing Company, Ltd.

Distributed to the book trade outside the United States and Canada

by Penguin Books Ltd.

Penguin Books Ltd., Harmondsworth, Middlesex, England

Penguin Books Australia Ltd., Ringwood, Victoria, Australia

Penguin Books N.Z. Ltd., 182-190 Wairau Road, Auckland 10, New Zealand

The character-set tables in this manual are reprinted by permission from the *IBM Operating System/2 User's Reference*, © 1987 by International Business Machines Corporation.

Microsoft®, MS®, MS-DOS®, and the Microsoft logo are registered trademarks of Microsoft Corporation.

IBM®, PC/AT®, and Personal System/2® are registered trademarks of International Business Machines Corporation.

Contents

Chapter 1 Introduction

1.1	Overview	3
1.2	How to Use This Manual	4
1.3	Naming Conventions	7
1.4	Notational Conventions	10

Chapter 2 Functions Directory

2.1	Introduction	13
2.2	Functions	14

Chapter 3 Messages Directory

3.1	Introduction	393
3.2	Messages	394

Chapter 4 Types, Macros, Structures

4.1	Introduction	471
4.2	Types	472
4.3	Macros	473
4.4	Structures	485

Chapter 5 File Formats

5.1	Introduction	529
5.2	Font-File Format	529
5.3	Font Signature	530
5.4	Font Metrics	530
5.5	Font Character Definition	536
5.6	Code-Page Font Support	540

Appendixes

Appendix A Error Values

A.1	Introduction	547
A.2	Errors	547

Appendix B Device Capabilities

B.1	Introduction	553
B.2	About Device Capabilities	553

Index

Figures

Figure 1.1	Sample Reference Page	4
------------	-----------------------------	---

Tables

Table 5.1	Additional Glyphs.....	541
-----------	------------------------	-----



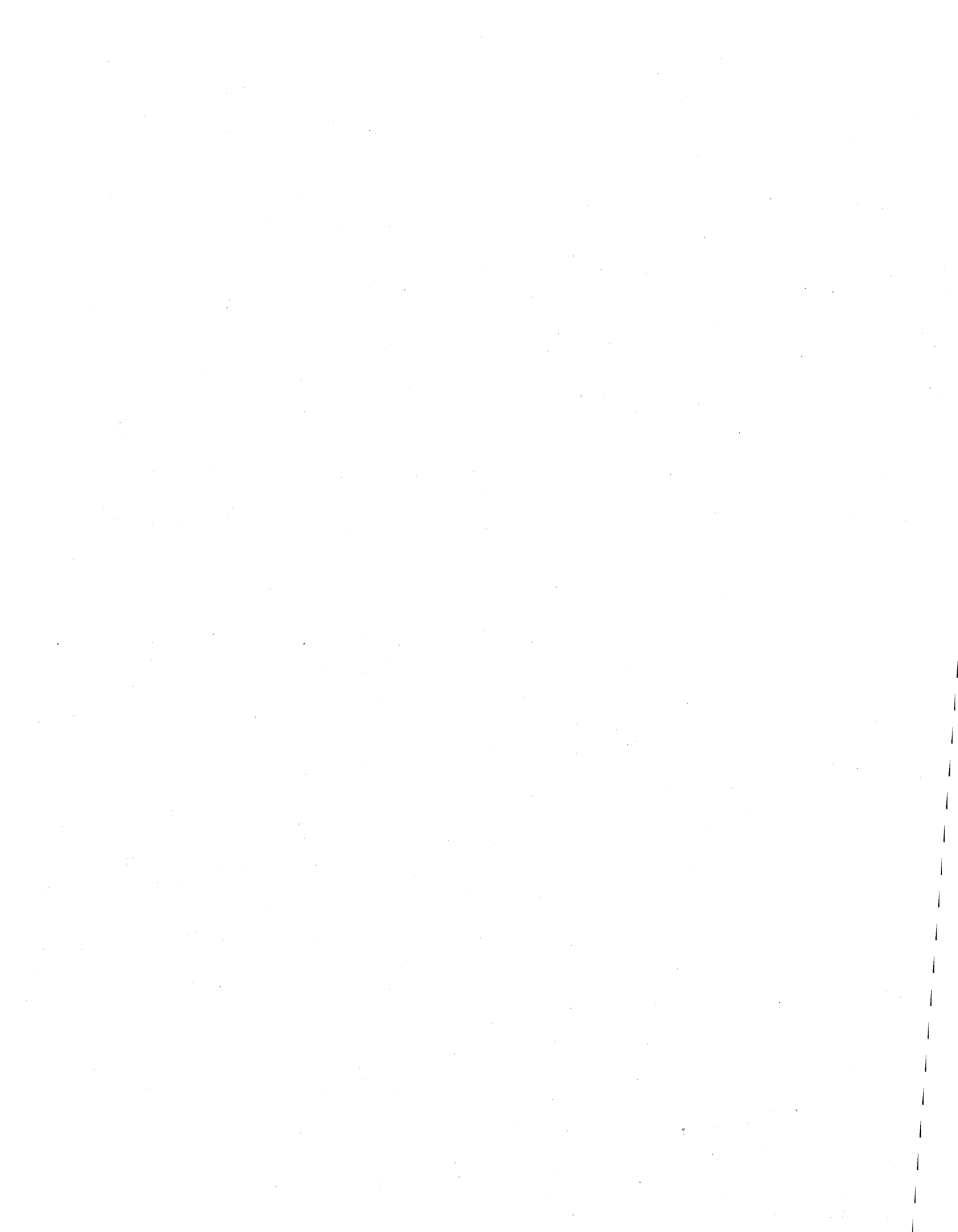


Chapter

1

Introduction

1.1	Overview	3
1.2	How to Use This Manual.....	4
1.2.1	C Format	5
1.2.2	MS OS/2 Include Files	5
1.2.3	MS OS/2 Calling Conventions	5
1.2.4	Bit Masks in Function Parameters.....	7
1.2.5	Structures	7
1.3	Naming Conventions.....	7
1.3.1	Parameter and Field Names	8
1.3.1.1	Prefixes	8
1.3.1.2	Base Types	9
1.3.2	Constant Names	10
1.4	Notational Conventions	10



1.1 Overview

This manual describes the **Dev**, **Gpi**, and **Win** system functions of Microsoft® Operating System/2 (MS® OS/2). These functions, also called the Presentation Manager functions, let programs use the window-management and graphics features of MS OS/2.

MS OS/2 system functions are designed to be used in C, Pascal, and other high-level-language programs, as well as in assembly-language programs. In MS OS/2, all programs request operating-system services by calling system functions.

This chapter, "Introduction," shows how to use this manual, provides a brief description of MS OS/2 calling conventions, illustrates function calls in various languages, and outlines MS OS/2 naming conventions.

Chapter 2, "Functions Directory," is an alphabetical listing of MS OS/2 Presentation Manager functions. This chapter defines each function's purpose, gives its syntax, describes the function parameters, and gives possible return values. Many functions also show simple program examples that illustrate how the function is used to carry out simple tasks.

Chapter 3, "Messages Directory," lists the messages sent and received by MS OS/2 Presentation Manager functions.

Chapter 4, "Types, Macros, Structures," describes the types, macros, and structures used by MS OS/2 Presentation Manager functions.

Chapter 5, "File Formats," describes the format of font files. Font files contain bitmap or vector information that MS OS/2 needs for drawing characters using **Gpi** functions.

Appendix A, "Error Values," lists error codes and their corresponding values.

Appendix B, "Device Capabilities," lists the device capabilities that can be determined by using the **DevQueryCaps** function.

This manual is intended to fully describe MS OS/2 Presentation Manager functions and the structures and file formats used with these functions. It does not show how to use these functions to carry out specific tasks. For more information on this topic, see the *Microsoft Operating System/2 Programmer's Reference, Volume 1*. Also, this manual does not describe MS OS/2 base system functions. MS OS/2 base system functions let programs use the operating system to carry out tasks such as reading from and writing to disk files; allocating memory; starting other programs; and using the keyboard, mouse, and video screen. For more information on MS OS/2 base system functions, see the *Microsoft Operating System/2 Programmer's Reference, Volume 3*.

1.2 How to Use This Manual

This manual provides detailed information about each MS OS/2 Presentation Manager function, message, macro, and structure. Each description has the following format:

Figure 1.1
Sample Reference Page

1	■	WinAlarm
2	[<pre> BOOL WinAlarm(<i>hwndDesktop</i>, <i>fsType</i>) HWND <i>hwndDesktop</i>; /* handle of the desktop */ USHORT <i>fsType</i>; /* alarm style */ </pre>
3	Ⓢ	The WinAlarm function generates an audible alarm that can be used to alert the user about special conditions.
4	Parameters	<p><i>hwndDesktop</i> Identifies the desktop window. This parameter can be HWND_DESKTOP or the desktop window handle.</p> <p><i>fsType</i> Specifies the alarm style. It can be one of the following values:</p> <pre> WA_WARNING WA_NOTE WA_ERROR </pre>
5	Return Value	The return value is TRUE if the function is successful or FALSE if an error occurs.
6	Example	<p>This example calls an application-defined initialization function, and if the function fails it calls WinAlarm to generate an audible alarm notifying the user the initialization failed.</p> <pre> if (!GenericInit()) { WinAlarm(HWND_DESKTOP, WA_ERROR); /* general initialization */ } </pre>
7	See Also	WinFlashWindow, WinSetSysValue

These are the elements shown:

- 1 The function, message, macro, or structure name.
- 2 The function, message, macro, or structure syntax. The syntax specifies the number of parameters (or fields) and gives the type of each. It also gives the order (from left to right) that parameters must be pushed on the stack. Comments to the right briefly describe the purpose of the parameter (or field).
- 3 A description of the function, message, macro, or structure, including its purpose and details of operation.
- 4 A full description of each parameter (or field), including permitted values and related structures.
- 5 A description of the function return value, including possible error values.
- 6 An example showing how the function can be used to accomplish a simple task.
- 7 A list of related functions, structures, macros, and messages.

1.2.1 C Format

In this manual, the syntax for MS OS/2 functions is given in C-language format. In your C-language sources, the function name must be spelled exactly as given in the syntax and the parameters must be used in the order given in the syntax. This syntax also applies to Pascal program sources.

The following example shows how to call the WinAlarm function in a C-language program:

```
/* sound an alarm when an error occurs */
WinAlarm(HWND_DESKTOP,      /* alarm for the desktop window */
          WA_ERROR);        /* tone for errors */
```

1.2.2 MS OS/2 Include Files

This manual uses many types, structures, and constants that are not part of standard C language. These items, designed for MS OS/2, are defined in the MS OS/2 C-language include files provided with the Microsoft OS/2 Presentation Manager Softset and the Microsoft OS/2 Presentation Manager Toolkit.

In C-language programs, the `#include` directive specifying `os2.h`, the MS OS/2 C-language include file, can be placed at the beginning of the source file to include the definitions for the special types, structures, and constants. Although there are many MS OS/2 include files, the `os2.h` file contains the additional `#include` directives needed to process the basic MS OS/2 definitions.

To speed up processing of the MS OS/2 C-language include files, many definitions are processed only if the C-language program explicitly defines a corresponding include constant. An include constant is simply a constant name, with the prefix `INCL_`, that controls a portion of the include files. If a constant is defined using the `#define` directive, the corresponding MS OS/2 definitions are processed. For a list of the include constants and a description of the MS OS/2 system functions they enable, see the *Microsoft Operating System/2 Programmer's Reference, Volume 1*.

1.2.3 MS OS/2 Calling Conventions

You must know MS OS/2 calling conventions to use MS OS/2 functions in other high-level languages or in assembly language. MS OS/2 functions use the Pascal (sometimes called the PLM) calling convention for passing parameters, and they apply some additional rules to support dynamic-link libraries. The following rules apply:

- You must push the parameters on the stack. In this manual, each function description lists the parameters in the order they must be pushed. The left parameter must be pushed first, the right parameter last. If a parameter specifies an address, the address must be a far address; that is, it must have the form *selector:offset*. The *selector* must be pushed first, then the *offset*.
- The function automatically removes the parameters from the stack as it returns. This means the function must have a fixed number of parameters.
- You must use an intersegment call instruction to call the function. This is required for all dynamic-link-library functions.

- The function returns a value, possibly an error value, in either the `ax` register or the `dx:ax` register pair. Only the `di` and `si` register values are guaranteed to be preserved by the function. MS OS/2 Presentation Manager functions may preserve other registers as well, but they do not preserve the `flags` register. The contents of the `flags` register are undefined; specifically, the direction flag in the register may be changed. However, if the direction flag was zero before the function was called, it will be zero after the function returns.

The following example shows how MS OS/2 calling conventions apply to the `WinCreateStdWindow` function in an assembly-language program:

```

EXTRN WINCREATESTDWINDOW:FAR

hwndParent      dd      01H
flCreateFlags   dd      0FH
szClientClass   db      "MyClass", 0
szTitle         db      "My Window", 0
hwndClient      dd      01H

push word ptr [hwndParent+2]      ; handle of the parent window
push word ptr [hwnParent]        ; frame-window style
push 0                          ; creation flags
push ds                          ; creation flags
push offset flCreateFlags
push ds                          ; client-window class name
push offset szClientClass        ; address of title-bar text
push ds                          ; client-window style
push 0                          ;
push 0                          ; handle of the resource file
push 1                          ; resource identifier
push ds                          ; address of client-window handle
push offset hwndClient
call WINCREATESTDWINDOW

```

The following example shows how to call the same `WinCreateStdWindow` function in a C-language program. In C, the `WinCreateStdWindow` function name, parameter types, and constant names are defined in `os2.h`, the MS OS/2 C-language include file.

```

# include <os2.h>

HWND hwndParent = HWND_DESKTOP;
ULONG flCreateFlags =
    FCF_TITLEBAR | FCF_SYSTEMMENU | FCF_MENU | FCF_SIZEBORDER;
HWND hwndClient;

WinCreateStdWindow(
    hwndParent,          /* handle of the parent window */
    0L,                 /* frame-window style           */
    &flCreateFlags,     /* creation flags               */
    "MyClass",          /* client-window class name     */
    "My Window",       /* address of title-bar text    */
    0L,                 /* client-window style          */
    0,                  /* handle of the resource file  */
    1,                  /* resource identifier           */
    &hwndClient);      /* address of client-window handle */

```

1.2.4 Bit Masks in Function Parameters

Many MS OS/2 system functions accept or return bit masks as part of their operation. A bit mask is a collection of two or more bit fields within a single byte, or a short or long value. Bit masks provide a way to pack many Boolean flags (flags whose values represent on/off or true/false values) into a single parameter or structure field. In assembly-language programming, it is easy to individually set, clear, or test the bits in a bit mask by using instructions that modify or examine bits within a byte or a word. In C-language programming, however, the programmer does not have direct access to these instructions, so the bitwise AND and OR operators typically are used to examine and modify the bit masks.

Since this manual presents the syntax of MS OS/2 system functions in C-language syntax, it also defines bit masks in a way that is easiest to work with using the C language: as a set of constant values. When a function parameter is a bit mask, this manual provides a list of constants (named or numeric) that represent the correct values used to set, clear, or examine each field in the bit mask. For example, the `fsSelection` field of the `FATTRS` structure in the `Gpi>CreateLogFont` function specifies several values, such as `FATTR_SEL_ITALIC` and `FATTR_SEL_UNDERSCORE`. These represent the “set” values of the fields in the bit mask. Typically, the description associated with the value explains the result of the function if the given value is used; that is, when the corresponding bit is set. Generally, the opposite result is assumed when the value is not used. For example, using `FATTR_SEL_ITALIC` in the `fsSelection` field enables the italic font; not using it disables the italic font.

1.2.5 Structures

Many MS OS/2 system functions use structures as input and output parameters. This manual defines all structures and their fields using C-language syntax. In most cases, the structure definition presented is copied directly from the C-language include files provided with the Microsoft C Optimizing Compiler. Occasionally, an MS OS/2 function may have a structure that has no corresponding include-file definition. In such cases, this manual gives an incomplete form of the C-language structure definition to indicate that the structure is not already defined in an include file.

1.3 Naming Conventions

In this manual, all parameter, variable, structure, field, and constant names conform to MS OS/2 naming conventions. MS OS/2 naming conventions are rules that define how to create names that indicate both the purpose and data type of an item used with MS OS/2 system functions. These naming conventions are used in this manual to help you readily identify the purpose and type of the function parameters and structure fields. These conventions are also used in most MS OS/2 sample program sources to make the sources more readable and informative.

1.3.1 Parameter and Field Names

With MS OS/2 naming conventions, all parameter and field names consist of up to three elements: a prefix, a base type, and a qualifier. A name always consists of at least a base type or a qualifier. In most cases, the name also includes a prefix.

The base type, always written in lowercase letters, identifies the data type of the item. The prefix, also written in lowercase letters, specifies additional information about the item, such as whether it is a pointer, an array, or a count of bytes. The qualifier, a short word or phrase written with the first letter of each word uppercase, specifies the purpose of the item.

There are several standard prefixes and base types. These are used for the data types most frequently used with MS OS/2.

1.3.1.1 Prefixes

The following is a list of standard prefixes used in MS OS/2 naming conventions:

Prefix	Description
<i>p</i>	Pointer. This prefix identifies a far, or 32-bit, pointer to a given item. For example, <i>pch</i> is a far pointer to a character.
<i>np</i>	Near pointer. This prefix identifies a near, or 16-bit, pointer to a given item. For example, <i>npch</i> is a near pointer to a character.
<i>a</i>	Array. This prefix identifies an array of two or more items of a given type. For example, <i>ach</i> is an array of characters.
<i>i</i>	Index. This prefix identifies an index into an array. For example, <i>ich</i> is an index to one character in an array of characters.
<i>c</i>	Count. This prefix identifies a count of items. It is usually combined with the base type of the items being counted instead of the base type of the actual parameter. For example, <i>cch</i> is a count of characters even though it may be declared with the type <code>USHORT</code> .
<i>h</i>	Handle. This prefix is used for values that uniquely identify an object but that cannot be used to access the object directly. For example, <i>hfile</i> is a handle of a file.
<i>off</i>	Offset. This prefix is used for values that represent offsets from the beginning of a buffer or a structure. For example, <i>off</i> is the offset from the beginning of the given segment to the specified byte.
<i>id</i>	Identifier. This prefix is used for values that identify an object. For example, <i>idSession</i> is a session identifier.

1.3.1.2 Base Types

The following is a list of standard base types used in MS OS/2 naming conventions:

Base type	Type/Description
<i>f</i>	BOOL . A 16-bit flag or Boolean value. The qualifier should describe the condition associated with the flag when it is TRUE. For example, <i>fSuccess</i> is TRUE if successful, FALSE if not; <i>fError</i> is TRUE if an error occurs and FALSE if no error occurs. For objects of type BOOL , a zero value implies FALSE; a nonzero value implies TRUE.
<i>ch</i>	CHAR . An 8-bit signed value.
<i>s</i>	SHORT . A 16-bit signed value.
<i>l</i>	LONG . A 32-bit signed value.
<i>uch</i>	UCHAR . An 8-bit unsigned value.
<i>us</i>	USHORT . A 16-bit unsigned value.
<i>ul</i>	ULONG . A 32-bit unsigned value.
<i>b</i>	BYTE . An 8-bit unsigned value. Same as <i>uch</i> .
<i>sz</i>	CHAR[] . Array of characters, terminated with a null character (the last byte is set to zero).
<i>fb</i>	UCHAR . Array of flags in a byte. This base type is used when more than one flag is packed in an 8-bit value. Values for such an array are typically created by using the logical OR operator to combine two or more values.
<i>fs</i>	USHORT . Array of flags in a short (16-bit unsigned value). This base type is used when more than one flag is packed in a 16-bit value. Values for such an array are typically created by using the logical OR operator to combine two or more values.
<i>fl</i>	ULONG . Array of flags in a long (32-bit unsigned value). This base type is used when more than one flag is packed in a 32-bit value. Values for such an array are typically created by using the logical OR operator to combine two or more values.
<i>sel</i>	SEL . A 16-bit value that is used to hold a segment selector.

The base type for a structure is usually derived from the structure name. An MS OS/2 structure name, always written in uppercase letters, is a word or phrase that describes the size, purpose, and/or intended content associated with the type. The base type is typically an abbreviation of the structure name. The following list gives the base types for the structures described in this manual:

<i>acc</i>	<i>fcdata</i>	<i>pil</i>
<i>acct</i>	<i>fm</i>	<i>ptri</i>
<i>arcp</i>	<i>gradl</i>	<i>qmsq</i>
<i>bmi</i>	<i>hci</i>	<i>rcfx</i>
<i>bmp</i>	<i>hpga</i>	<i>rcl</i>
<i>btncd</i>	<i>ibmd</i>	<i>rgb</i>
<i>cbnd</i>	<i>krnpr</i>	<i>rgnrc</i>
<i>clsi</i>	<i>lbnd</i>	<i>sbcd</i>
<i>crst</i>	<i>matlf</i>	<i>sizfx</i>
<i>csri</i>	<i>mbhdr</i>	<i>sizl</i>
<i>ctchbf</i>	<i>mbnd</i>	<i>smhs</i>
<i>dde</i>	<i>mi</i>	<i>swctl</i>
<i>ddei</i>	<i>mqi</i>	<i>swent</i>
<i>dop</i>	<i>oi</i>	<i>swp</i>
<i>dlgt</i>	<i>pbnd</i>	<i>ti</i>
<i>dlgti</i>	<i>pib</i>	<i>ubtn</i>
<i>driv</i>	<i>proge</i>	<i>wprm</i>
<i>erri</i>	<i>progt</i>	<i>wywin</i>
<i>fat</i>	<i>ptfx</i>	

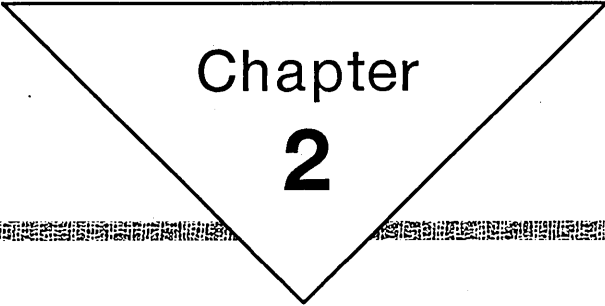
1.3.2 Constant Names

A constant name is a descriptive name for a numeric value used with an MS OS/2 function. All constant names are written in uppercase letters and have a prefix derived from the name of the function, object, or idea associated with the constant. The prefix is followed by an underscore (`_`) and the rest of the constant name, which indicates the meaning of the constant and may specify a value, action, color, or condition. A few common constants do not have prefixes—for example, `NULL` is used for null pointers of all types, and `TRUE` and `FALSE` are used with the `BOOL` data type.

1.4 Notational Conventions

The following notational conventions are used throughout this manual:

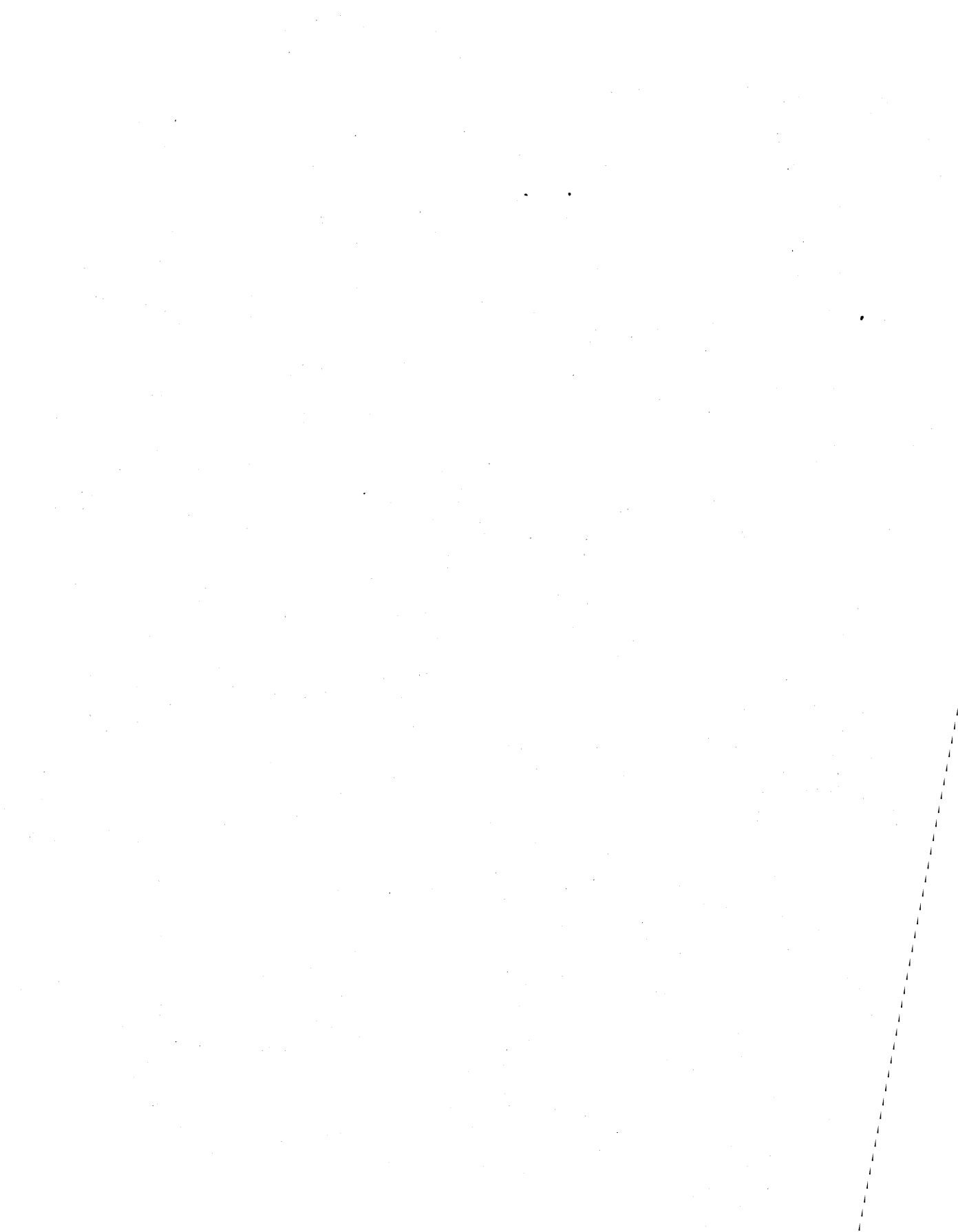
Convention	Meaning
bold	Bold type is used for keywords—for example, the names of functions, data types, structures, and macros. These names are spelled exactly as they should appear in source programs.
<i>italics</i>	Italic type is used to indicate the name of an argument; this name must be replaced by an actual argument. Italics are also used to show emphasis in text.
monospace	Monospace type is used for example program-code fragments.



Chapter
2

Functions Directory

2.1	Introduction	13
2.2	Functions.....	14



2.1 Introduction

This chapter describes MS OS/2 **Dev**, **Gpi**, and **Win** functions. These functions, also called MS OS/2 Presentation Manager functions, provide the special Presentation Manager features of MS OS/2, such as windows, message queues, and device-independent graphics. The **Dev**, **Gpi**, and **Win** functions represent three distinct function groups. As described in the following list, programs use these function groups to carry out specific tasks:

Function group	Usage
Dev	Use the Presentation Manager device (Dev) functions to open and control Presentation Manager device drivers. These functions let you create device contexts that you can associate with a presentation space and use with the Gpi functions to carry out device-independent graphics for displays, printers, and plotters.
Gpi	Use the graphics programming interface (Gpi) functions to create graphics output for a display, printer, and other output devices. The Gpi functions give you a full range of graphics primitives, from lines to complex curves to bitmaps. You choose the attributes for the primitives, such as color, line width, and pattern, and then draw lines, text, and shapes. The retained-graphics capability lets you save the drawing in segments and build complex pictures by drawing a chain of segments.
Win	Use the window-manager (Win) functions to create and manage windows. Presentation Manager applications use windows as the main interface with the user. The Win functions let you create menus, scroll bars, and dialog windows that let the user choose commands and supply input. Your application receives all mouse and keyboard input as messages from the message queue. The Win functions let you retrieve messages from the queue and dispatch them to the window the input is intended for.

This chapter gives complete syntax, purpose, and parameter descriptions for each function. Types, macros, and structures used by a function are given with the function; these are described more fully in Chapter 4, "Types, Macros, Structures." The numeric values for error values returned by the functions are listed in Appendix A, "Error Values."

Many of the function descriptions in this chapter include examples. The examples show how to use the functions to accomplish simple tasks. In nearly all cases, the examples are code fragments, not complete programs. A code fragment is intended to show the context in which a function can be used, but often assumes that variables, structures, and constants used in the example have been defined and/or initialized. Also, a code fragment may use comments to represent a task instead of giving the actual statements.

Although the examples are not complete, you can still use them in your

programs if you take the following steps:

- Include the *os2.h* file in your program.
- Define the appropriate include constants for the functions, structures, and constants used in the example.
- Define and initialize all variables.
- Replace comments that represent tasks with appropriate statements.
- Check return values for errors and take appropriate actions.

2.2 Functions

The following is a complete list, in alphabetical order, of the MS OS/2 Dev, Gpl, and Win functions.

■ DevCloseDC

HMF DevCloseDC(*hdc*)

HDC *hdc*; /* device-context handle */

The **DevCloseDC** function closes the specified device context. If the device context is associated with a presentation space or was created by using the **WinOpenWindowDC** function, an error occurs and the device context is not closed. This function decrements the use count (by one) of processes that have accessed the device context. The device context is deleted when the use count reaches zero.

Parameters *hdc* Identifies the device context. An error results if this parameter identifies a screen device context or is associated with a micro presentation space.

Return Value The return value is DEV_OK if the function is successful and the device context is not a metafile device context. The return value is DEV_ERROR if an error occurs. Any other return value indicates that the function closed a metafile device context and returned its handle.

See Also DevOpenDC, WinOpenWindowDC

■ DevEscape

LONG DevEscape(*hdc, cmdCode, cblnData, pblnData, pcbOutData, pbOutData*)

HDC *hdc*; /* device-context handle */
 LONG *cmdCode*; /* escape function to perform */
 LONG *cblnData*; /* size of input buffer */
 PBYTE *pblnData*; /* pointer to input buffer */
 PLONG *pcbOutData*; /* pointer to buffer for number of bytes received */
 PBYTE *pbOutData*; /* pointer to output buffer */

The **DevEscape** function allows applications to access facilities of a device not otherwise available through the applications programming interface (API). Calls to escape functions are, in general, sent to the device driver and must be understood by it.

Parameters *hdc* Identifies the device context.

cmdCode Specifies the escape function to perform. The following escape functions are currently defined:

DEVESC_QUERYYESCSUPPORT
 DEVESC_GETSCALINGFACTOR
 DEVESC_STARTDOC
 DEVESC_ENDDOC
 DEVESC_NEXTBAND
 DEVESC_ABORTDOC
 DEVESC_NEWFRAME
 DEVESC_DRAFTMODE
 DEVESC_FLUSHOUTPUT
 DEVESC_RAWDATA

Devices can define additional escape functions by using other *cmdCode* values in the following ranges:

Range	Meaning
32768–40959	Not stored in a metafile and not recorded (passed to the device driver for PM_Q_STD).
40960–49151	Stored in a metafile only (passed to the device driver for PM_Q_STD).
49152–57343	Stored in a metafile and recorded (not passed to the device driver for PM_Q_STD).
57344–65535	Recorded only (not passed to the device driver for PM_Q_STD).

cbInData Specifies the number of bytes of data in the buffer pointed to by the *pbInData* parameter.

pbInData Points to the buffer that contains the input data required for the escape function.

pcbOutData Points to the buffer that receives the number of bytes of data in the buffer pointed by the *pbOutData* parameter. If data is returned in the *pbOutData* parameter, *pcbOutData* is updated to the number of bytes of data returned.

pbOutData Points to the buffer that receives the output from this escape. If this parameter is NULL, no data is returned.

Return Value

The return value is DEV_OK if the function is successful, DEVESC_ERROR if an error occurs, or DEVESC_NOTIMPLEMENTED if the escape function is not implemented for the specified code.

Comments

The standard escape functions, or escapes, are listed as follows, with the contents of each DevEscape parameter:

The DEVESC_QUERYYESCSUPPORT escape determines whether the device driver has implemented a particular escape. The return value gives the result. This escape is not stored in a metafile or recorded.

For DEVESC_QUERYYESCSUPPORT, the DevEscape parameters contain the following information:

Parameter	Description
<i>cbInData</i>	Specifies the number of bytes pointed to by the <i>pbInData</i> parameter.
<i>pbInData</i>	Specifies the escape-code value of the escape function to be checked.
<i>pcbOutData</i>	Not used; can be set to NULL.
<i>pbOutData</i>	Not used; can be set to NULL.

The DEVESC_GETSCALINGFACTOR escape 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 the *pbOutData* parameter. For example, 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.

For DEVESC_GETSCALINGFACTOR, the DevEscape parameters contain the following information:

Parameter	Description
<i>cbInData</i>	Not used; can be set to zero.
<i>pbInData</i>	Not used; can be set to NULL.
<i>pcbOutData</i>	Points to the number of bytes of data pointed to by the <i>pbOutData</i> parameter. Upon return, this parameter is updated to the number of bytes returned.
<i>pbOutData</i>	Points to the buffer that receives the output from this escape. A structure is returned that specifies the scaling factors for the <i>x</i> and <i>y</i> axes.

The DEVESC_STARTDOC escape allows an application to indicate that a new print job is starting and that all subsequent calls to DEVESC_NEWFRAME should be spooled under the same job, until DEVESC_ENDDOC is called. This ensures that documents longer than one page are not interspersed with other jobs.

For DEVESC_STARTDOC, the DevEscape parameters contain the following information:

Parameter	Description
<i>cbInData</i>	Specifies the number of bytes pointed to by the <i>pbInData</i> parameter.
<i>pbInData</i>	Points to the null-terminated ASCII string that specifies the name of the document.
<i>pcbOutData</i>	Not used; can be set to NULL.
<i>pbOutData</i>	Not used; can be set to NULL.

The DEVESC_ENDDOC escape ends a print job started by the DEVESC_STARTDOC escape.

For DEVESC_ENDDOC, the DevEscape parameters contain the following information:

Parameter	Description
<i>cbInData</i>	Not used; can be set to zero.
<i>pbInData</i>	Not used; can be set to NULL.
<i>pcbOutData</i>	Points to the buffer that specifies the number of characters in the string pointed to by the <i>pbOutData</i> parameter. This parameter should be NULL if the number of characters is zero.
<i>pbOutData</i>	Points to the USHORT value that specifies the job identifier if a spooler print job was created.

The DEVESC_NEXTBAND escape allows an application to signal that it has finished writing to a "band," or rectangle. The coordinates of the next band are returned. This escape is used by applications that perform handle banding ("for-printing") themselves.

For DEVESC_NEXTBAND, the DevEscape parameters contain the following information:

Parameter	Description
<i>cbInData</i>	Not used; can be set to zero.
<i>pbInData</i>	Not used; can be set to NULL.
<i>pcbOutData</i>	Points to the number of bytes of data pointed to by the <i>pbOutData</i> parameter. Upon return, the escape updates this parameter to the number of bytes returned.
<i>pbOutData</i>	Points to the address of the buffer that receives the output from this escape. A structure is returned that specifies the device coordinates of the next band, which is a rectangle.

The DEVESC_ABORTDOC escape stops the current job, erasing everything written by the application to the device since the DEVESC_ENDDOC escape was called.

For DEVESC_ABORTDOC, the DevEscape parameters contain the following information:

Parameter	Description
<i>cbInData</i>	Not used; can be set to zero.
<i>pbInData</i>	Not used; can be set to NULL.
<i>pcbOutData</i>	Not used; can be set to NULL.
<i>pbOutData</i>	Not used; can be set to NULL.

The DEVESC_NEWFRAME escape allows an application to signal when it has finished writing to a page. You usually use this escape with a printer device to advance to a new page. Calling this escape, which is similar to processing the GpiErase function for a screen device context, resets the screen attributes.

For DEVESC_NEWFRAME, the DevEscape parameters contain the following information:

Parameter	Description
<i>cbInData</i>	Not used; can be set to zero.
<i>pbInData</i>	Not used; can be set to NULL.
<i>pcbOutData</i>	Not used; can be set to NULL.
<i>pbOutData</i>	Not used; can be set to NULL.

The DEVESC_DRAFTMODE escape turns draft mode on or off. Turning draft mode on instructs the device driver to print faster and with lower quality. You can change the draft mode only at page boundaries—for example, after a call to the DEVESC_NEWFRAME escape.

For `DEVESC_DRAFTMODE`, the `DevEscape` parameters contain the following information:

Parameter	Description
<i>cbInData</i>	Specifies the number of bytes pointed to by the <i>pbInData</i> parameter.
<i>pbInData</i>	Points to the <code>SHORT</code> value that specifies the draft mode; 1 for on, 0 for off.
<i>pcbOutData</i>	Not used; can be set to <code>NULL</code> .
<i>pbOutData</i>	Not used; can be set to <code>NULL</code> .

The `DEVESC_FLUSHOUTPUT` escape removes any output from the device buffer.

For `DEVESC_FLUSHOUTPUT`, the `DevEscape` parameters contain the following information:

Parameter	Description
<i>cbInData</i>	Not used; can be set to zero.
<i>pbInData</i>	Not used; can be set to <code>NULL</code> .
<i>pcbOutData</i>	Not used; can be set to <code>NULL</code> .
<i>pbOutData</i>	Not used; can be set to <code>NULL</code> .

The `DEVESC_RAWDATA` escape allows an application to send “raw,” or binary, data directly to a device driver. For example, in the case of a printer device driver, the data could be a stream of printer data.

If binary data is mixed with other data—for example, `Gpi` data—being sent to the same page of a device context, the results are unpredictable and depend upon the action taken by the Presentation Manager device driver, which, might even ignore the `Gpi` data completely. In general, you should send binary data either to a separate page, using the `DEVESC_NEWFRAME` escape to obtain a new page, or to a separate document, using the `DEVESC_STARTDOC` and `DEVESC_ENDDOC` escapes to create a new document.

For `DEVESC_RAWDATA`, the `DevEscape` parameters contain the following information:

Parameter	Description
<i>cbInData</i>	Specifies the number of bytes pointed to by the <i>pbInData</i> parameter.
<i>pbInData</i>	Points to the binary data.
<i>pcbOutData</i>	Not used; can be set to <code>NULL</code> .
<i>pbOutData</i>	Not used; can be set to <code>NULL</code> .

See Also

`GpiErase`

■ DevOpenDC

HDC DevOpenDC (*hab*, *type*, *pszToken*, *count*, *pbData*, *hdcComp*)

HAB *hab*; /* anchor-block handle */

LONG *type*; /* type of device context */

PSZ *pszToken*; /* pointer to device-information token */

LONG *count*; /* number of elements in structure */

PDEVOPENDATA *pbData*; /* pointer to structure for device context */

HDC *hdcComp*; /* handle of compatible device context */

The **DevOpenDC** function creates a device context. This function initializes the use count (to one) of the number of processes that have access to the device context.

Parameters

hab Identifies the anchor block.

type Specifies the type of device context; it can be one of the following:

Value	Meaning
OD_QUEUED	A device, such as a printer or plotter, for which to queue output.
OD_DIRECT	A device, such as a printer or plotter, for which to not queue output.
OD_INFO	Same as for OD_DIRECT, but used only to retrieve information (for example, font metrics). You can draw to a presentation space associated with such a device context, but you cannot update any output.
OD_METAFILE	A device context that is used to draw a metafile. The graphics field defines the area of interest within the metafile picture.
OD_MEMORY	A device context that is used to contain a bitmap.

pszToken Points to the null-terminated string that contains the device-information token. This device information, which is held in the *os2.ini* file, is the same as that which may be pointed to by the *pbData* parameter; any information obtained from *pbData* overrides the information obtained by using this parameter. If you specify an asterisk (*) for *pszToken*, no device information is taken from the *os2.ini* file. MS OS/2 version 1.1 acts as if "*" is specified but allows you to specify any string.

count Specifies the number of elements in the structure pointed to by the *pbData* parameter. This number may be less than the number of items in the full list if omitted items are irrelevant or are supplied from the *pszToken* parameter or elsewhere.

pbData Points to a data area that describes the output device. This area can be either an array of pointers or a **DEVOPENSTRUC** structure, which has the following form:

```

typedef struct _DEVOPENSTRUC {
    PSZ      pszLogAddress;
    PSZ      pszDriverName;
    PDRIVDATA pdriv;
    PSZ      pszDataType;
    PSZ      pszComment;
    PSZ      pszQueueProcName;
    PSZ      pszQueueProcParams;
    PSZ      pszSpoolerParams;
    PSZ      pszNetworkParams;
} DEVOPENSTRUC;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

hdcComp Identifies the compatible device context. When the *type* parameter is `OD_MEMORY`, this parameter identifies a device context that is compatible with the bitmaps to be used with it. If you do not supply this parameter or if it is `NULL`, the device context that MS OS/2 opens is compatible with the screen.

Return Value The return value identifies the device context if the function is successful. The return value is `DEV_ERROR` if an error occurs.

■ DevPostDeviceModes

LONG DevPostDeviceModes(*hab*, *pbDriverData*, *pszDriverName*, *achDeviceName*, *pszLogAddr*)

HAB *hab*; /* anchor-block handle */
PDRIVDATA *pbDriverData*; /* pointer to buffer for data */
PSZ *pszDriverName*; /* pointer to string for driver name */
PSZ *achDeviceName*; /* pointer to device name */
PSZ *pszLogAddr*; /* pointer to string for name of output device */

The **DevPostDeviceModes** function causes a device driver to post a dialog box that allows the user to set options for the device—for example, resolution, font cartridges, and so forth.

The application can call this function first with a `NULL` data pointer to find out how much storage is needed for the data buffer. Having allocated the storage, the application can then call the function a second time in order to have the buffer filled with data.

Once the data has been returned, you can pass it to the **DevOpenDC** function as the buffer data pointed to by the *pbDriverData* parameter.

Parameters *hab* Identifies the anchor block.

pbDriverData Points to the data buffer that receives device data defined by the driver. If this parameter is `NULL`, the function returns the required size of the buffer. The format of the data is the same as for the *pbData* parameter of the **DevOpenDC** function.

pszDriverName Points to the null-terminated string that contains the name of the device driver.

achDeviceName Points to a null-terminated string that identifies the particular device (model number, etc.). This string must not exceed 32 bytes. Valid names are defined by device drivers.

pszLogAddr Points to the null-terminated string that contains the logical address of the output device—for example, `LPT1`.

Return Value The return value if the *pbDriverData* parameter is NULL is the size (in bytes) required for the data buffer, DPDM_NONE if there are no options that can be set, or DPDM_ERROR if an error occurs.

The return value if *pbDriverData* is not NULL is DEV_OK if the function is successful, DPDM_NONE if there is no device mode, or DPDM_ERROR if an error occurs.

See Also DevOpenDC

■ DevQueryCaps

BOOL DevQueryCaps(*hdc, lStartitem, cItems, allItems*)

HDC *hdc*; /* device-context handle */
LONG *lStartitem*; /* first item to retrieve */
LONG *cItems*; /* number of items to retrieve */
PLONG *allItems*; /* array for device characteristics */

The **DevQueryCaps** function queries the characteristics of the specified device.

Parameters *hdc* Identifies the device context.

lStartitem Specifies the first item of information to retrieve.

cItems Specifies the number of items of information to retrieve.

allItems Points to an array of device characteristics, starting with the item specified by the *lStartitem* parameter. For more information about device characteristics, see Appendix B, "Device Capabilities."

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also DevOpenDC

■ DevQueryDeviceNames

BOOL DevQueryDeviceNames(*hab, pszDriverName, pcMaxNames, achDeviceName, achDeviceDesc, pcMaxDataTypes, achDataType*)

HAB *hab*; /* anchor-block handle */
PSZ *pszDriverName*; /* pointer to string for device name */
PLONG *pcMaxNames*; /* maximum number of device drivers */
PSTR32 *achDeviceName*; /* pointer to array of device names */
PSTR64 *achDeviceDesc*; /* pointer to array of device descriptions */
PLONG *pcMaxDataTypes*; /* maximum number of data types */
PSTR16 *achDataType*; /* pointer to array of data types */

The **DevQueryDeviceNames** function returns the device names, descriptions, and data types supported by the specified device driver.

The application can call the function first with the *pcMaxNames* and *pcMaxDataTypes* parameters set to zero in order to find how much storage is

needed for the data buffers. Having allocated the storage, the application then calls the function a second time in order to have the buffers filled with data for the data to be filled in.

Parameters

hab Identifies the anchor block.

pszDriverName Points to the null-terminated string that contains the name of the device driver.

pcMaxNames Points to the maximum number of device names and descriptions that can be returned. If this parameter is zero, the number of device names and descriptions supported is returned and the arrays pointed to by the *achDeviceName* and *achDeviceDesc* parameters are not updated. If this parameter is nonzero, then its value is updated to the number returned in the arrays pointed to by *achDeviceName* and *achDeviceDesc* and the arrays are updated.

achDeviceName Points to an array of null-terminated strings, each element of which identifies a particular device (for example, model number). Valid names are defined by device drivers. IBM4201 is an example of a device name.

achDeviceDesc Points to an array of null-terminated strings, each element of which is a description of a particular device (for example, model name). Valid names are defined by device drivers. IBM 4201 Proprinter is an example of a device description.

pcMaxDataTypes Points to the maximum number of data types that can be returned. If this parameter is zero, the number of data types supported is returned and the array pointed to by the *achDataType* parameter is not updated. If this parameter is nonzero, then its value is updated to the number returned and the array is updated.

achDataType Points to an array of null-terminated strings, each element of which identifies a data type. Valid data types are defined by device drivers. PM_Q_STD is an example of a data type.

Return Value

The return value is TRUE if the function is successful or FALSE if an error occurs.

■ DevQueryHardcopyCaps

```
LONG DevQueryHardcopyCaps(hdc, iStartForm, cForms, phci)
```

```
HDC hdc;           /* device-context handle */
```

```
LONG iStartForm;  /* index of form code to start from */
```

```
LONG cForms;      /* number of forms to query */
```

```
PHCINFO phci;    /* pointer to structure for results */
```

The **DevQueryHardcopyCaps** function returns information about the hardcopy capabilities of a device.

You can use the *iStartForm* and *cForms* parameters together to enumerate all available form codes without having to allocate a buffer large enough to hold information on them all.

Parameters

hdc Identifies the device context.

iStartForm Specifies the index of the form code from which to start the query. The first form code is specified by the value 0.

cForms Specifies the number of forms to query.

phci Points to the buffer that contains the results of the query. The result consists of *cForms* copies of the **HCINFO** structure. The **HCINFO** structure has the following form:

```
typedef struct _HCINFO {
    CHAR szFormname[32];
    LONG cx;
    LONG cy;
    LONG xLeftClip;
    LONG yBottomClip;
    LONG xRightClip;
    LONG yTopClip;
    LONG xPels;
    LONG yPels;
    LONG flAttributes;
} HCINFO;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

If there are five form codes defined and the *iStartForm* parameter is 2 and the *cForms* parameter is 3, a query is performed for form codes 2, 3, and 4 and the result is returned in the buffer pointed to by the *phci* parameter.

The return value if *cForms* is zero is the number of available forms, or if *cForms* is nonzero, the number of forms returned. The return value is **DQHC_ERROR** if an error occurs.

■ GpiAssociate

BOOL GpiAssociate (*hps, hdc*)

HPS *hps*; /* presentation-space handle */

HDC *hdc*; /* device-context handle */

The **GpiAssociate** function associates a presentation space with a device context and resets the presentation space. Once a device context is associated with a presentation space, all subsequent drawing in the presentation space is copied to the device.

Only one device context can be associated with a presentation space at a time. **GpiAssociate** cannot associate a new device context with a presentation space until the current device context is released. The function releases the current device context from the presentation space if *hdc* is NULL.

Parameters *hps* Identifies a normal presentation space. Micro and cached presentation spaces cannot be used.

hdc Identifies the device context. Although any type of device context may be used, the device context must not be already associated with a presentation space.

Return Value The return value is **GPL_OK** if the function is successful or **GPL_ERROR** if an error occurred.

Errors Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

```
PMERR_DC_IS_ASSOCIATED
PMERR_INV_MICROPS_FUNCTION
PMERR_PS_IS_ASSOCIATED
```

Comments When **GpiAssociate** resets the presentation space, it sets all attributes to their default values, sets the model transform to unity, sets the current position to (0,0), closes any open path, area, or element brackets, closes any open segment, removes any clip path, viewing limits and clip region, and enables kerning if the device supports kerning.

The **GpiCreatePS** function can also be used to associate a device context with a new presentation space.

Example This example releases the current device context and associates a new device context with the presentation space.

```
HPS hps;
HDC hdcPrinter;
GpiAssociate(hps, NULL); /* release the current device context */
GpiAssociate(hps, hdcPrinter); /* associate a printer device context */
```

See Also **GpiCreatePS**, **GpiResetPS**

■ GpiBeginArea

BOOL GpiBeginArea(*hps*, *fOptions*)

HPS *hps*; /* presentation-space handle */

ULONG *fOptions*; /* area-option flag */

The **GpiBeginArea** function starts an area bracket, that is, it starts a sequence of functions that define the shape of an area. All subsequent drawing functions, up to the next **GpiEndArea** function, apply to the new area. The *fOptions* parameter specifies whether the figures in the area have boundary lines and which filling mode to use for constructing the interior of the area.

Parameters *hps* Identifies the presentation space.

fOptions Specifies the area options. It can be any combination of the following values:

Value	Meaning
BA_ALTERNATE	Constructs the interior in alternate mode (default).
BA_BOUNDARY	Boundary lines are drawn.
BA_NOBOUNDARY	Boundary lines are not drawn (default).
BA_WINDING	Constructs the interior in winding mode.

Return Value The return value is **GPL_OK** or **GPL_HITS** if the function is successful (it is **GPL_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is **GPL_ERROR** if an error occurred.

Errors Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

```
PMERR_ALREADY_IN_AREA
PMERR_INV_AREA_CONTROL
PMERR_INV_IN_PATH
```

Example This example uses the **GpiBeginArea** function to draw an area. The area, an isosceles triangle, is drawn with boundary lines and filled using the alternate filling mode.

```
HPS hps;
POINTL ptlStart = { 0, 0 };
POINTL ptlTriangle[] = { 100, 100, 200, 0, 0, 0 };
GpiMove(hps, &ptlStart); /* move to starting point (0, 0) */
GpiBeginArea(hps, /* start the area bracket */
             BA_BOUNDARY | /* draw boundary lines */
             BA_ALTERNATE); /* fill interior with alternate mode */
GpiPolyLine(hps, 3L, ptlTriangle); /* draw the triangle */
GpiEndArea(hps); /* end the area bracket */
/* and fill the area */
```

See Also **GpiEndArea**

■ GpiBeginElement

BOOL GpiBeginElement(*hps*, *lType*, *pszDesc*)

HPS *hps*; /* presentation-space handle */

LONG *lType*; /* element type */

PSZ *pszDesc*; /* pointer to element description */

The **GpiBeginElement** function starts an element bracket, that is, a sequence of functions that define the contents of an element. All subsequent graphics functions, up to the next **GpiEndElement** or **GpiCloseSegment** function, apply to the new element.

The **GpiBeginElement** may only be used while creating a segment. The element type and element description, specified by the *lType* and *pszDesc* parameters, are values that the application supplies to distinguish one element from another within a segment.

Parameters *hps* Identifies the presentation space.

lType Specifies the element type. It can be any integer value.

pszDesc Points to a null-terminated string. If no description is needed, it may point to an empty string.

Return Value The return value is **GPL_OK** if the function is successful or **GPL_ERROR** if an error occurred.

Errors Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

PMERR_ALREADY_IN_ELEMENT
PMERR_INV_MICROPS_FUNCTION

Comments The **GpiBeginElement** function cannot be used within an element bracket.

Example This example uses the **GpiBeginElement** function to create an element in a segment. The element type is 1 and the element description is "Triangle". The application can use these later to identify the element.

```
POINTL ptlStart = { 0, 0 };
POINTL ptlTriangle[] = { 100, 100, 200, 0, 0, 0 };
GpiBeginElement(hps, /* start element bracket */
                1L, /* element type is 1 */
                "Triangle"); /* element description */
GpiMove(hps, &ptlStart); /* move to start point (0, 0) */
GpiPolyLine(hps, 3L, ptlTriangle); /* draw triangle */
GpiEndElement(hps); /* end element bracket */
```

See Also **GpiCloseSegment**, **GpiDeleteElement**, **GpiEndElement**, **GpiQueryElement**, **GpiQueryElementPointer**, **GpiSetElementPointer**

■ GpiBeginPath

BOOL GpiBeginPath(*hps*, *idPath*)

HPS *hps*; /* presentation-space handle */

LONG *idPath*; /* path identifier */

The **GpiBeginPath** function starts a path bracket, that is, starts a sequence of functions that define the shape and size of a path. **GpiBeginPath** sets the path

identifier and initializes the path, clearing any path created previously with this identifier. All subsequent drawing functions, up to the next **GpiEndPath** function, apply to the new path.

Parameters *hps* Identifies the presentation space.

idPath Specifies the path identifier; for MS OS/2 version 1.1, it must be 1.

Return Value The return value is **GPL_OK** if the function is successful or **GPL_ERROR** if an error occurred.

Errors Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

PMERR_ALREADY_IN_PATH
PMERR_INV_PATH_ID

Comments Since there is a limit to the size of a path, any line or curve drawing function within a path bracket that would exceed the limit returns the **PMERR_PATH_TOO_BIG** error value.

Example This example uses the **GpiBeginPath** function to create a path. The path, an isosceles triangle, is given path identifier 1. After the path bracket is ended using **GpiEndPath**, a subsequent call to the **GpiFillPath** function draws and fills the path.

```
HPS hps;
POINTL ptlStart = { 0, 0 };
POINTL ptlTriangle[] = { 100, 100, 200, 0 };
GpiBeginPath(hps, 1L);
GpiMove(hps, &ptlStart);
GpiPolyLine(hps, 2L, ptlTriangle);
GpiCloseFigure(hps);
GpiEndPath(hps);
GpiFillPath(hps, 1L, FPATH_ALTERNATE);
```

/* start the path bracket */
/* move to starting point */
/* draw the three sides */
/* close the triangle */
/* end the path bracket */
/* draw and fill the path */

See Also **GpiCloseFigure**, **GpiEndPath**, **GpiFillPath**, **GpiModifyPath**, **GpiSetClipPath**, **GpiSetLineWidthGeom**, **GpiStrokePath**

■ GpiBitBlt

LONG GpiBitBlt(*hpsTarg*, *hpsSrc*, *cPoints*, *aptl*, *lRop*, *flOptions*)

HPS *hpsTarg*; /* target presentation-space handle */
HPS *hpsSrc*; /* source presentation-space handle */
LONG *cPoints*; /* number of points in array */
PPOINTL *aptl*; /* pointer to array */
LONG *lRop*; /* mixing method */
ULONG *flOptions*; /* line/column-compression flag */

The **GpiBitBlt** function copies a bitmap from one presentation space to another. It can also modify the bitmap within a rectangle in a presentation space. The exact operation carried out by **GpiBitBlt** depends on the raster operation specified by the *lRop* parameter.

If *lRop* directs **GpiBitBlt** to copy a bitmap, the function copies the bitmap from a source presentation space specified by *hpsSrc* to a target presentation space specified by *hpsTarg*. Each presentation space must be associated with a device context for the display, for memory, or for some other suitable raster device. The target and source presentation spaces can be the same if desired. The *aptl*

parameter points to an array of points that specify the corners of a rectangle containing the bitmap in the source presentation space as well as the corners of the rectangle in the target presentation space to receive the bitmap. If the source and target rectangles are not the same, **GpiBitBlit** stretches or compresses the bitmap to fit the target rectangle.

If *IRop* directs **GpiBitBlit** to modify a bitmap, the function uses the raster operation to determine how to alter the bits in a rectangle in the target presentation space. Raster operations include changes such as inverting target bits, replacing target bits with pattern bits, and mixing target and pattern bits to create new colors. For some raster operations, the function mixes the bits of a bitmap from a source presentation space with the target and/or pattern bits.

Parameters

hpsTarg Identifies the target presentation space.

hpsSrc Identifies the source presentation space.

cPoints Specifies the number of points pointed to by the *aptl* parameter. It may be one of the following values:

Value	Meaning
2	The points specify the lower-left and upper-right corners of the target rectangle. If 2 is given, the raster operation specified by the <i>IRop</i> parameter must not include a source.
3	The points specify the lower-left and upper-right corners of the target rectangle, and the lower-left corner of the source rectangle. The upper-right corner of the source rectangle is computed such that the target and source rectangles have equal width and height. Any raster operation may be used. If the operation does not include a source, the third point is ignored.
4	The points specify the lower-left and upper-right corners of the target and the source rectangles. If the rectangles do not have equal width and height, the source bitmap is stretched or compressed to fit the target rectangle. GpiBitBlit uses the <i>fOptions</i> parameter to determine how the bitmap should be compressed. If the raster operation does not include a source, the source coordinates are ignored.

aptl Points to an array of **POINTL** structures containing the number of points specified in the *cPoints* parameter. The points must be given in the following order:

Element index	Coordinate
0	Specifies the lower-left corner of the target rectangle.
1	Specifies the upper-right corner of the target rectangle.
2	Specifies the lower-left corner of the source rectangle.
3	Specifies the upper-right corner of the source rectangle.

All points must be in device coordinates. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

lRop Specifies the raster operation for the function. It can be any value in the range 0 through 255 or one of the following values, which represent common raster operations:

Value	Meaning
ROP_DSTINVERT	Inverts the target.
ROP_MERGECOPY	Combines the source and the pattern using the bitwise AND operator.
ROP_MERGEPAINT	Combines the inverse of the source and the target using the bitwise OR operator.
ROP_NOTSRCCOPY	Copies the inverse of the source to the target.
ROP_NOTSRCERASE	Combines the inverse of the source and the inverse of the target bitmaps using the bitwise AND operator.
ROP_ONE	Sets all target pels to 1.
ROP_PATCOPY	Copies the pattern to the target.
ROP_PATINVERT	Combines the target and the pattern using the bitwise exclusive XOR operator.
ROP_PATPAINT	Combines the inverse of the source, the pattern, and target using the bitwise OR operator.
ROP_SRCAND	Combines the source and target bitmaps using the bitwise AND operator.
ROP_SRCCOPY	Copies the source bitmap to the target.
ROP_SRCERASE	Combines the source and the inverse of the target bitmaps using the bitwise AND operator.
ROP_SRCINVERT	Combines the source and target bitmaps using the bitwise exclusive OR operator.
ROP_SRCPAINT	Combines the source and target bitmaps using the bitwise OR operator.
ROP_ZERO	Sets all target pels to 0.

fOptions Specifies how to compress a bitmap if the target rectangle is smaller than the source. It can be one of the following values:

Value	Meaning
BBO_AND	Compresses two rows or columns into one by combining them with the bitwise AND operator. This value is useful for compressing bitmaps
BBO_IGNORE	Compresses two rows or columns by throwing one out. This value is useful for compressing color bitmaps. that have black images on a white background.

Value	Meaning
BBO_OR	Compresses two rows or columns into one by combining them with the bitwise OR operator. This value is the default and is useful for compressing bitmaps that have white images on a black background.

All values in the range 0x0100 through 0xFF00 are reserved for privately supported modes for particular devices.

Return Value

The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

Errors

Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_INCOMPATIBLE_BITMAP
PMERR_INV_BITBLT_MIX
PMERR_INV_BITBLT_STYLE
PMERR_INV_COORDINATE
PMERR_INV_DC_TYPE
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT

```

Comments

The source and target presentation spaces may be associated with any device context having raster capabilities. Some raster devices, such as banded printers, can receive bitmaps but cannot supply them. These devices cannot be used as a source.

`GpiBitBlt` does not affect the pels in the upper and right boundaries of the target rectangle. This means the function draws up to but does not include those pels.

If *IRop* includes a pattern, `GpiBitBlt` uses the current area color, area background color, pattern set, and pattern symbol of the target presentation space. Although the function may stretch or compress the bitmap, it never stretches or compresses the pattern.

If the target and source presentation spaces are associated with device contexts that have different color formats, `GpiBitBlt` converts the bitmap color format as it copies the bitmap. This applies to bitmaps copied to or from a device context having a monochrome format. To convert a monochrome bitmap to a color bitmap, `GpiBitBlt` converts 1 pels to the target's foreground color, and 0 pels to the current area background color. To convert a color bitmap to a monochrome bitmap, `GpiBitBlt` converts pels with the source's background color to the target's background color, and all other pels to the target's foreground color.

The bitmap associated with a source presentation space is always a finite size. Although `GpiBitBlt` will copy a bitmap when given a source rectangle that is larger than the source bitmap or extends past the boundaries of the source bitmap, any pels not associated with the source bitmap are undefined.

Example

This example uses `GpiBitBlt` to copy and compress a bitmap in a presentation space. The function copies the bitmap that is 100 pels wide and 100 pels high into a 50-by-50-pel rectangle at the location (300,400). Since the raster operation

is ROP_SRCCOPY, GpiBitBlt replaces the image previously in the target rectangle. The function compresses the bitmap to fit the new rectangle by discarding extra rows and columns as specified by the BBO_IGNORE option.

```
HPS hps;
POINTL aptl[4] = {
    300, 400, /* lower-left corner of target */
    350, 450, /* upper-right corner of target */
    0, 0, /* lower-left corner of source */
    100, 100 }; /* upper-right corner of source */

GpiBitBlt(hps, /* target presentation space */
hps, /* source presentation space */
4L, /* four points needed to compress */
aptl, /* points to source and target */
ROP_SRCCOPY, /* copy source replacing target */
BBO_IGNORE); /* discard extra rows and columns */
```

See Also

DevOpenDC, GpiCreateBitmap, GpiLoadBitmap, GpiSetBitmap, GpiSetBitmapDimension, GpiSetBitmapId, GpiWCBitBlt

■ GpiBox

```
LONG GpiBox(hps, cmdControl, pptl, IHRound, IVRound)
HPS hps; /* presentation-space handle */
LONG cmdControl; /* fill and outline indicator */
PPOINTL pptl; /* pointer to structure for box corners */
LONG IHRound; /* horizontal length of rounding-ellipse axis */
LONG IVRound; /* vertical length of rounding-ellipse axis */
```

The GpiBox function draws a rectangular box or a box with rounded corners. The function draws the box by drawing the outline of a rectangle. The current position specifies one corner and the point given by *pptl* specifies the other. The sides of the box are always parallel to the *x* and *y* axes. The function may fill the interior with the current fill pattern. If a rounded box is requested, the function rounds the corners of the rectangle using quarter ellipses. The *IHRound* and *IVRound* parameters specify the lengths of the major and minor axes for the ellipse. If either the *IHRound* or the *IVRound* parameter is zero, no rounding occurs.

The current position is unchanged by this function.

Parameters

hps Identifies the presentation space.

cmdControl Specifies whether to draw the box's interior and/or outline. It can be one of the following values:

Value	Meaning
DRO_FILL	Fills the interior.
DRO_OUTLINE	Draws the outline.
DRO_OUTLINEFILL	Draws the outline and fills the interior.

pptl Points to the POINTL structure that contains the coordinates of a corner of the box. The POINTL structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

IHRound Specifies the horizontal length (in world coordinates) of the full axis of the ellipse used for rounding at each corner.

IVRound Specifies the vertical length (in world coordinates) of the full axis of the ellipse used for rounding at each corner.

Return Value The return value is `GPL_OK` or `GPL_HITS` if the function is successful (it is `GPL_HITS` if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is `GPL_ERROR` if an error occurs.

Errors Use the `WinGetLastError` function to retrieve the error value, which may be the following value:

`PMERR_INV_BOX_CONTROL`

Comments `GpiBox` can be used in an area bracket but only if `DRO_OUTLINE` is used. If the current position is (x_0, y_0) , and *pptl* is set to (x_1, y_1) , the box is drawn from (x_0, y_0) to (x_1, y_0) to (x_1, y_1) to (x_0, y_1) to (x_0, y_0) . This can affect the way the box is filled when drawn in an area.

When correlating a segment, a box drawn using `GpiBox` will be “hit” if the box boundary intersects in the pick aperture. If the pick aperture lies within the box, a hit occurs only if the interior is drawn, that is, only if the `DRO_FILL` or `DRO_OUTLINEFILL` option is used.

Example This example calls `GpiBox` to draw a series of rounded boxes, one inside another.

```
POINTL pt1 = { 100, 100 };
SHORT i;

for (i = 0; i < 5; i++)
    GpiBox(hps,
           DRO_OUTLINE,
           (POINTL) &pt1,
           i * 10L,
           i * 10L);
           /* handle to a ps          */
           /* fill the interior      */
           /* address of the corner  */
           /* horizontal length     */
           /* vertical length       */
```

See Also `GpiBeginArea`, `GpiEndArea`

■ GpiCallSegmentMatrix

LONG GpiCallSegmentMatrix(*hps*, *idSegment*, *cElements*, *pmatlf*, *IType*)

HPS *hps*; /* presentation-space handle */
LONG *idSegment*; /* segment identifier */
LONG *cElements*; /* number of matrix elements to examine */
PMATRIXLF *pmatlf*; /* pointer to structure for matrix */
LONG *IType*; /* transformation modifier */

The `GpiCallSegmentMatrix` function draws the specified segment using an instance transformation. The function combines the instance transformation pointed to by *pmatlf* with the current model transformation, then draws the segment as if calling the `GpiDrawSegment` function. The combined transformation applies only while the function draws the segment. `GpiCallSegmentMatrix` does not modify the current model transformation.

Parameters

hps Identifies the presentation space.

idSegment Specifies the segment to draw. This value must be greater than zero.

cElements Specifies the number of matrix elements pointed to by *pmatlf*. It can be any value from 0 through 9.

pmatlf Points to a **MATRIXLF** structure that contains the matrix for the instance transformation. Although a transformation requires nine matrix elements, the function copies from the structure only the number of matrix elements specified by *cElements*. If *cElements* is less than nine, the function supplies the remaining elements by substituting corresponding elements from the identity matrix. The **MATRIXLF** structure has the following form:

```
typedef struct _MATRIXLF {
    FIXED fxM11;
    FIXED fxM12;
    LONG  lM13;
    FIXED fxM21;
    FIXED fxM22;
    LONG  lM23;
    LONG  lM31;
    LONG  lM32;
    LONG  lM33;
} MATRIXLF;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

lType Specifies how to combine the instance transformation with the model transformation. It can be one of the following values:

Value	Meaning
TRANSFORM_ADD	Adds the model transformation to the instance transformation (MODEL * INSTANCE).
TRANSFORM_PREEMPT	Adds the instance transformation to the model transformation (INSTANCE * MODEL).
TRANSFORM_REPLACE	Replaces the model transform with the instance transformation.

Return Value The return value is **GPL_OK** or **GPL_HITS** if the function is successful (it is **GPL_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is **GPL_ERROR** if an error occurs.

Errors Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

```
PMERR_CALLED_SEG_IS_CURRENT
PMERR_CALLED_SEG_NOT_FOUND
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_SEG_NAME
PMERR_INV_TRANSFORM_TYPE
PMERR_SEG_CALL_RECURSIVE
```

Example This example calls the **GpiCallSegmentMatrix** function to draw a segment three times. Each time the segment is drawn, the instance transformation doubles in size. The result is three triangles with the last triangle twice the size of the second, and the second twice the size of the first.

```

POINTL ptlStart = { 0, 0 };
POINTL ptlTriangle[] = { 100, 100, 200, 0, 0, 0 };
MATRIXLF matlfInstance = { 1, 0, 0, 0, 0, 1, 0, 0, 0, 1 };

GpiOpenSegment(hps, 1L);          /* open the segment */
GpiMove(hps, &ptlStart);         /* move to start point (0, 0) */
GpiPolyLine(hps, 3L, ptlTriangle); /* draw the triangle */
GpiCloseSegment(hps);           /* close the segment */

for (i = 0; i < 3; i++) {
    /*
     * Draw the segment after adding the matrix to the model
     * transformation.
     */
    GpiCallSegmentMatrix(hps, 1L, 9, &matlfInstance, TRANSFORM_ADD);
    matlfInstance.fxM11 *= 2;
    matlfInstance.fxM22 *= 2;
}

```

See Also **GpiDrawSegment**

■ GpiCharString

LONG GpiCharString(*hps, cchString, pchString*)

HPS *hps*; /* presentation-space handle */
LONG *cchString*; /* number of characters in string */
PCH *pchString*; /* pointer to string to draw */

The **GpiCharString** function draws a character string positioned at the current position. After the function draws the string, it sets the current position to the end of the character string.

Parameters *hps* Identifies the presentation space.

cchString Specifies the number of characters in the string pointed to by *pchString*.

pchString Points to the character string to be drawn.

Return Value The return value is **GPL_OK** or **GPL_HITS** if the function is successful (it is **GPL_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is **GPL_ERROR** if an error occurs.

Example This example uses the **GpiCharString** function to draw the string "Hello". The **GpiMove** function moves the current position to (100,100) so that the string starts there.

```

HPS hps;
POINTL ptlStart;

ptlStart.x = 100L;
ptlStart.y = 100L;

/* Start string at (100, 100). */
GpiMove(hps, &ptlStart);

/* Draw the 5-character string. */
GpiCharString(hps, 5L, "Hello");

```

See Also **GpiCharStringAt, GpiCharStringPos**

■ GpiCharStringAt

```
LONG GpiCharStringAt(hps, pptlStart, cchString, pchString)
HPS hps;          /* presentation-space handle */
PPOINTL pptlStart; /* pointer to structure for starting position */
LONG cchString;   /* number of characters in string */
PCH pchString;   /* pointer to string to draw */
```

The **GpiCharStringAt** function draws a character string starting at the specified position. After the function draws the string, it sets the current position to the end of the character string.

Parameters *hps* Identifies the presentation space.

pptlStart Points to the **POINTL** structure that contains the starting position in world coordinates. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

cchString Specifies the number of characters pointed to by *pchString*.

pchString Points to the character string to be drawn.

Return Value The return value is **GPL_OK** or **GPL_HITS** if the function is successful (it is **GPL_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is **GPL_ERROR** if an error occurs.

Example The example uses the **GpiCharStringAt** function to draw the string “Hello” starting at the position (100,100). It then uses the **GpiMove** and **GpiCharString** functions to draw the same string at exactly the same position.

```
HPS hps;
POINTL ptlStart;

ptlStart.x = 100L;
ptlStart.y = 100L;

/* Draw the string "Hello" at (100, 100). */
GpiCharStringAt(hps, &ptlStart, 5, "Hello");

/* These two calls are identical to the one above. */
GpiMove(hps, &ptlStart);
GpiCharString(hps, 5L, "Hello");
```

See Also **GpiCharString**, **GpiMove**

■ GpiCharStringPos

```
LONG GpiCharStringPos(hps, prcl, flOptions, cchString, pchString, adx)
HPS hps;          /* presentation-space handle */
PRECTL prcl;      /* pointer to structure for rectangle coordinates */
ULONG flOptions;  /* formatting flags */
LONG cchString;   /* number of characters in string */
PCH pchString;   /* pointer to string to draw */
PLONG adx;       /* pointer to array of increment values */
```

The **GpiCharStringPos** function draws a character string starting at the current position and using one or more formatting options. The options direct the function to draw a background for the string, clip the string to the given rectangle, or position the characters in the string using distances given in an array. After drawing the string, the function either leaves the current position at the end of the string or resets it to the beginning of the string.

Parameters

hps Identifies the presentation space.

prcl Points to a **RECTL** structure that contains the lower-left and upper-right corners of a rectangle. The function draws the rectangle if the **CHS_OPAQUE** option is given. It uses the rectangle to clip the string if the **CHS_CLIP** option is given. Otherwise the rectangle is ignored. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

flOptions Specifies the formatting options. It can be one or more of the following values:

Value	Meaning
CHS_CLIP	Clips the string to the rectangle, omitting any portion of any character outside the rectangle. The function clips the string regardless of whether CHS_OPAQUE is specified.
CHS_LEAVEPOS	Resets the current position back to the start of the string. If not given, GpiCharStringPos moves the current position to the end of string.
CHS_OPAQUE	Draws the rectangle whose lower-left and upper-right corners are specified by <i>prcl</i> , then fills the rectangle with the current background color. The string is drawn after filling the rectangle.
CHS_VECTOR	Advances the current position after each character is drawn by using the next value in the array <i>adx</i> . The current character direction defines which direction the current position is advanced.

All other values are reserved.

cchString Specifies the number of characters in the string pointed to by *pchString*.

pchString Points to the character string to be drawn.

adx Points to an array of increment values. Each value is a 4-byte, signed integer specifying the distance in world coordinates to advance the current position after drawing a character. There must be one value for each character in the string. The first element specifies the distance to advance after drawing the first character, the second element specifies the distance after the second character, and so on.

Return Value The return value is `GPL_OK` or `GPL_HITS` if the function is successful (it is `GPL_HITS` if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is `GPL_ERROR` if an error occurs.

Comments If `CHS_OPAQUE` is specified and the drawing mode is `DM_RETAIN`, `GpiCharStringPos` uses the color mix mode `BM_OVERPAINT` to fill the rectangle. In other drawing modes, the function uses the `BM_LEAVEALONE`. `GpiCharStringPos` draws the rectangle using the coordinates specified in *prcl*. It does not use the start of the string to compute the rectangle's location.

See Also `GpiCharString`, `GpiCharStringAt`, `GpiCharStringPosAt`

■ GpiCharStringPosAt

LONG GpiCharStringPosAt(*hps*, *pptlStart*, *prcl*, *fOptions*, *cchString*, *pchString*, *adx*)

```

HPS hps;           /* presentation-space handle */
PPOINTL pptlStart; /* pointer to structure for starting position */
PRECTL prcl;       /* pointer to structure for rectangle coordinates */
ULONG fOptions;    /* formatting flags */
LONG cchString;    /* number of characters in string */
PCH pchString;    /* pointer to string to draw */
PLONG adx;        /* increment vector */

```

The `GpiCharStringPosAt` function draws a character string starting at the specified position and using one or more formatting options. The options direct the function to draw a background for the string, clip the string to the given rectangle, or position the characters in the string using distances given in an array. After drawing the string, the function either leaves the current position at the end of the string or resets it to the beginning of the string.

Parameters *hps* Identifies the presentation space.

pptlStart Points to a `POINTL` structure that contains the starting position in world coordinates. The `POINTL` structure has the following form:

```

typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;

```

For a full description, see Chapter 4, "Types, Macros, Structures."

prcl Points to a `RECTL` structure that contains the lower-left and upper-right corners of a rectangle. The function draws the rectangle if the `CHS_OPAQUE` option is given. It uses the rectangle to clip the string if the `CHS_CLIP` option is given. Otherwise the rectangle is ignored. The `RECTL` structure has the following form:

```
typedef struct _RECTL {
    LONG xLeft;
    LONG yBottom;
    LONG xRight;
    LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

fOptions Specifies the formatting options. It can be one or more of the following values:

Value	Meaning
CHS_CLIP	Clips the string to the rectangle, omitting any portion of any character outside the rectangle. The function clips the string regardless of whether CHS_OPAQUE is specified.
CHS_LEAVEPOS	Resets the current position back to the start of the string. If not given, GpiCharStringPos moves the current position to the end of string.
CHS_OPAQUE	Draws the rectangle whose the lower-left and upper-right corners are specified by <i>prcl</i> , then fills the rectangle with the current background color. The string is drawn after filling the rectangle.
CHS_VECTOR	Advances the current position after each character is drawn by using the next value in the array <i>adx</i> . The current character direction defines which direction the current position is advanced.

All other values are reserved.

cchString Specifies the number of characters in the string pointed to by *pchString*.

pchString Points to the character string to be drawn.

adx Points to an array of increment values. Each value is a 4-byte, signed integer specifying the distance in world coordinates to advance the current position after drawing a character. There must be one value for each character in the string. The first element specifies the distance to advance after drawing the first character, the second element specifies the distance after the second character, and so on.

Return Value	The return value is <code>GPL_OK</code> or <code>GPL_HITS</code> if the function is successful (it is <code>GPL_HITS</code> if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is <code>GPL_ERROR</code> if an error occurs.
Comments	If <code>CHS_OPAQUE</code> is specified and the drawing mode is <code>DM_RETAIN</code> , GpiCharStringPosAt uses the color mix mode <code>BM_OVERPAINT</code> to fill the rectangle. In other drawing modes, the function uses the <code>BM_LEAVEALONE</code> . GpiCharStringPos draws the rectangle using the coordinates specified in <i>prcl</i> . It does not use the start of the string to compute the rectangle's location.
See Also	GpiCharStringPos

■ GpiCloseFigure

BOOL GpiCloseFigure(*hps*)

HPS *hps*; /* presentation-space handle */

The **GpiCloseFigure** function closes an open figure in a path bracket. A figure is open unless it is explicitly closed by using the **GpiCloseFigure** function. A figure can be open even if the current point and the starting point of the figure are equal.

Parameters *hps* Identifies the presentation space.

Return Value The return value is **GPL_OK** if the function is successful or **GPL_ERROR** if an error occurred.

Example This example uses the **GpiCloseFigure** function to close a triangle drawn in a path bracket. The triangle starts at (0,0). Since the current position just before the **GpiCloseFigure** is (200,0), the function closes the triangle by drawing a line from (200,0) to (0,0).

```
HPS hps;
POINTL ptlStart = { 0, 0 };
POINTL ptlPoints[] = { 100, 100, 200, 0 };

GpiBeginPath(hps, 1L);                               /* start the path bracket */
GpiMove(hps, &ptlStart);                             /* move to starting point */
GpiPolyLine(hps, 2L, ptlPoints);                     /* draw the three sides */
GpiCloseFigure(hps);                                 /* close the triangle */
GpiEndPath(hps);                                     /* end the path bracket */
```

See Also **GpiBeginPath, GpiEndPath**

■ GpiCloseSegment

BOOL GpiCloseSegment(*hps*)

HPS *hps*; /* presentation-space handle */

The **GpiCloseSegment** function closes the current segment. Closing a segment does not delete the segment or affect output on the current device. If any element bracket is open, **GpiCloseSegment** automatically closes it.

Parameters *hps* Identifies the presentation space.

Return Value The return value is **GPL_OK** if the function is successful or **GPL_ERROR** if an error occurred.

Errors Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

```
PMERR_AREA_INCOMPLETE
PMERR_IMAGE_INCOMPLETE
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_MODE_FOR_REOPEN_SEG
PMERR_PATH_INCOMPLETE
```

Comments You must explicitly end any area or path bracket before closing the segment. Failing to end an area or path may invalidate the segment.

GpiCloseSegment resets the current viewing transformation to identity.

Example

This example uses the `GpiCloseSegment` function to close a segment. The `GpiOpenSegment` opens the segment; `GpiMove` and `GpiPolyLine` draw a triangle.

```
POINTL pt1Start = { 0, 0 };
POINTL pt1Triangle[] = { 100,100, 200,0, 0,0 };

GpiOpenSegment(hps, 1L);           /* open the segment */
GpiMove(hps, &pt1Start);          /* move to start point (0,0) */
GpiPolyLine(hps, 3L, pt1Triangle); /* draw triangle */
GpiCloseSegment(hps);            /* close the segment */
```

See Also

`GpiOpenSegment`

■ GpiCombineRegion

LONG GpiCombineRegion(*hps, hrgnDest, hrgnSrc1, hrgnSrc2, cmdMode*)

HPS *hps*; /* presentation-space handle */
HRGN *hrgnDest*; /* handle of destination region */
HRGN *hrgnSrc1*; /* handle of first source region */
HRGN *hrgnSrc2*; /* handle of second source region */
LONG *cmdMode*; /* combination method */

The `GpiCombineRegion` function combines two source regions identified by *hrgnSrc1* and *hrgnSrc2*. The new region replaces the destination region identified by *hrgnDest*. If one of the source regions is also given as the destination region, the function replaces that source region with the new region, but does not affect the other source region.

Parameters

hps Identifies the presentation space. The presentation space must be associated with a device context.

hrgnDest Identifies the destination region.

hrgnSrc1 Identifies the first source region.

hrgnSrc2 Identifies the second source region.

cmdMode Specifies how to combine the source regions. It can be one of the following values:

Value	Meaning
CRGN_AND	Creates the intersection of the source regions (<i>hrgnSrc1</i> INTERSECT <i>hrgnSrc2</i>). The new region contains only the parts of the source regions that are common.
CRGN_COPY	Copies the first source region to the destination. The function does not use the <i>hrgnSrc2</i> parameter.
CRGN_DIFF	Creates the difference of the source region (<i>hrgnSrc1</i> INTERSECT NOT <i>hrgnSrc2</i>). The new region contains the parts of the first source region that are not also in the second region.
CRGN_OR	Creates the union of the two source regions (<i>hrgnSrc1</i> UNION <i>hrgnSrc2</i>). The new region contains all parts of both source regions.
CRGN_XOR	Creates the “symmetric” difference of the source regions (<i>hrgnSrc1</i> - <i>hrgnSrc2</i>). The new region contains only the parts of the source regions that are not common.

Return Value The return value is RGN_NULL, RGN_RECT, or REGN_COMPLEX if the function is successful. The return value is RGN_ERROR if an error occurred.

Errors Use the `WinGetLastError` function to retrieve the error value, which may be the following:

PMERR_INV_REGION_MIX

Comments The source and destination regions must belong to the same presentation space or to presentation spaces associated with a similar device context.

Example This example uses the `GpiCombineRegion` function to create a complex region consisting of everything in two rectangles except where they overlap.

```
HRCN hrgn1, hrgn2, hrgn3;
RECTL rclRect1 = { 0, 0, 100, 100 };
RECTL rclRect2 = { 50, 50, 200, 200 };

hrgn1 = GpiCreateRegion(hps, 1L, &rclRect1); /* create first region */
hrgn2 = GpiCreateRegion(hps, 1L, &rclRect2); /* create second region */
hrgn3 = GpiCreateRegion(hps, 0L, NULL);    /* create empty region */

/* Combine first and second regions, replacing the empty region. */
GpiCombineRegion(hrgn3, hrgn1, hrgn2, CRGN_XOR);
```

See Also `GpiCreateRegion`

■ **GpiComment**

BOOL GpiComment (*hps, cbData, pbData*)

HPS *hps*; /* presentation-space handle */
LONG *cbData*; /* length of comment string */
PBYTE *pbData*; /* pointer to the comment string */

The `GpiComment` function adds a comment string to a segment.

Parameters *hps* Identifies the presentation space.
cbData Specifies the length in bytes of the comment string pointed to by *pbData*.
pbData Points to the comment string. The string must not be longer than 255 bytes.

Return Value The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

Example This example uses the `GpiComment` function to comment the contents of a segment.

```
POINTL ptlStart = { 0, 0 };
POINTL ptlTriangle[] = { 100, 100, 200, 0, 0, 0 };

GpiOpenSegment(hps, 0L); /* open the segment */
GpiComment(hps, 18L, "Start point (0, 0)");
GpiMove(hps, &ptlStart);
GpiComment(hps, 13L, "Draw triangle");
GpiPolyLine(hps, 3L, ptlTriangle);
GpiCloseSegment(hps); /* close the segment */
```

See Also `GpiCloseSegment`, `GpiMove`, `GpiOpenSegment`, `GpiPolyLine`

■ GpiConvert

BOOL GpiConvert(*hps, lSrc, lTarg, cPoints, aptl*)

HPS *hps*; /* presentation-space handle */
LONG *lSrc*; /* source coordinate space */
LONG *lTarg*; /* target coordinate space */
LONG *cPoints*; /* number of coordinate pairs in structure */
POINTL *aptl*; /* pointer to structure for coordinate pairs */

The **GpiConvert** function converts one or more points from one coordinate space to another. For each **POINTL** structure in the array pointed to by *aptl*, the function replaces the original *x*- and *y*-coordinate values with the converted values.

Parameters

hps Identifies the presentation space.

lSrc Specifies the source coordinate space. It can be one of the following values:

Value	Meaning
CVTC_DEFAULTPAGE	Page space prior to default viewing transform
CVTC_DEVICE	Device space
CVTC_MODEL	Model space
CVTC_PAGE	Page space after default viewing transform
CVTC_WORLD	World coordinates

lTarg Specifies the target coordinate space. It can be one of the following values:

Value	Meaning
CVTC_DEFAULTPAGE	Page space prior to default viewing transform
CVTC_DEVICE	Device space
CVTC_MODEL	Model space
CVTC_PAGE	Page space after default viewing transform
CVTC_WORLD	World coordinates

cPoints Specifies the number of coordinate pairs pointed to by *aptl*.

aptl Points to an array of **POINTL** structures containing the coordinate pairs. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

The return value is **GPL_OK** if the function is successful or **GPL_ERROR** if an error occurred.

See Also

GpiSetModelTransformMatrix, **GpiSetPageViewport**, **GpiSetSegmentTransformMatrix**, **GpiSetViewingTransformMatrix**

■ GpiCopyMetaFile

HMF GpiCopyMetaFile(*hmfSrc*)

HMF *hmfSrc*; /* handle of source metafile */

The **GpiCopyMetaFile** function creates a copy of the metafile identified by *hmfSrc* and returns a handle for the new metafile. The new metafile can be edited or deleted without affecting the original metafile.

Parameters *hmfSrc* Identifies the source metafile. The source metafile must have been loaded previously using the **GpiLoadMetaFile** function or created previously using the **DevOpenDC** and **DevCloseDC** functions.

Return Value The return value is the handle of the new metafile if the function is successful, or it is **GPL_ERROR** if an error occurred.

Example This example uses the **GpiCopyMetaFile** function to copy make a copy of the metafile loaded using the **GpiLoadMetaFile** function.

```
HMF hmf1, hmf2;
```

```
GpiLoadMetaFile(hmf1, "sample.met"); /* load the metafile from disk */
hmf2 = GpiCopyMetaFile(hmf1);      /* copy the metafile */
```

See Also **DevCloseDC**, **DevOpenDC**, **GpiLoadMetaFile**

■ GpiCorrelateChain

LONG GpiCorrelateChain(*hps*, *lType*, *pptl*, *lMaxHits*, *lMaxDepth*, *alSegTag*)

HPS *hps*; /* presentation-space handle */
 LONG *lType*; /* segment type */
 PPOINTL *pptl*; /* pointer to structure for aperture center */
 LONG *lMaxHits*; /* maximum number of hits */
 LONG *lMaxDepth*; /* maximum number of segment/tag pairs to return */
 PLONG *alSegTag*; /* pointer to array of segment and tag identifiers */

The **GpiCorrelateChain** function correlates the segment chain, identifying each tagged primitive that intersects the current aperture, as set by the **GpiSetPick-ApertureSize** function.

The **GpiCorrelateChain** function correlates a segment chain by searching for each tagged primitive in each segment that lies completely or partially within the aperture. Each instance of a tagged primitive in the aperture is called a "hit." The function records a hit by copying the identifier of the segment containing the primitive (along with the identifier for its tag) to the array pointed to by *alSegTag*. After searching all segments in the chain, **GpiCorrelateChain** returns the number of hits it located.

Parameters *hps* Identifies the presentation space.
lType Specifies the type of segment to correlate. It can be one of the following values:

Value	Meaning
PICKSEL_ALL	Correlate all segments with nonzero identifiers regardless of the detectability and visibility attributes of the segments.
PICKSEL_VISIBLE	Correlate visible and detectable segments with nonzero identifiers.

pptl Points to the POINTL structure that contains the position (in presentation page units) of the center of the aperture. The POINTL structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

lMaxHits Specifies the maximum number of hits to record.

lMaxDepth Specifies the maximum number of segment/tag pairs to record for each hit.

alSegTag Points to the array to receive the segment/tag pairs. The array must be large enough to receive $8 \times lMaxHits \times lMaxDepth$ bytes.

Return Value

The return value is the number of hits that occurred if the function is successful or `GPI_ERROR` if an error occurred.

Errors

Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_AREA_INCOMPLETE
PMERR_IMAGE_INCOMPLETE
PMERR_INV_CORRELATE_DEPTH
PMERR_INV_CORRELATE_TYPE
PMERR_INV_MAX_HITS
PMERR_INV_MICROPS_FUNCTION
PMERR_PATH_INCOMPLETE
```

Comments

`GpiCorrelateChain` may record more than one segment for each hit. It first records the segment containing the hit, then the segment that called the first segment, and so on until the function either records the original segment in this chain or has recorded *lMaxDepth* segments. If the function finds less than *lMaxDepth* segments for the hit, the function records zeros so that exactly *lMaxDepth* records are copied for each hit. The function records all hits up to *lMaxHits*, then continues to count the hits even though it no longer records them. The return value specifies the complete number of hits, not just those recorded.

The function searches only segments that have nonzero identifiers. If the function encounters a segment with a zero identifier, it stops the search even if subsequent segments in the chain have nonzero identifiers. During the search, the function ignores primitives that do not have nonzero identifiers. The function never records more than one hit for a tag in a segment even if that tag is used with many primitives.

See Also

`GpiCorrelateFrom`, `GpiCorrelateSegment`, `GpiSetPickApertureSize`

■ GpiCorrelateFrom

LONG GpiCorrelateFrom(*hps, idFirstSegment, idLastSegment, IType, pptl, IMaxHits, IMaxDepth, aSegTag*)

HPS *hps*; /* presentation-space handle */
LONG *idFirstSegment*; /* first segment to correlate */
LONG *idLastSegment*; /* last segment to correlate */
LONG *IType*; /* segment type */
PPOINTL *pptl*; /* pointer to structure for aperture center */
LONG *IMaxHits*; /* maximum number of hits */
LONG *IMaxDepth*; /* maximum number of segment/tag pairs to return */
PLONG *aSegTag*; /* pointer to array of segment and tag identifiers */

The **GpiCorrelateFrom** function correlates a portion of the segment chain, identifying each tagged primitive that intersects the current aperture, as set by the **GpiSetPickApertureSize** function.

The **GpiCorrelateFrom** function correlates a portion of the segment chain by searching for each tagged primitive that lies completely or partially within the aperture. Each instance of a tagged primitive in the aperture is called a “hit.” The function records a hit by copying the identifier of the segment containing the primitive (along with the identifier for its tag) to the array pointed to by *aSegTag*. The function starts the search with the segment identified by *idFirstSegment* and includes chained and called segments up to, and including, the segment identified by *idLastSegment*. After searching these segments, **GpiCorrelateFrom** returns the number of hits it located.

Parameters

hps Identifies the presentation space.

idFirstSegment Specifies the first segment to correlate. This value must be greater than zero.

idLastSegment Specifies the last segment to correlate. This value must be greater than zero.

IType Specifies the type of segment to correlate. It can be one of the following values:

Value	Meaning
PICKSEL_ALL	Correlate all segments with nonzero identifiers regardless of the detectability and visibility attributes of the segments.
PICKSEL_VISIBLE	Correlate visible and detectable segments with nonzero identifiers.

pptl Points to the **POINTL** structure that contains the position (in presentation page units) of the center of the aperture. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

IMaxHits Specifies the maximum number of hits to record.

IMaxDepth Specifies the maximum number of segment/tag pairs to record.

alSegTag Points to the array to receive the segment/tag pairs. The array must be large enough to receive $8 \times IMaxHits \times IMaxDepth$ bytes.

Return Value The return value is the number of hits that occurred if the function is successful or `GPL_ERROR` if an error occurred.

Errors Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_AREA_INCOMPLETE
PMERR_IMAGE_INCOMPLETE
PMERR_INV_CORRELATE_DEPTH
PMERR_INV_CORRELATE_TYPE
PMERR_INV_MAX_HITS
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_SEG_NAME
PMERR_PATH_INCOMPLETE
```

Comments `GpiCorrelateFrom` may record more than one segment for each hit. It first records the segment containing the hit, then the segment that called the first segment, and so on until the function either records the original segment in this chain or has recorded *IMaxDepth* segments. If the function finds less than *IMaxDepth* segments for the hit, the function records zeros so that exactly *IMaxDepth* records are copied for each hit. The function records all hits up to *IMaxHits*, then continues to count the hits even though it no longer records them. The return value specifies the complete number of hits, not just those recorded.

The function searches only segments that have nonzero identifiers. If the function encounters a segment with a zero identifier, it stops the search even if subsequently called segments have nonzero identifiers. During the search, the function ignores primitives that do not have nonzero identifiers. The function never records more than one hit for a tag in a segment even if that tag is used with many primitives.

If the *idFirstSegment* parameter does not exist, or is not in the segment chain, the function returns an error. If the segment specified by *idLastSegment* does not exist, is not in the chain, or is chained before *idFirstSegment*, no error results and the function continues to the end of the chain.

See Also `GpiCorrelateChain`, `GpiCorrelateSegment`

■ GpiCorrelateSegment

```
LONG GpiCorrelateSegment(hps, idSegment, IType, pptl, IMaxHits, IMaxDepth, alSegTag)
HPS hps;          /* presentation-space handle          */
LONG idSegment;   /* segment to correlate                */
LONG IType;       /* segment type                         */
PPOINTL pptl;    /* pointer to structure for aperture center */
LONG IMaxHits;   /* maximum number of hits              */
LONG IMaxDepth;  /* maximum number of segment/tag pairs to return */
PLONG alSegTag; /* pointer to array of segment and tag identifiers */
```

The `GpiCorrelateSegment` function correlates the specified segment, identifying each tagged primitive that intersects the current aperture, as set by the `GpiSetPickApertureSize` function.

The **GpiCorrelateSegment** function correlates a segment by searching for each tagged primitive in the segment that lies completely or partially within the aperture. Each instance of a tagged primitive in the aperture is called a "hit." The function records a hit by copying the identifier of the segment containing the primitive (along with the identifier for its tag) to the array pointed to by *alSegTag*. The function also searches segments that are called by the specified segment. After searching all segments, **GpiCorrelateSegment** returns the number of hits it located.

Parameters

hps Identifies the presentation space.

idSegment Specifies the segment to correlate. This value must be greater than zero.

lType Specifies the type of segment to correlate. It can be one of the following values:

Value	Meaning
PICKSEL_ALL	Correlate all segments with nonzero identifiers regardless of the detectability and visibility attributes of the segments.
PICKSEL_VISIBLE	Correlate visible and detectable segments with nonzero identifiers.

pptl Points to the **POINTL** structure that contains the position (in presentation page units) of the center of the aperture. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

lMaxHits Specifies the maximum number of hits to record.

lMaxDepth Specifies the maximum number of segment/tag pairs to record.

alSegTag Points to the array to receive the segment/tag pairs. The array must be large enough to receive $8 \times lMaxHits \times lMaxDepth$ bytes.

Return Value

The return value is the number of hits that occurred if the function is successful or **GPL_ERROR** if an error occurred.

Errors

Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

```
PMERR_AREA_INCOMPLETE
PMERR_IMAGE_INCOMPLETE
PMERR_INV_CORRELATE_DEPTH
PMERR_INV_CORRELATE_TYPE
PMERR_INV_MAX_HITS
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_SEG_NAME
PMERR_PATH_INCOMPLETE
```

Comments

GpiCorrelateSegment may record more than one segment for each hit. It first records the segment containing the hit, then the segment that called the first segment, and so on until the function either records the original segment in this chain or records *IMaxDepth* segments. If the function finds less than *IMaxDepth* segments for the hit, the function records zeros so that exactly *IMaxDepth* records are copied for each hit. The function records all hits up to *IMaxHits*, then continues to count the hits even though it no longer records them. The return value specifies the complete number of hits, not just those recorded.

The function searches only segments that have nonzero identifiers. If the function encounters a segment with a zero identifier, it stops the search even if subsequently called segments have nonzero identifiers. During the search, the function ignores primitives that do not have nonzero identifiers. The function never records more than one hit for a tag in a segment even if that tag is used with many primitives.

See Also

GpiCorrelateChain, **GpiCorrelateFrom**

■ GpiCreateBitmap

```
HBITMAP GpiCreateBitmap(hps, pbmpFormat, fOptions, pbData, pbmiData)
HPS hps;                /* presentation-space handle */
PBITMAPINFOHEADER pbmpFormat; /* pointer to structure for format data */
ULONG fOptions;        /* options */
PBYTE pbData;         /* pointer to buffer of image data */
PBITMAPINFO pbmiData; /* pointer to structure for color and format */
```

The **GpiCreateBitmap** function creates a bitmap and returns a bitmap handle identifying the bitmap. The new bitmap has the width, height, and format specified by fields of the structure pointed to by *pbmpFormat*. The *fOptions* parameter specifies whether to initialize the bitmap color and image. If the parameter is **CBM_INIT**, the function uses the bitmap image data pointed to by *pbData* and the bitmap color data pointed to by *pbmiData* to initialize the bitmap. If **CBM_INIT** is not given, the bitmap's initial image and color are undefined.

The bitmap handle can be used in subsequent functions that accept bitmap handles. In most cases, the bitmap is set to a memory presentation space using the **GpiSetBitmap** function, then copied to the screen or a printer using the **GpiBitBlt** function.

Parameters

hps Identifies the presentation space.

pbmpFormat Points to the **BITMAPINFOHEADER** structure that contains the bitmap format data. The **BITMAPINFOHEADER** structure has the following form:

```
typedef struct _BITMAPINFOHEADER {
    ULONG   cbFix;
    USHORT  cx;
    USHORT  cy;
    USHORT  cPlanes;
    USHORT  cBitCount;
} BITMAPINFOHEADER;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

flOptions Specifies whether to initialize the bitmap. It can one of the following values:

Value	Meaning
CBM_INIT	Initializes the bitmap, using the bitmap image and color data specified by the <i>pbData</i> and <i>pbmiData</i> parameters.
0x0000	Does not initialize the bitmap.

pbData Points to the buffer that contains bitmap image data. The image data defines the color of each pel in the bitmap. This parameter is ignored if CBM_INIT is not given.

pbmiData Points to a **BITMAPINFO** structure that contains the bitmap format and color data. The format data is identical to the data pointed to by the *pbmpFormat* parameter. The color data follows immediately after the format data, and consists of two or more RGB color values. The exact number depends on the bitmap format. This parameter is ignored if CBM_INIT is not given. The **BITMAPINFO** structure has the following form:

```
typedef struct _BITMAPINFO {
    ULONG   cbFix;
    USHORT  cx;
    USHORT  cy;
    USHORT  cPlanes;
    USHORT  cBitCount;
    RGB     argbColor[1];
} BITMAPINFO;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

The return value identifies the new bitmap if the function is successful, or is **GPL_ERROR** if an error occurred.

Errors

Use the **WinGetLastError** function to retrieve the error value, which may be the following:

PMERR_INV_USAGE

Comments

The full number of bitmap formats depends on what the associated device supports. However, most devices support the following standard bitmap formats:

Format	Description
Monochrome	1 bit per pel and 1 color plane
16-color	4 bits per pel and 1 color plane
256-color	8 bits per pel and 1 color plane
Full-color	24 bits per pel and 1 color plane

When initializing the bitmap, the bitmap color data must consist of an appropriate number of RGB color values. For monochrome format, it must have 2 values; for 16-color format, 16 values; and for 256-color format, 256 values. No color values are required for the full-color format, since the image data for each pel fully specifies the pel color.

When **CBM_INIT** is given, the function continues to copy data from the buffer until the entire bitmap is initialized. The function expects each row of image data to contain a multiple of 32 bits (4 bytes). Although the bitmap width does not have to be a multiple of 32, the image data must be. Any extra bits at the end of a row are ignored.

The new bitmap belongs to the device context associated with the given presentation space. It can be set to any presentation space having the same device context or having a compatible device context.

Example

The following example loads a bitmap resource from memory and uses the **GpiCreateBitmap** function to create the bitmap. This is similar to using the **GpiLoadBitmap** function, except it gives the application the chance to modify the bitmap image data before creating the bitmap.

```
SEL sel; /* selector for segment containing bitmap resource */
PBITMAPFILEHEADER pbfh; /* bitmap resource header information */
PBYTE pb; /* pointer to bitmap image data in resource */
HBITMAP hbm; /* bitmap handle */

DosGetResource(NULL, RT_BITMAP, 1, &sel); /* load bitmap resource #1 */
pbfh = MAKEP(sel, 0); /* bitmap file header in resource */
pb = MAKEP(sel, pbfh->offBits); /* image data starts at offBits */

/* make any changes to bitmap image data here */

hbm = GpiCreateBitmap(hps, /* presentation space */
                    &(pbfh->bmp), /* bitmap information in file */
                    CBM_INIT, /* initialize the bitmap */
                    pb, /* bitmap data */
                    &(pbfh->bmp)); /* bitmap information in file */

DosFreeSeg(sel); /* free bitmap resource */
```

See Also

DosFreeSeg, **DosGetResource**, **GpiDeleteBitmap**, **GpiLoadBitmap**, **GpiQueryDeviceBitmapFormats**

■ GpiCreateLogColorTable

BOOL GpiCreateLogColorTable (*hps*, *flOptions*, *lFormat*, *iStart*, *ciTable*, *alTable*)

HPS *hps*; /* presentation-space handle */
ULONG *flOptions*; /* options */
LONG *lFormat*; /* format of entries */
LONG *iStart*; /* starting index */
LONG *ciTable*; /* number of entries in table */
PLONG *alTable*; /* pointer to array for table */

The **GpiCreateLogColorTable** function creates a logical color table. The logical color table has the format specified by *lFormat*, with the initial value of each entry specified by the array *alTable*.

Parameters

hps Identifies the presentation space.

flOptions Specifies whether the logical color table uses pure, realizable, or default color values. It can be one of the following values:

Value	Meaning
0x0000	Creates a logical color table having the entries specified by <i>alTable</i> . The logical color table entries map to existing device colors in the physical palette or to dithered colors if no matching device color is in the palette. This means the table is not realized and does not require pure colors.

Value	Meaning
LCOL_PURECOLOR	Creates a logical color table whose entries map to pure (nondithered) colors only. If not given, the function creates a color table whose entries map to dithered colors if the physical palette does not contain matching device colors.
LCOL_REALIZABLE	Creates a logical color table that can be realized by using the <code>GpiRealizeColorTable</code> function. Until the logical color table is realized, colors in the table map to the existing device colors in the physical palette. This option is useful only for devices that permit realization of logical color tables.
LCOL_RESET	Resets all entries in the logical color table to default values before initializing the entries specified by the <i>alTable</i> parameter. This option is useful for quickly initializing all entries without supplying initial values for every element in <i>alTable</i> .

lFormat Specifies the logical color table format. It can be one of the following values:

Value	Meaning
LCOLF_CONSECRGB	Creates a color table having consecutive entries. The first entry has the index specified by <i>iStart</i> .
LCOLF_INDRGB	Creates a color table. The entries are not required to be consecutive. The <i>alTable</i> array specifies both the index and RGB color value for each entry.
LCOLF_RGB	Enables direct RGB color mapping. Applications use RGB values instead of color indexes to specify the colors in subsequent drawing functions.

iStart Specifies the color index of the first entry for a color table having LCOLF_CONSECRGB format. If LCOLF_CONSECRGB is not given, this parameter is ignored.

clTable Specifies the number of elements in the array *alTable*. If the format LCOLF_INDRGB is given, this parameter must be an even number (that is, two elements for each entry). If LCOL_RESET or LCOLF_RGB is given, this parameter can be zero.

alTable Specifies the start address of the array that contains the color table entries. The format depends on the value of *lFormat*, as follows:

Value	Format
LCOLF_CONSECRGB	Each element is a 4-byte RGB color value.
LCOLF_INDRGB	Each pair of elements contains a 4-byte color index and a 4-byte RGB color value, in that order.
LCOLF_RGB	No elements required.

Return Value The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

Errors Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_INV_COLOR_DATA
PMERR_INV_COLOR_FORMAT
PMERR_INV_COLOR_OPTIONS
```

Comments Although `GpiCreateLogColorTable` can create a realizable color table, it does not realize the colors. Until the color table is realized by using the `GpiRealizeColorTable` function, the logical color table entries are mapped to the existing colors in the physical palette. Realizing the logical color table causes the physical palette colors to be replaced with the realized colors for the logical color table entries.

The default physical palette contains at least the standard 16 PC colors (unless this is not physically possible). If a device supports more than 16 colors, the physical palette may have additional colors, but there is no guarantee that these additional colors are the same on every device.

Example This example uses the `GpiCreateLogColorTable` function to create a logical color table, using data from the previous logical color table:

```
ULONG alTable[16];                /* assume 16 entries */
/* retrieve the current table */
GpiQueryLogColorTable(hps, OL, OL, 16L, alTable);
alTable[1] = 0x000080;            /* change the second entry to light blue */
GpiCreateLogColorTable(hps,      /* presentation space */
    OL,                          /* no special options */
    LCOLF_CONSECRGB,            /* consecutive RGB values */
    OL,                          /* start with color index 0 */
    16,                          /* 16 entries */
    alTable);                   /* RGB color values */
```

See Also `DevQueryCaps`, `GpiErase`, `GpiQueryColorData`, `GpiQueryLogColorTable`, `GpiRealizeColorTable`, `GpiSetBitmapBits`, `WinSetSysColors`

■ GpiCreateLogFont

```
LONG GpiCreateLogFont(hps, pchName, lcid, pfat)
HPS hps;                /* presentation-space handle */
PSTR8 pchName;          /* pointer to logical-font name */
LONG lcid;              /* local identifier */
PFATTRS pfat;          /* pointer to structure for font attributes */
```

The `GpiCreateLogFont` function creates a logical font. A logical font is a list of font attributes, such as face name, average width, and maximum height, that an application uses to request a physical font. A physical font is the bitmap or vector information the system uses to draw characters on a device. Applications create logical fonts to specify the fonts they need, and the system maps the logical fonts to matching physical fonts.

`GpiCreateLogFont` creates a logical font using the font attributes specified in the structure pointed to by the `pfat` parameter. Each logical font has a local identifier and logical font name, specified by the `lcid` and `pchName` parameters, to uniquely identify it. The local identifier can then be used in subsequent graphics functions to identify the font.

Since a physical font that exactly matches the logical font may not be available, the system usually maps the logical font to the closest matching physical font. The system uses rules to map the font—for example, it chooses a font with a greater height if a font of the exact height is not available. An application can force the system to choose a particular font by setting the value of the `lMatch` field in the `FATTRS` structure to be that returned for the desired font by the `GpiQueryFonts` function. After `GpiCreateLogFont` chooses the physical font, this choice does not change for a particular logical font.

Parameters

hps Identifies the presentation space.

pchName Points to an 8-character logical-font name. It can be `NULL`, if no logical font name is desired.

lcid Specifies 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 this parameter is already in use to refer to a font or bitmap.

pfat Points to a `FATTRS` structure that will contain the attributes of the logical font that is created. The `FATTRS` structure has the following form:

```
typedef struct _FATTRS {
    USHORT  usRecordLength;
    USHORT  fsSelection;
    LONG    lMatch;
    CHAR    szFaceName[FACE_SIZE];
    USHORT  idRegistry;
    USHORT  usCodePage;
    LONG    lMaxBaselineExt;
    LONG    lAveCharWidth;
    USHORT  fsType;
    SHORT   sQuality;
    USHORT  fsFontUse;
} FATTRS;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

The return value is 2 if a matching font is found, 1 if a matching font could not be found, or zero if an error occurred.

Errors

Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_FONT_NOT_LOADED
PMERR_INV_FONT_ATTRS
```

Comments

To choose the system default font, set the face name to `NULL` and all other attributes in the `FATTR` structure, except the code page, to zero.

To use a font, the application sets the font for the presentation space by specifying the local identifier for the corresponding logical font with the `GpiSetCharSet` function. Once a font is set, the system uses the font for subsequent text output.

Example

This example uses the `GpiCreateLogFont` function to create a logical font with the local identifier 1. The logical font has the face name “Courier” and requested width and height of 12 pels. Once the font is created, the example sets the font using the local identifier and displays a string in the font at the point (100,100).

```

USHORT i;
POINTL ptl = { 100, 100 };
FATTRS fat;

fat.usRecordLength = sizeof(FATTRS); /* set size of structure */
fat.fsSelection = 0; /* use default selection */
fat.lMatch = 0L; /* do not force match */
fat.idRegistry = 0; /* use default registry */
fat.usCodePage = 850; /* code page 850 */
fat.lMaxBaselineExt = 12L; /* requested font height is 12 pels */
fat.lAveCharWidth = 12L; /* requested font width is 12 pels */
fat.fsType = FATTR_TYPE_FIXED; /* fixed-spacing font */
fat.fsFontUse = FATTR_FONTUSE_NOMIX; /* do not mix with graphics */

/* copy Courier to szFacename field */
for (i=0; fat.szFacename[i] = "Courier"[i]; i++);

GpiCreateLogFont(hps, /* presentation space */
                NULL, /* do not use logical font name */
                1L, /* local identifier */
                &fat); /* structure with font attributes */

GpiSetCharSet(hps, 1L); /* set font for presentation space */
GpiCharStringAt(hps, &ptl, 5L, "Hello"); /* display a string */

```

See Also

GpiCharStringAt, GpiCreateLogFont, GpiQueryFonts, GpiSetCharSet

■ GpiCreatePS

HPS GpiCreatePS(*hab*, *hdc*, *psizl*, *flOptions*)

HAB *hab*; /* anchor-block handle */
HDC *hdc*; /* device-context handle */
PSIZEL *psizl*; /* pointer to structure for page size */
ULONG *flOptions*; /* presentation-space options */

The **GpiCreatePS** function creates a presentation space. The presentation space has the presentation type, page size, page unit, and storage format specified by *psizl* and *flOptions*. The function also associates the device context specified by *hdc* with the presentation space if a device context is given. The presentation space, identified by the handle returned by **GpiCreatePS**, can be used in subsequent **Gpi** functions to draw to the associated device.

Parameters

hab Identifies the anchor block.

hdc Identifies a device context. It is required only if the **GPIA_ASSOC** option is given in *flOptions*. It must be a handle to a device context if the **GPI_MICRO** option is given. Otherwise, it can be **NULL**.

psizl Points to a **SIZEL** structure that contains the width and height of the presentation page. The width and height can be zero if the **GPIA_ASSOC** option is given. The width and height must be non-zero if the **PU_ARBITRARY** option is given. The **SIZEL** structure has the following form:

```

typedef struct _SIZEL {
    LONG cx;
    LONG cy;
} SIZEL;

```

For a full description, see Chapter 4, "Types, Macros, Structures."

flOptions Specifies the presentation-space options. The options define the page unit, storage format, and presentation type for the presentation space, as

well as specifying whether to associate a device context with the new presentation space. The *flOptions* parameter must include exactly one of the following page unit options combined with no more than one each of the following storage format, presentation type, and association options:

Page unit	Meaning
PU_ARBITRARY	Sets units initially to pels but permits the units to be modified later using the <code>GpiSetPageViewport</code> function.
PU_HIENGLISH	Sets units to 0.001 inch.
PU_HIMETRIC	Sets units to 0.01 millimeter.
PU_LOENGLISH	Sets units to 0.01 inch.
PU_LOMETRIC	Sets units to 0.1 millimeter.
PU_PELS	Sets units to pels.
PU_TWIPS	Sets units to 1/1440 inch (1/20 point).
Storage format	Meaning
GPIF_DEFAULT	Stores coordinates as 2-byte integers. GPIF_DEFAULT is the default if no storage format is given.
GPIF_LONG	Stores coordinates as 4-byte integers.
GPIF_SHORT	Stores coordinates as 2-byte integers.
Presentation type	Meaning
GPIT_MICRO	Creates a micro presentation space. The presentation space must be associated with a screen device context. The GPIA_ASSOC option and a device context must also be given.
GPIT_NORMAL	Creates a normal presentation space. The presentation space can be associated with any device context and used with retained graphics. If a presentation-space type is not given, the default is GPIT_NORMAL.
Association	Meaning
GPIA_ASSOC	Associates the device context specified by <i>hdc</i> with the new presentation space. If <i>hdc</i> identifies a memory device context, GPIT_MICRO must be set or the system will issue a warning.
GPIA_NOASSOC	Creates presentation space without associating a device context. GPIA_NOASSOC is the default if an association option is not given.

Return Value The return value is the handle of the presentation space if the function is successful or `GPI_ERROR` if an error occurred.

Errors Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_OR_INCOMPAT_OPTIONS`

Comments

The presentation type can be normal or micro. Normal presentation spaces can be associated with any device context and can be used for retained graphics. Micro presentation spaces can be associated with any device, but only when they are created. They can never be reassociated. The `GPIA_ASSOC` and `GPIA_NOASSOC` options specify whether the new presentation space is to be associated with the device context identified by *hdc*. If not associated, the `GpiAssociate` function must be used to associate a device context. A presentation space can not be used without an associated device.

The page unit specifies the unit of measure used to draw to the device. For example, if the page unit is pels, a line 100 units long in world space coordinates is 100 pels long on the device.

The presentation page size specifies the width and height of the presentation page. The presentation page and page viewport define how points in the presentation page space are mapped to the pels in the device space. This is important for programs that need to change the page unit without recreating the presentation space.

The storage format specifies the internal format for coordinate values stored in the segments. This is important for applications that edit segments.

Example

This example uses the `GpiCreatePS` function to create a micro presentation space for a memory device context. The function associates the presentation space with the device context and sets the page units to pels. By default, the presentation space is a normal presentation space that uses local storage format.

```
HDC hdc;
HPS hps;
SIZEL sizl = { 0, 0 }; /* use same page size as device */
DEVOPENSTRUC dop;

dop.pszLogAddress = NULL;
dop.pszDriverName = (PSZ) "DISPLAY";
dop.pdriv = NULL;
dop.pszDataType = NULL;

/* Create the memory device context. */
hdc = DevOpenDC(hab, OD_MEMORY, "*", 4L, &dop, NULL);

/* Create the presentation and associate the memory device context. */
hps = GpiCreatePS(hab, hdc, &sizl, PU_PELS | GPIT_MICRO | GPIA_ASSOC);
```

See Also

`GpiDestroyPS`, `GpiSetPageViewport`

■ GpiCreateRegion

```
HRGN GpiCreateRegion(hps, crcl, prcl)
HPS hps; /* presentation-space handle */
LONG crcl; /* number of rectangles */
PRECTL prcl; /* pointer to structure for rectangles */
```

The `GpiCreateRegion` function creates a region for the device associated with the specified presentation space. The region is the union of the rectangles specified by the *prcl* parameter.

Parameters

hps Identifies the presentation space.

crcl Specifies the number of rectangles specified in the *prcl* parameter. If the *crcl* parameter is equal to zero, an empty region is created, and *prcl* is ignored.

prcl Points to an array of **RECTL** structures. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG xLeft;
    LONG yBottom;
    LONG xRight;
    LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

The return value is a handle to the region if the function is successful or zero if an error occurred. It is an error if this function is issued when there is no device context associated with the presentation space.

Example

This example uses the **GpiCreateRegion** function to create a region consisting of the union of three rectangles:

```
HRGN hrgn;
RECTL arcl[3] = { 100, 100, 200, 200, /* handle for region */
                 150, 150, 250, 250, /* 1st rectangle */
                 200, 200, 300, 300 }; /* 2nd rectangle */
                                     /* 3rd rectangle */

hrgn = GpiCreateRegion(hps, /* presentation space */
                      3L, /* three rectangles */
                      arcl); /* pointer to array of rectangles */
```

See Also

GpiCombineRegion, **GpiDestroyRegion**

■ GpiDeleteBitmap

BOOL GpiDeleteBitmap(*hbm*)

HBITMAP *hbm*; /* bitmap handle */

The **GpiDeleteBitmap** function deletes the bitmap specified by *hbm*.

Parameters

hbm Identifies the bitmap to delete.

Return Value

The return value is **GPI_OK** if the function is successful or **GPI_ERROR** if an error occurred.

Errors

Use the **WinGetLastError** function to retrieve the error value, which may be the following:

PMERR_BITMAP_IS_SELECTED

Example

This example uses the **GpiDeleteBitmap** function to delete a bitmap. The **GpiSetBitmap** function releases the bitmap from the presentation space before deleting it. This is needed only if the bitmap is set in the presentation space.

```
HBITMAP hbm, hbmPrevious;

hbm = GpiLoadBitmap(hps, NULL, 1, OL, OL); /* load the bitmap */
hbmPrevious = GpiSetBitmap(hps, hbm); /* set bitmap for PS */

/* use GpiBitBlt to display bitmap */

GpiSetBitmap(hps, hbmPrevious); /* release bitmap from PS */
GpiDeleteBitmap(hbm); /* delete the bitmap */
```

See Also

GpiCreateBitmap, **GpiLoadBitmap**, **GpiQueryDeviceBitmapFormats**, **GpiSetBitmap**

■ GpiDeleteElement

BOOL GpiDeleteElement(*hps*)

HPS *hps*; /* presentation-space handle */

The **GpiDeleteElement** function deletes an element from the currently open segment. The function deletes the element pointed to by the element pointer, then moves the element pointer to the preceding element (if any). The segment containing the element must be open and the drawing mode must be **DM_RETAIN**.

GpiDeleteElement cannot be used in an element bracket.

Parameters *hps* Identifies the presentation space.

Return Value The return value is **GPI_OK** if the function is successful or **GPI_ERROR** if an error occurred.

Errors Use the **WinGetLastError** function to retrieve the error value, which may be the following:

PMERR_INV_MICROPS_FUNCTION

Example This example uses the **GpiDeleteElement** function to delete the third element from the previously created segment 2:

```
GpiOpenSegment(hps, 2L);          /* open segment #2      */
GpiSetElementPointer(hps, 3L);    /* move to third element */
GpiDeleteElement(hps);           /* delete element       */
GpiCloseSegment(hps);           /* close the segment    */
```

See Also **GpiBeginElement**, **GpiEndElement**, **GpiQueryElement**, **GpiQueryElementPointer**, **GpiSetElementPointer**

■ GpiDeleteElementRange

BOOL GpiDeleteElementRange(*hps, idFirstElement, idLastElement*)

HPS *hps*; /* presentation-space handle */

LONG *idFirstElement*; /* first element */

LONG *idLastElement*; /* last element */

The **GpiDeleteElementRange** function deletes one or more elements from the currently open segment. The function deletes all elements between and including the elements specified by *idFirstElement* and *idLastElement*, then moves the element pointer to the preceding element (if any). The function rounds *idFirstElement* or *idLastElement* to a valid element-pointer position if the given position does not point to an element. The segment containing the element must be open and the drawing mode must be **DM_RETAIN**.

GpiDeleteElementRange cannot be used in an element bracket.

Parameters *hps* Identifies the presentation space.

idFirstElement Specifies the element-pointer position of the first element to delete.

idLastElement Specifies the element-pointer position of the last element to delete.

- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:
- `PMERR_INV_MICROPS_FUNCTION`
- Example** This example uses the `GpiDeleteElementRange` function to delete the second through fifth elements in the previously created segment 2:
- ```
GpiOpenSegment(hps, 2L); /* open segment # 2 */
GpiDeleteElementRange(hps, 2L, 5L); /* delete elements 2 through 5 */
GpiCloseSegment(hps); /* close the segment */
```
- See Also** `GpiOffsetElementPointer`, `GpiQueryElementPointer`, `GpiSetElementPointer`

## ■ **GpiDeleteElementsBetweenLabels**

**BOOL GpiDeleteElementsBetweenLabels** (*hps, idFirstLabel, idLastLabel*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *idFirstLabel*; /\* label of first element \*/  
**LONG** *idLastLabel*; /\* label of last element \*/

The `GpiDeleteElementsBetweenLabels` function deletes one or more elements from the currently open segment. The function deletes all elements between but not including the elements having the labels specified by the *idFirstLabel* and *idLastLabel* parameters, then moves the element pointer to the element preceding the deleted elements (if any). If either label cannot be found between the current element-pointer position and the end of the segment, the function deletes no elements and returns an error value. The segment containing the element must be open and the drawing mode must be `DM_RETAIN`.

`GpiDeleteElementBetweenLabels` cannot be used in an element bracket.

- Parameters** *hps* Identifies the presentation space.  
*idFirstLabel* Specifies the label that marks the start of the elements to delete.  
*idLastLabel* Specifies the label that marks the end of the elements to delete.

**Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

`PMERR_INV_MICROPS_FUNCTION`  
`PMERR_LABEL_NOT_FOUND`

**Example** This example uses the `GpiDeleteElementsBetweenLabels` function to delete the elements between but not including the elements having the labels 1 and 2:

```
GpiOpenSegment(hps, 2L); /* open segment #2 */
/* delete elements between 1 and 2 */
GpiDeleteElementsBetweenLabels(hps, 1L, 2L);
GpiCloseSegment(hps); /* close the segment */
```

**See Also** `GpiLabel`, `GpiSetElementPointerAtLabel`

## ■ GpiDeleteMetaFile

**BOOL GpiDeleteMetaFile** (*hmf*)

**HMF** *hmf*; /\* metafile handle \*/

The **GpiDeleteMetaFile** function deletes the metafile specified by *hmf*.

**Parameters**     *hmf*   Identifies the metafile.

**Return Value**   The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**See Also**        **DevCloseDC**, **DevOpenDC**, **GpiLoadMetaFile**

## ■ GpiDeleteSegment

**BOOL GpiDeleteSegment** (*hps*, *idSegment*)

**HPS** *hps*;         /\* presentation-space handle \*/

**LONG** *idSegment*; /\* Identifier of segment to delete \*/

The **GpiDeleteSegment** function deletes the segment specified by *idSegment*. If the segment is open, the function automatically closes the segment before deleting it. If the segment is in the picture chain, the function removes it from the chain.

This function deletes only segments created using the **GpiOpenSegment** function.

**Parameters**     *hps*   Identifies the presentation space.

*idSegment*   Specifies the segment to delete; it must be greater than zero.

**Return Value**   The return value is **TRUE** if the function is successful or **FALSE** if an error occurred.

**Errors**         Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

**PMERR\_INV\_MICROPS\_FUNCTION**  
**PMERR\_INV\_SEG\_NAME**

**Example**         This example uses the **GpiDeleteSegment** function to delete segment 4:

```
POINTL pt1Start = { 0, 0 };
POINTL pt1Triangle[] = { 100, 100, 200, 0, 0, 0 };

GpiOpenSegment(hps, 4L); /* open the segment */
GpiMove(hps, &pt1Start); /* move to start point (0, 0) */
GpiPolyLine(hps, 3L, pt1Triangle); /* draw triangle */
GpiCloseSegment(hps); /* close the segment */

.

GpiDeleteSegment(hps, 4L); /* delete segment #4 */
```

**See Also**        **GpiCloseSegment**, **GpiDeleteSegments**, **GpiOpenSegment**, **GpiQuerySegmentNames**

## ■ GpiDeleteSegments

---

**BOOL GpiDeleteSegments** (*hps, idFirstSegment, idLastSegment*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *idFirstSegment*; /\* identifier of first segment \*/

**LONG** *idLastSegment*; /\* identifier of last segment \*/

The **GpiDeleteSegments** function deletes the segments between and including the segments specified by the *idFirstSegment* and *idLastSegment* parameters. If *idFirstSegment* and *idLastSegment* are equal, the function deletes only that segment. If *idFirstSegment* is greater than *idLastSegment*, the function deletes only the segment specified by *idFirstSegment*. If any of the segments is open, the function closes the segment before deleting it. If any of the segments is in the picture chain, the function removes the segment from the chain.

This function deletes only segments created using the **GpiOpenSegment** function.

**Parameters** *hps* Identifies the presentation space.

*idFirstSegment* Specifies the identifier of the first segment to delete. This parameter must be greater than zero.

*idLastSegment* Specifies the identifier of the last segment to delete. This parameter must be greater than zero.

**Return Value** The return value is **GPL\_OK** if the function is successful or **GPL\_ERROR** if an error occurred.

**Errors** Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

PMERR\_INV\_MICROPS\_FUNCTION  
PMERR\_INV\_SEG\_NAME

**Example** This example uses the **GpiDeleteSegments** function to delete segments 4 through 6:

```
GpiDeleteSegments(hps, 4L, 6L); /* delete segments 4 through 6 */
```

**See Also** **GpiCloseSegment**, **GpiDeleteSegment**, **GpiOpenSegment**, **GpiQuerySegmentNames**

## ■ GpiDeleteSetId

---

**BOOL GpiDeleteSetId** (*hps, lcid*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *lcid*; /\* local identifier for font or bitmap \*/

The **GpiDeleteSetId** function deletes a logical font or removes the tag from a tagged bitmap, depending on the object identified by local identifier *lcid*. If the object is a logical font, the function deletes it, making it no longer available for use. If the object is a bitmap, the function removes the tag, but the bitmap handle remains valid. In either case, the function frees the local identifier for use with another object.

- Parameters** *hps* Identifies the presentation space.  
*lcid* Specifies the local identifier for the object. If this parameter is set to `LCID_ALL`, the function deletes all logical fonts and removes the tags from all tagged bitmaps.
- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.
- Example** This example uses the `GpiDeleteSetId` function to delete a logical font. The `GpiSetCharSet` function is required only if the logical font is the current font for the presentation space.
- ```
FATRS fat;
/* create and set the font */
GpiCreateLogFont(hps, NULL, 1L, &fat);
GpiSetCharSet(hps, 1L);
.
GpiSetCharSet(hps, 0L);          /* release the font before deleting */
GpiDeleteSetId(hps, 1L);       /* delete the logical font */
```
- See Also** `GpiSetBitmapId`, `GpiSetCharSet`

■ GpiDestroyPS

BOOL GpiDestroyPS(*hps*)

HPS *hps*; /* presentation-space handle */

The `GpiDestroyPS` function destroys the presentation space and releases all resources owned by the presentation space. This function should only be used to destroy presentation spaces created by the `GpiCreatePS` function.

- Parameters** *hps* Identifies the presentation space.
- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.
- Example** This example uses the `GpiDestroyPS` function to destroy the presentation space associated with a memory device context:
- ```
HDC hdc;
HPS hps;
SIZEL page = { 0, 0 };
/* create the memory device context and presentation space */
hdc = DevOpenDC(hab, OD_MEMORY, "", 0L, NULL, NULL);
hps = GpiCreatePS(hab, hdc, &page, PU_PELS | GPIT_MICRO | GPIA_ASSOC);
.
GpiDestroyPS(hps); /* destroy the presentation space */
DevCloseDC(hdc); /* close the device context */
```
- See Also** `GpiCreatePS`

## ■ GpiDestroyRegion

---

**BOOL GpiDestroyRegion**(*hps*, *hrgn*)

**HPS** *hps*; /\* presentation-space handle \*/

**HRGN** *hrgn*; /\* handle of region to destroy \*/

The **GpiDestroyRegion** function destroys the region specified by *hrgn*. The function destroys the region only if the device context containing the region is associated with the given presentation space.

**Parameters**     *hps*   Identifies the presentation space.

*hrgn*   Identifies the region to destroy.

**Return Value**    The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**Example**         This example uses the **GpiDestroyRegion** function to destroy a region after drawing a complex figure:

```
HRGN hrgn;
RECTL arcl[3] = { 10, 10, 20, 20, 15, 15, 25, 25, 20, 20, 30, 30 };

hrgn = GpiCreateRegion(hps, 3L, arcl); /* use 3 rectangles */
GpiPaintRegion(hps, hrgn); /* paint the region */
GpiDestroyRegion(hps, hrgn); /* destroy the region */
```

**See Also**         **GpiCreateRegion**

## ■ GpiDrawChain

---

**BOOL GpiDrawChain**(*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The **GpiDrawChain** function draws the picture chain. The function draws all segments in the picture chain, including called segments. **GpiDrawChain** draws the segments using the current draw controls (except correlation control), as set by the **GpiSetDrawControl** function. The function does not affect drawing modes or open segments.

The function cannot be used in an area, path, or element bracket.

**Parameters**     *hps*   Identifies the presentation space.

**Return Value**    The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**Errors**         Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

```
PMERR_AREA_INCOMPLETE
PMERR_IMAGE_INCOMPLETE
PMERR_INV_MICROPS_FUNCTION
PMERR_PATH_INCOMPLETE
PMERR_STOP_DRAW_OCCURRED
```

**See Also**         **GpiCloseSegment**, **GpiDrawDynamics**, **GpiDrawFrom**, **GpiDrawSegment**, **GpiQuerySegmentNames**, **GpiSetDrawControl**

## ■ GpiDrawDynamics

---

**BOOL GpiDrawDynamics** (*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The **GpiDrawDynamics** function draws the dynamic segments in the picture chain. The function draws all dynamic segments unless a previous call to the **GpiRemoveDynamics** function restricts the drawing to a selected range. The function draws the segments using the current draw controls (except correlation control), as set by the **GpiSetDrawControl** function.

**Parameters**      *hps*    Identifies the presentation space.

**Return Value**    The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**Errors**            Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

**PMERR\_AREA\_INCOMPLETE**  
**PMERR\_INV\_MICROPS\_FUNCTION**  
**PMERR\_PATH\_INCOMPLETE**  
**PMERR\_STOP\_DRAW\_OCCURRED**

**See Also**         **GpiCloseSegment**, **GpiDrawChain**, **GpiDrawFrom**, **GpiDrawSegment**, **GpiQuerySegmentNames**, **GpiRemoveDynamics**, **GpiSetDrawControl**

## ■ GpiDrawFrom

---

**BOOL GpiDrawFrom** (*hps*, *idFirstSegment*, *idLastSegment*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *idFirstSegment*; /\* first chain segment to draw \*/

**LONG** *idLastSegment*; /\* last chain segment to draw \*/

The **GpiDrawFrom** function draws one or more segments in the picture chain. The function draws all chained and called segments between and including the segments identified by the *idFirstSegment* and *idLastSegment* parameters. Although *idFirstSegment* must identify an existing segment, *idLastSegment* need not. If *idLastSegment* does not specify an existing segment, the function draws to the end of the picture chain.

**GpiDrawFrom** draws the segments using the current draw controls (except correlation control), as set by the **GpiSetDrawControl** function. The function does not affect drawing modes or open segments. Also, **GpiDrawFrom** cannot be used in an area, path, or element bracket.

**Parameters**      *hps*    Identifies the presentation space.

*idFirstSegment*    Specifies the identifier of the first segment to draw. This parameter must be greater than zero.

*idLastSegment*    Specifies the identifier of the last segment to draw. This parameter must be greater than zero.



- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:
- ```

PMERR_AREA_INCOMPLETE
PMERR_IMAGE_INCOMPLETE
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_SEG_NAME
PMERR_PATH_INCOMPLETE
PMERR_STOP_DRAW_OCCURRED

```
- Example** This example uses the `GpiDrawFrom` function to draw all segments in the picture chain between and including the segments 1 and 4:
- ```
GpiDrawFrom(hps, 1L, 4L);
```
- See Also** `GpiCloseSegment`, `GpiDrawChain`, `GpiDrawDynamics`, `GpiDrawSegment`, `GpiQuerySegmentNames`, `GpiSetDrawControl`

## ■ GpiDrawSegment

---

**BOOL** `GpiDrawSegment`(*hps*, *idSegment*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *idSegment*; /\* identifier of segment to draw \*/

The `GpiDrawSegment` function draws the specified segment. The function draws the segments using the current draw controls (except correlation control), as set by the `GpiSetDrawControl` function. The function does not affect drawing modes or open segments.

`GpiDrawSegment` cannot be used in an area, path, or element bracket.

- Parameters** *hps* Identifies the presentation space.  
*idSegment* Identifies the segment to draw. This parameter must be greater than zero.

**Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```

PMERR_AREA_INCOMPLETE
PMERR_IMAGE_INCOMPLETE
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_SEG_NAME
PMERR_PATH_INCOMPLETE
PMERR_STOP_DRAW_OCCURRED

```

**Example** This example uses the `GpiDrawSegment` function to draw segment 4:

```
POINTL ptlStart = { 0, 0 };
POINTL ptlTriangle[] = { 100, 100, 200, 0, 0, 0 };

GpiOpenSegment(hps, 4L); /* open the segment */
GpiMove(hps, &ptlStart); /* move to start point (0, 0) */
GpiPolyLine(hps, 3L, ptlTriangle); /* draw triangle */
GpiCloseSegment(hps); /* close the segment */
.
.
.
GpiDrawSegment(hps, 4L); /* draw segment #4 */
```

**See Also** `GpiCloseSegment`, `GpiDrawChain`, `GpiDrawDynamics`, `GpiDrawFrom`, `GpiQuerySegmentNames`, `GpiSetDrawControl`

## ■ GpiElement

**LONG GpiElement**(*hps*, *lType*, *psz*, *cb*, *pb*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *lType*; /\* element type \*/  
**PSZ** *psz*; /\* pointer to element descriptor \*/  
**LONG** *cb*; /\* length in bytes of buffer for graphics orders \*/  
**PBYTE** *pb*; /\* pointer to buffer for graphics orders \*/

The `GpiElement` function draws an element. The element consists of one or more graphics orders in the buffer pointed to by *pb*. The function executes each order as if it were the corresponding `Gpi` function.

The function adds the element to the current open segment if the drawing mode is `DM_RETAIN` or `DM_DRAWANDRETAIN`. Otherwise, it just draws the element. The element must not contain graphics orders for an element bracket. Similarly, the function cannot be used in an element bracket.

The function sets the type and descriptor for the element to the values given by *lType* and *psz*. The type and descriptor are a useful way of uniquely identifying the element when it is added to a segment. The type and descriptor can be retrieved at any time by using the `GpiQueryElementType` function.

### Parameters

*hps* Identifies the presentation space.

*lType* Specifies the integer value to use for the element type.

*psz* Points to the null-terminated string to use for the element descriptor.

*cb* Specifies the length of graphics order data for the element.

*pb* Points to the buffer that contains the graphics orders for the element. The buffer must not exceed 63K.

### Return Value

The return value is `GPL_OK` or `GPL_HITS` if the function is successful. (It is `GPL_HITS` if the detectable attribute is set for the presentation space and a correlation hit occurs.) The return value is `GPL_ERROR` if an error occurs.

### Errors

Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_DATA_TOO_LONG
PMERR_INV_LENGTH
PMERR_INV_MICROPS_FUNCTION
```

**Comments** **GpiElement** does not convert coordinates. This may affect drawing the element if the format for the coordinates in the graphics orders is not the correct format for the presentation space.

**See Also** **GpiBeginElement**, **GpiDeleteElement**, **GpiEndElement**, **GpiQueryElement**, **GpiQueryElementPointer**, **GpiQueryElementType**, **GpiSetElementPointer**

## ■ GpiEndArea

**LONG GpiEndArea(hps)**

**HPS hps;** /\* presentation-space handle \*/

The **GpiEndArea** function ends an area bracket—that is, it ends the sequence of functions (starting with the **GpiBeginArea** function) that define the outline of an area. The function automatically closes any open figure in the area, if necessary, by drawing a line from the current position to the starting point of the figure, then draws the area using the filling mode specified by the **GpiBeginArea** function that started the area bracket.

The **GpiEndArea** function does not change the current position unless it must draw a line to close a figure in the area. In this case the new position is the last point in the line.

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is **GPL\_OK** or **GPL\_HITS** if the function is successful. (It is **GPL\_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs.) The return value is **GPL\_ERROR** if an error occurs.

**Example** This example uses the **GpiEndArea** function to end an area bracket. The function draws the area (a triangle) by filling the outline with the current fill pattern.

```
POINTL pt1Start = { 0, 0 };
POINTL pt1Triangle[] = { 100, 100, 200, 0, 0, 0 };

GpiBeginArea(hps, BA_NOBOUNDARY | BA_ALTERNATE);
GpiMove(hps, &pt1Start);
GpiPolyLine(hps, 3L, pt1Triangle);
GpiEndArea(hps);
```

**See Also** **GpiBeginArea**

## ■ GpiEndElement

**BOOL GpiEndElement(hps)**

**HPS hps;** /\* presentation-space handle \*/

The **GpiEndElement** function ends an element bracket—that is, it ends the sequence of functions (starting with the **GpiBeginElement** function) that define the contents of an element. The **GpiEndElement** function may only be used while creating a segment.

- Parameters** *hps* Identifies the presentation space.
- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:

PMERR\_INV\_MICROPS\_FUNCTION

- Example** This example uses the `GpiEndElement` function to end an element bracket:

```
POINTL ptlStart = { 0, 0 };
POINTL ptlTriangle[] = { 100, 100, 200, 0, 0, 0 };
.
.
GpiBeginElement(hps, 1L, "Triangle"); /* begin the element bracket */
GpiMove(hps, &ptlStart); /* move to start point (0, 0) */
GpiPolyLine(hps, 3L, ptlTriangle); /* draw triangle */
GpiEndElement(hps); /* end element bracket */
```

- See Also** `GpiBeginElement`, `GpiDeleteElement`, `GpiQueryElement`, `GpiQueryElementPointer`, `GpiSetElementPointer`

## ■ GpiEndPath

**BOOL GpiEndPath**(*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The `GpiEndPath` function ends a path bracket—that is, it ends the sequence of functions (starting with the `GpiBeginPath` function) that define the outline of a path.

- Parameters** *hps* Identifies the presentation space.
- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

- Example** This example uses the `GpiEndPath` function to end a path bracket. When the path bracket is ended, a subsequent call to the `GpiFillPath` function draws and fills the path.

```
POINTL ptlStart = { 0, 0 };
POINTL ptlTriangle[] = { 100, 100, 200, 0 };
GpiBeginPath(hps, 1L); /* start the path bracket */
GpiMove(hps, &ptlStart); /* move to starting point */
GpiPolyLine(hps, 2L, ptlTriangle); /* draw the three sides */
GpiCloseFigure(hps); /* close the triangle */
GpiEndPath(hps); /* end the path bracket */
GpiFillPath(hps, 1L, FPATH_ALTERNATE); /* draw and fill the path */
```

- See Also** `GpiBeginPath`

## ■ GpiEqualRegion

---

**LONG GpiEqualRegion**(*hps*, *hrgn1*, *hrgn2*)

**HPS** *hps*; /\* presentation-space handle \*/

**HRGN** *hrgn1*; /\* handle of the first region \*/

**HRGN** *hrgn2*; /\* handle of the second region \*/

The **GpiEqualRegion** function checks two regions for equality. Regions are equal if the difference between the two regions is an empty region. The function compares the regions only if the device context containing the regions is associated with the given presentation space.

**Parameters**

*hps* Identifies the presentation space.

*hrgn1* Identifies the first region.

*hrgn2* Identifies the second region.

**Return Value** The return value is **EQRGN\_NOTEQUAL** or **EQRGN\_EQUAL** if the function is successful, or **EQRGN\_ERROR** if an error occurred.

**See Also** [WinEqualRect](#)

## ■ GpiErase

---

**BOOL GpiErase**(*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The **GpiErase** function clears the display associated with the specified presentation space. The function clears the display by filling it with the color specified by the **CLR\_BACKGROUND** color index for the presentation space. The function clips the output to the current clipping region, graphics field, and visual region (if any), but does not clip to the current viewing limits and clipping path. Also, the function ignores the the current draw controls (as set by the **GpiSetDrawControl** function).

**Parameters**

*hps* Identifies the presentation space.

**Return Value** The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**Example** This example uses the **GpiErase** function to clear the display before drawing:

```
GpiErase(hps); /* clear the display */
GpiMove(hps, &pt1Start); /* draw a triangle */
GpiPolyLine(hps, 3L, pt1Triangle);
```

**See Also** [GpiCloseSegment](#), [GpiSetColor](#), [GpiSetDrawControl](#)

## ■ GpiErrorSegmentData

**LONG GpiErrorSegmentData**(*hps, pidSegment, plContext*)

**HPS** *hps*; /\* presentation-space handle \*/

**PLONG** *pidSegment*; /\* pointer to segment identifier \*/

**PLONG** *plContext*; /\* pointer to variable for error type \*/

The **GpiErrorSegmentData** function returns information about the last error that occurred while drawing a segment. The function copies the segment identifier and error type to the variables pointed to by *pidSegment* and *plContext*, then returns either a byte offset or an element pointer position, depending on the type of error.

### Parameters

*hps* Identifies the presentation space.

*pidSegment* Points to a variable to receive the identifier of the segment causing the error.

*plContext* Points to a variable to receive the error type. It can be one of the following values:

| Value        | Meaning                                                                                                                                                                         |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GPIE_DATA    | A graphics order in the buffer for the <b>GpiPutData</b> function caused an error. The return value is the byte offset from the beginning of the buffer to this graphics order. |
| GPIE_ELEMENT | A graphics order in the buffer for the <b>GpiElement</b> function caused an error. The return value is the byte offset from the beginning of the buffer to this graphics order. |
| GPIE_SEGMENT | An element in the given segment caused an error. The return value is the position of the element pointer for this element.                                                      |

### Return Value

The return value is either a byte offset or an element pointer position if the function is successful. Otherwise, it is **GPI\_ALTEERROR**.

### Errors

Use the **WinGetLastError** function to retrieve the error value, which may be the following:

**PMERR\_INV\_MICROPS\_FUNCTION**

### See Also

**GpiCloseSegment**, **GpiElement**, **GpiOpenSegment**, **GpiPutData**

## ■ GpiExcludeClipRectangle

---

**LONG GpiExcludeClipRectangle** (*hps, prcl*)

**HPS** *hps*; /\* presentation-space handle \*/  
**PRECTL** *prcl*; /\* pointer to structure for rectangle coordinates \*/

The **GpiExcludeClipRectangle** function excludes a rectangle from the clip region. The function excludes all points in the rectangle except points on the top and right boundary.

**Parameters** *hps* Identifies the presentation space.

*prcl* Points to a **RECTL** structure containing the rectangle. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is **RGN\_COMPLEX**, **RGN\_NULL**, or **RGN\_RECT** if the function is successful or **RGN\_ERROR** if an error occurred.

**See Also** **GpiIntersectClipRectangle**, **WinExcludeUpdateRegion**

## ■ GpiFillPath

---

**LONG GpiFillPath** (*hps, idPath, flFill*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *idPath*; /\* identifier of path \*/  
**LONG** *flFill*; /\* fill mode \*/

The **GpiFillPath** function draws the interior of the path specified by *idPath* by filling it with the current fill pattern. The function first closes any open figures in the path, then fills the closed figures using the filling mode specified by *flFill*. Finally, the function deletes the path.

**Parameters** *hps* Identifies the presentation space.

*idPath* Specifies the path whose interior is to be drawn; it must equal 1.

*flFill* Specifies the fill option. It can be one of the following values:

| Value                  | Meaning                                             |
|------------------------|-----------------------------------------------------|
| <b>FPATH_ALTERNATE</b> | Fills the path using the alternate (even/odd) rule. |
| <b>FPATH_WINDING</b>   | Fills the path using the winding rule.              |

The default is **FPATH\_ALTERNATE**.

- Return Value** The return value is `GPL_OK` or `GPL_HITS` if the function is successful. (It is `GPL_HITS` if the detectable attribute is set for the presentation space and a correlation hit occurs.) The return value is `GPL_ERROR` if an error occurs.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:
- ```
PMERR_INV_PATH_ID
PMERR_PATH_UNKNOWN
```
- Example** This example uses the `GpiFillPath` function to draw the interior of the given path. The path, an isosceles triangle, is not closed when it is created, so the `GpiFillPath` function closes it before filling.
- ```
POINTL pt1Start = { 0, 0 };
POINTL pt1Triangle[] = { 100, 100, 200, 0, 0, 0 };

GpiBeginPath(hps, 1L); /* create a path */
GpiMove(hps, &pt1Start);
GpiPolyLine(hps, 3L, pt1Triangle);
GpiEndPath(hps);

GpiFillPath(hps, 1L, EPATH_ALTERNATE); /* fill the path */
```
- See Also** `GpiBeginPath`, `GpiEndPath`

## ■ GpiFullArc

```
LONG GpiFullArc(hps, flFlags, fxMultiplier)
HPS hps; /* presentation-space handle */
LONG flFlags; /* fill and outline indicator */
FIXED fxMultiplier; /* arc-size multiplier */
```

The `GpiFullArc` function creates a full arc. A full arc is a complete circle or ellipse, drawn by using the current arc parameters. The function first scales the width and height of the arc by using the multiplier specified by the `fxMultiplier` parameter, then draws either the outline of the arc, the interior of the arc, or both, depending on the flags specified by the `flFlags` parameter.

The function uses the current position as the center of the arc but does not change the current position. The function uses the arc parameters to determine whether to draw the full arc clockwise or counterclockwise. When an arc is used as part of an area or path, the direction in which the arc is drawn can affect how it is filled.

**Parameters** `hps` Identifies the presentation space.

`flFlags` Specifies whether to fill and/or outline the arc. It can be one of the following values:

| Value                        | Meaning                                                                 |
|------------------------------|-------------------------------------------------------------------------|
| <code>DRO_FILL</code>        | Fills the interior of the arc with the current fill pattern.            |
| <code>DRO_OUTLINE</code>     | Draws the outline of the arc by using the current line style and color. |
| <code>DRO_OUTLINEFILL</code> | Draws the outline and fills the arc interior.                           |

Do not use `DRO_FILL` or `DRO_OUTLINEFILL` when using `GpiFullArc` in an area bracket.



*fxMultiplier* Specifies how much to scale the width and height of the arc. It must be a fixed-point value in the range 1 through 255 (or in the range 0x10000 through 0xFF0000 if expressed as 32-bit values). This means the function can scale the arc from 1 to 255 times the current arc-parameter dimensions.

**Return Value** The return value is `GPL_OK` or `GPL_HITS` if the function is successful (it is `GPL_HITS` if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is `GPL_ERROR` if an error occurs.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_INV_ARC_CONTROL
PMERR_INV_MULTIPLIER
```

**Comments** When correlating an arc, the system generates a hit if the arc boundary intersects the pick aperture. If the pick aperture is inside the arc, the system generates a hit only if the interior of the arc has been filled.

**Example** This example uses `GpiFullArc` to draw five concentric circles. The arc parameters are set before drawing the arc. Only the outline is drawn for the arc.

```
SHORT i;
ARCPARAMS arcp = { 1, 1, 0, 0 };
GpiSetArcParams(hps, &arcp);

for (i = 5; i > 0; i --)
 GpiFullArc(hps, /* presentation-space handle */
 DRO_OUTLINE, /* outline */
 MAKEFIXED(i, 0)); /* converts integer to fixed point */
```

**See Also** `GpiMove`, `GpiPointArc`, `GpiQueryArcParams`, `GpiSetArcParams`, `GpiSetAttrs`, `GpiSetColor`, `GpiSetCurrentPosition`, `GpiSetLineType`

## ■ GpiGetData

---

```
LONG GpiGetData(hps, idSegment, off, pcmdFormat, cb, pb)
HPS hps; /* presentation-space handle */
LONG idSegment; /* segment identifier */
PLONG off; /* pointer to variable for segment offset */
LONG pcmdFormat; /* conversion type */
LONG cb; /* length in bytes of the data buffer */
PBYTE pb; /* pointer to buffer for data */
```

The `GpiGetData` function copies graphics orders from the specified segment to the specified buffer. The function continues to copy the graphics orders from the segment to the buffer until all orders in the segment have been copied or the number of bytes specified by the `cb` parameter have been copied. If the function fills the buffer, the last order in the buffer may not be complete since the function does not stop on an order boundary when copying to the buffer. In any case, the function returns the number of bytes copied to the buffer.

The function starts copying graphics-order data from the location specified by the `off` parameter. If this parameter is zero, the function copies from the beginning of the segment. After copying the data, the function replaces the value in `off` with the offset to the next byte of data to copy from the segment (if any). This value can be used to specify the next location to copy.

The **GpiGetData** function cannot be used to copy data from an open segment, but it can be used to copy data while some other segment is open.

## Parameters

*hps* Identifies the presentation space.

*idSegment* Specifies the segment identifier.

*off* Specifies the offset from the beginning of the segment to the next byte of graphics order data to copy. If this parameter is zero, the function copies from the beginning of the segment.

*pcmdFormat* Points to the variable that contains the coordinate conversion type. The variable can be one of the following values:

| Value           | Meaning                                                                                                  |
|-----------------|----------------------------------------------------------------------------------------------------------|
| DFORM_NOCONV    | Copies coordinates without converting. The coordinates are in the format used by the presentation space. |
| DFORM_PCLONG    | Converts coordinates to PC-format long (4-byte) integers.                                                |
| DFORM_PCSHORT   | Converts coordinates to PC-format short (2-byte) integers.                                               |
| DFORM_S370SHORT | Converts coordinates to S/370-format short (2-byte) integers.                                            |

*cb* Specifies the length in bytes of the buffer to receive the graphics orders.

*pb* Points to the buffer that receives the graphics-order data.

## Return Value

The return value is the number of graphics-order bytes copied if the function is successful or **GPI\_ALTError** if an error occurred.

## Errors

Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

```
PMERR_DATA_TOO_LONG
PMERR_INV_GETDATA_CONTROL
PMERR_INV_LENGTH
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_SEG_OFFSET
PMERR_SEG_NOT_FOUND
```

## Example

This example uses the **GpiGetData** function to copy data from one segment to another:

```
LONG fFormat = DFORM_NOCONV; /* do not convert coordinates */
LONG offSegment = 0L; /* offset in segment */
LONG offNextElement = 0L; /* offset in segment to next element */
LONG cb = 0L; /* bytes retrieved */
BYTE abBuffer[512];

GpiOpenSegment(hps, 3L); /* open segment to receive the data */
do {
 offSegment += cb;
 offNextElement = offSegment;
 cb = GpiGetData(hps, 2L, &offNextElement, fFormat, 512L, abBuffer);
```

```

 /* put data in other segment */
 if (cb > OL) GpiPutData(hps, /* presentation-space handle */
 fFormat, /* format of coordinates */
 &cb, /* number of bytes in buffer */
 abBuffer); /* buffer with graphics-order data */
 } while (cb > 0);
 GpiCloseSegment(hps); /* close segment that received the data */

```

See Also **GpiPutData**

## ■ Gpimage

**LONG Gpimage**(*hps, IFormat, psizl, cbData, pbData*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *IFormat*; /\* image data format \*/  
**PSIZEL** *psizl*; /\* pointer to structure for image width and height \*/  
**LONG** *cbData*; /\* length in bytes of the image data \*/  
**PBYTE** *pbData*; /\* pointer to image data \*/

The **Gpimage** function draws an image. An image is a rectangular array of pels, each pel having either the current foreground or background color. Each image has a width and height specified by the *psizl* parameter. The width and height determines how many pels there are in the horizontal and vertical directions.

**Gpimage** draws the image by using the image data pointed to by the *pbData* parameter to set the color of each pel in the image. Each pel is represented by one bit in the image data. If the bit is 1, the pel has the foreground color; if the bit is 0, the pel has the background color. The function combines each pel with the color already on the display by using the foreground mix mode for foreground pels and the background mix mode for background pels. The function places the upper-left corner of the image at the current position but does not change the current position.

**Parameters** *hps* Identifies the presentation space.

*IFormat* Specifies the format of the image data. This is a reserved field; it must be set to zero.

*psizl* Points to a **SIZEL** structure containing the width and height of the image in pels. The maximum width allowed is 2040 pixels. The **SIZEL** structure has the following form:

```

typedef struct _SIZEL {
 LONG cx;
 LONG cy;
} SIZEL;

```

For a full description, see Chapter 4, "Types, Macros, Structures."

*cbData* Specifies the length in bytes of the image data.

*pbData* Points to the image data. The pels must be given, row by row, starting at the top and running from left to right within each row.

## Return Value

The return value is **GPL\_OK** or **GPL\_HITS** if the function is successful (it is **GPL\_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is **GPL\_ERROR** if an error occurs.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_INV_IMAGE_DATA_LENGTH
PMERR_INV_IMAGE_DIMENSION
PMERR_INV_IMAGE_FORMAT
```

**Comments** The image data is an array of bytes. Each byte in the array represents eight pels, with the high bit representing the leftmost pel. The function draws the image from left to right and top to bottom. For each row of the image, the function continues to read bytes from the array until all pels in the row are set. If the image width is not a multiple of 8, any remaining bits in the last byte for the row are ignored. The function continues until all rows are set. This means the number of bytes in the image data (and the length specified for the data) must be equal to the height in pels multiplied by the width in bytes.

**Example** This example uses `GpiImage` to draw an 8-by-8 image. The image data is specified as an array of bytes.

```
SIZEL siz1 = { 8, 8 }; /* image is 8 pels wide by 8 pels high */
BYTE abImage[] = { 0x00, 0x18, 0x3c, 0x7e, 0xff,
 0xff, 0x7e, 0x3c, 0x18, 0x00 };

GpiImage(hps, 0L, & siz1, 8L, abImage); /* draws the image */
```

**See Also** `GpiSetAttrs`

## ■ GpiIntersectClipRectangle

```
LONG GpiIntersectClipRectangle(hps, prcl)
HPS hps; /* presentation-space handle */
PRECTL prcl; /* pointer to structure for rectangle coordinates */
```

The `GpiIntersectClipRectangle` function sets the new clip region (in device coordinates) to the intersection of the current clip region and the specified rectangle.

**Parameters** *hps* Identifies the presentation space.  
*prcl* Points to a `RECTL` structure. The `RECTL` structure has the following form:

```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is `RGN_NULL`, `RGN_RECT`, or `RGN_COMPLEX` if the function is successful, or `RGN_ERROR` if an error occurred.

**See Also** `GpiExcludeClipRectangle`

## ■ GpiLabel

---

**BOOL GpiLabel**(*hps, idLabel*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *idLabel*; /\* label \*/

The **GpiLabel** function creates a label element. A label element is an element in a segment that contains nothing more than a 32-bit value. The function creates a label for an element in the current open segment. If no segment is open, no label is created.

The **GpiLabel** function cannot be used in an element bracket.

**Parameters** *hps* Identifies the presentation space.

*idLabel* Specifies the label. It can be any value in the range 0x00000000 through 0xFFFFFFFF.

**Return Value** The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**Errors** Use the **WinGetLastError** function to retrieve the error value, which may be the following:

PMERR\_INV\_MICROPS\_FUNCTION

**Comments** The **GpiLabel** function is intended to be used to uniquely identify elements in a segment that may be edited. Label elements are typically placed near elements to be edited. The label can be used with the **GpiSetElementPointerAtLabel** function to move the element pointer to the given element.

**Example** This example uses the **GpiLabel** function to create label elements in a segment. If the segment is subsequently edited, the label elements can still be used to locate the elements near it.

```
POINTL ptlStart = { 0, 0 };
POINTL ptlTriangle[] = { 100, 100, 200, 0, 0, 0 };

GpiOpenSegment(hps, 4L); /* creates a segment */
GpiLabel(hps, 5L); /* creates label 5 */
GpiLabel(hps, 10L); /* creates label 10 */
GpiMove(hps, &ptlStart);
GpiCloseSegment(hps);
GpiPolyLine(hps, 3L, ptlTriangle);
```

**See Also** **GpiSetElementPointerAtLabel**

## ■ GpiLine

---

**LONG GpiLine**(*hps, pptl*)

**HPS** *hps*; /\* presentation-space handle \*/  
**PPOINTL** *pptl*; /\* pointer to structure for the end point \*/

The **GpiLine** function draws a straight line from the current position to the specified end point. The function then moves the current position to the end point.

The function draws the line by using the current values of the line-color, line-mix, line-width, and line-type attributes. These values are set by using the **GpiSetAttrs** function.

- Parameters** *hps* Identifies the presentation space.  
*pptl* Points to a POINTL structure that contains the end point of the line. The POINTL structure has the following form:
- ```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```
- For a full description, see Chapter 4, “Types, Macros, Structures.”
- Return Value** The return value is GPL_OK or GPL_HITS if the function is successful (it is GPL_HITS if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is GPL_ERROR if an error occurs.
- Example** This example uses GpiLine to draw an X.
- ```
POINTL pt1[4] = { 0, 0, 100, 100, 0, 100, 100, 0 };

GpiMove(hps, &pt1[0]);
GpiLine(hps, &pt1[1]);
GpiMove(hps, &pt1[2]);
GpiLine(hps, &pt1[3]);
```
- See Also** GpiMove, GpiPolyLine, GpiSetAttrs, GpiSetColor, GpiSetCurrentPosition, GpiSetLineStyle

## ■ GpiLoadBitmap

**HBITMAP GpiLoadBitmap**(*hps, hmod, idBitmap, IWidth, IHeight*)

**HPS** *hps*; /\* presentation-space handle \*/  
**HMODULE** *hmod*; /\* module handle \*/  
**USHORT** *idBitmap*; /\* bitmap identifier \*/  
**LONG** *IWidth*; /\* width in pels of the bitmap \*/  
**LONG** *IHeight*; /\* height in pels of the bitmap \*/

The **GpiLoadBitmap** function loads a bitmap resource from the specified module and uses it to create a bitmap having the specified width and height. The function uses the image data in the bitmap resource to initialize the bitmap image. If the *IWidth* or *IHeight* parameter is zero, the function creates a bitmap having the width or height given in the bitmap resource. If *IWidth* or *IHeight* is not zero, the function stretches or compresses the bitmap image to the specified width or height.

The bitmap handle can be used in subsequent functions that accept bitmap handles. In most cases, the bitmap is set to a memory presentation space by using the **GpiSetBitmap** function then copied to the screen or a printer by using the **GpiBitBlt** function.

- Parameters** *hps* Identifies the presentation space.  
*hmod* Specifies the module handle of the dynamic-link library containing the bitmap resource. If this parameter is NULL, the function loads the bitmap from the application’s executable file.  
*idBitmap* Specifies the identifier of the bitmap within the resource file.  
*IWidth* Specifies the width in pels of the bitmap.  
*IHeight* Specifies the height in pels of the bitmap.

- Return Value** The return value is a handle to the bitmap if the function is successful or `GPI_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:  
**PMERR\_INV\_BITMAP\_DIMENSION**
- Example** This example uses the `GpiLoadBitmap` function to create a bitmap by using the bitmap resource in the application's executable file. The bitmap must have been added to the executable file by using Resource Compiler.
- ```

HBITMAP hbm;                /* handle of the bitmap */
hbm = GpiLoadBitmap(hps,    /* presentation-space handle */
  NULL,                    /* loads from application's file */
  1,                        /* bitmap resource #1 */
  64L,                      /* sets width to 64 pels */
  64L);                     /* sets height to 64 pels */

```
- See Also** `GpiCreateBitmap`, `GpiDeleteBitmap`, `GpiSetBitmap`, `GpiSetBitmapBits`, `GpiSetBitmapDimension`, `GpiSetBitmapId`, `WinGetSysBitmap`

■ GpiLoadFonts

BOOL GpiLoadFonts(*hab*, *pszModName*)

HAB *hab*; /* anchor-block handle */

PSZ *pszModName*; /* pointer to module name */

The `GpiLoadFonts` function loads fonts from the specified resource file. Once loaded, the fonts are private fonts and can be used by any thread in the process. Any other process can use the fonts but only if it also loads the font by using the `GpiLoadFonts`. The function loads a copy of the fonts once only. Any subsequent call to the function by another process for the same fonts simply increments the use count for the resource and gives that process access.

Parameters *hab* Identifies the anchor block.

pszModName Points to a null-terminated string. This string must be a valid MS OS/2 filename. If it does not specify a path and the filename extension, the function appends the default extension (*.dll*) and searches for the font resource file in the directories specified by the `libpath` command in the `config.sys` file.

Return Value The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

Example This example uses the `GpiLoadFonts` function to load all fonts from the font resource file *helv.fon*. The `GpiQueryFonts` function retrieves the number of fonts loaded.

```
LONG cFonts = 0L;
```

```
GpiLoadFonts(hab, "helv");
cFonts = GpiQueryFonts(hps, QF_PRIVATE, NULL, &cFonts, 0L, NULL);
```

See Also `GpiCreateLogFont`, `GpiDeleteSetId`, `GpiQueryFonts`, `GpiUnloadFonts`

■ GpiLoadMetaFile

HMF GpiLoadMetaFile(*hab*, *pszFilename*)

HAB *hab*; /* anchor-block handle */

PSZ *pszFilename*; /* pointer to filename of metafile */

The **GpiLoadMetaFile** function loads data from a file into a metafile. The function first creates the metafile, then copies the data and returns the metafile handle. The metafile handle can be used in subsequent calls to the **GpiPlayMetaFile** or **GpiDeleteMetaFile** function.

Parameters *hab* Identifies the anchor block.

pszFilename Points to a null-terminated string. This string must be a valid MS OS/2 filename that specifies the path and filename of the file to load into a metafile.

Return Value The return value is a handle to the metafile if the function is successful or **GPI_ERROR** if an error occurred.

Example This example uses the **GpiLoadMetaFile** function to load a metafile with data from the file *sample.met*. Later, the metafile is deleted by using the **GpiDeleteMetaFile** function.

```
HMF hmf;
GpiLoadMetaFile(hmf, "sample.met"); /* loads metafile from disk */
.
.
GpiDeleteMetaFile(hmf); /* deletes metafile */
```

See Also **GpiCopyMetaFile**, **GpiDeleteMetaFile**, **GpiPlayMetaFile**, **GpiSaveMetaFile**, **GpiSetMetaFileBits**

■ GpiMarker

LONG GpiMarker(*hps*, *pptl*)

HPS *hps*; /* presentation-space handle */

PPOINTL *pptl*; /* pointer to structure for marker position */

The **GpiMarker** function draws a marker, placing the center of the marker at the point specified by the *pptl* parameter. The current marker set and marker symbol attributes specify the marker to draw.

The function moves the current position to the specified point.

Parameters *hps* Identifies the presentation space.

pptl Points to a **POINTL** structure that contains the position of the marker. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, "Types, Macros, and Structures."

Return Value The return value is `GPL_OK` or `GPL_HITS` if the function is successful (it is `GPL_HITS` if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is `GPL_ERROR` if an error occurs.

Example This example uses the `GpiMarker` function to draw a marker at the point (10,10).

```
POINTL pt1 = { 10, 10 };
GpiMarker(hps, &pt1);
```

See Also `GpiMove`, `GpiPolyMarker`, `GpiSetAttrs`, `GpiSetCurrentPosition`, `GpiSetMarkerBox`, `GpiSetMarkerSet`

■ GpiModifyPath

BOOL GpiModifyPath(hps, idPath, cmdMode)

```
HPS hps;           /* presentation-space handle */
LONG idPath;       /* path identifier */
LONG cmdMode;      /* modification options */
```

The `GpiModifyPath` function modifies a path. Modifying a path affects the way the `GpiFillPath` function draws the path. For example, a modified path can be used to draw a wide line; that is, a line having a width specified by the current geometric-line width. The function modifies the path as specified by the `cmdMode` parameter.

The `GpiModifyPath` can modify the path for drawing as a wide line. In this case, the `GpiFillPath` function draws a line that follows the path. The line has the current geometric-line width and is filled with the current fill pattern. Furthermore, the current line-join attribute defines how to draw the intersection of two lines at their end points and the current line-end attribute defines how to draw the end of a line, respectively. `GpiModifyPath` prevents `GpiFillPath` from closing open figures in the path. The line-end attribute applies to the start and end points of open figures. If a figure is closed by using the `GpiCloseFigure` function, the line-join attribute applies to the start and end points. If a line is joined to an arc, the line-join attribute applies to the intersection at the end points. If two lines intersect at any place other than their end points, the `GpiFillPath` function draws the wide line so that the intersection is filled despite the fill mode.

Parameters *hps* Identifies the presentation space.

idPath Specifies the identifier of the path to modify; it must be 1.

cmdMode Specifies how to modify the path. It can be the following value:

Value	Meaning
<code>MPATH_STROKE</code>	Converts the path to a wide line. The line width is the current geometric-line width set by using the <code>GpiSetLineWidthGeom</code> function.

Return Value The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

Errors Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_INV_PATH_ID
PMERR_PATH_UNKNOWN
```

Example This example uses the `GpiModifyPath` function to modify the given path. The `GpiFillPath` function then draws the path.

```
POINTL ptlStart = { 0, 0 };
POINTL ptlTriangle[] = { 100, 100, 200, 0, 0, 0 };

GpiBeginPath(hps, 1L); /* creates path */
GpiMove(hps, &ptlStart);
GpiPolyLine(hps, 3L, ptlTriangle);
GpiEndPath(hps);

GpiModifyPath(hps,
              1L,
              MPATH_STROKE); /* modifies path for wide line */
GpiFillPath(hps, 1L, FPATH_ALTERNATE); /* draws the wide line */
```

See Also `GpiBeginPath`, `GpiCloseFigure`, `GpiEndPath`, `GpiSetLineEnd`, `GpiSetLineJoin`, `GpiSetLineWidthGeom`

■ GpiMove

```
BOOL GpiMove(hps, pptl)
HPS hps; /* presentation-space handle */
PPOINTL pptl; /* pointer to structure for new position */
```

The `GpiMove` function moves the current position to the specified point. When used in an area bracket, the function closes the current open figure (if any) and marks the start of a new figure.

Parameters *hps* Identifies the presentation space.

pptl Points to a `POINTL` structure containing the position to move to. The `POINTL` structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is `GPL_OK` or `GPL_HITS` if the function is successful (it is `GPL_HITS` if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is `GPL_ERROR` if an error occurs.

Example This example uses the `GpiMove` function to draw an X. The function moves the current position to the starting point of each leg of the character.

```
POINTL ptl[4] = { 0, 0, 100, 100, 0, 100, 100, 0 };

GpiMove(hps, &ptl[0]); /* move to (0,0) */
GpiLine(hps, &ptl[1]);
GpiMove(hps, &ptl[2]); /* move to (0,100) */
GpiLine(hps, &ptl[3]);
```

See Also `GpiSetCurrentPosition`

■ GpiOffsetClipRegion

LONG GpiOffsetClipRegion(*hps*, *pptl*)

HPS *hps*; /* presentation-space handle */
PPOINTL *pptl*; /* pointer to structure for offset increments */

The **GpiOffsetClipRegion** function moves the clip region. The function moves the clip region by adding the *x*- and *y*-coordinates in the point specified by the *pptl* parameter to the region's current position. The *x*- and *y*-coordinates may be either positive or negative, so the region can move in any direction.

Parameters *hps* Identifies the presentation space.

pptl Points to a **POINTL** structure that contains the offset increments in world coordinates. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

Return Value The return value is **RGN_NULL**, **RGN_RECT**, or **RGN_COMPLEX** if the function is successful, or **RGN_ERROR** if an error occurred.

See Also **GpiSetClipRegion**

■ GpiOffsetElementPointer

BOOL GpiOffsetElementPointer(*hps*, *off*)

HPS *hps*; /* presentation-space handle */
LONG *off*; /* offset to add to element pointer */

The **GpiOffsetElementPointer** function moves the element pointer by the number of elements specified by the *off* parameter. The function starts the move at the current element-pointer position, and moves the element pointer either toward the beginning or end of the segment, depending on whether *off* is negative or positive. If *off* specifies more elements than actually exist between the current position and the beginning or end, the function sets the element pointer to zero or to the last element in the segment, depending on the direction of the move.

The **GpiOffsetElementPointer** function affects the current open segment. If no segment is open, the function is ignored. Also, the function cannot be used in an element bracket.

Parameters *hps* Identifies the presentation space.

off Specifies the offset to be added to the element pointer.

Return Value The return value is **GPI_OK** if the function is successful or **GPI_ERROR** if an error occurred.

Errors Use the `WinGetLastError` function to retrieve the error value, which may be the following:

PMERR_INV_MICROPS_FUNCTION

Example This example uses the `GpiOffsetElementPointer` function to move to the element associated with a label element. Combining the `GpiSetElementPointerAtLabel` and `GpiOffsetElementPointer` functions is a convenient way to locate elements in segments that have been edited.

```
POINTL pt1Start = { 0, 0 };
POINTL pt1Triangle[] = { 100, 100, 200, 0, 0, 0 };

GpiOpenSegment(hps, 4L);          /* creates a segment with labels */
GpiLabel(hps, 5L);    GpiMove(hps, &pt1Start);
GpiLabel(hps, 10L);   GpiPolyLine(hps, 3L, pt1Triangle);
GpiCloseSegment(hps);
.
.
.
GpiOpenSegment(hps, 4L);
GpiSetElementPointerAtLabel(hps, 10L) /* move to label 10 */
GpiOffsetElementPointer(hps, 1L);    /* move to polyline element */
```

See Also `GpiSetEditMode`, `GpiSetElementPointer`, `GpiSetElementPointerAtLabel`

■ GpiOffsetRegion

BOOL GpiOffsetRegion(*hps*, *hrgn*, *pptl*)

HPS *hps*; /* presentation-space handle */
HRGN *hrgn*; /* region handle */
PPOINTL *pptl*; /* pointer to structure for offset increments */

The `GpiOffsetRegion` function moves a region. The function moves the region by adding the *x*- and *y*-coordinates in the point specified by the *pptl* parameter to the region's current position. The *x*- and *y*-coordinates may be either positive or negative, so the region can move in any direction.

Parameters *hps* Identifies the presentation space.

hrgn Identifies the region to move. The region must belong to the device context associated with the presentation space.

pptl Points to a `POINTL` structure that contains the offset increments for the move. The `POINTL` structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

Return Value The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

See Also `GpiCreateRegion`, `GpiDestroyRegion`

■ GpiOpenSegment

BOOL GpiOpenSegment(*hps*, *idSegment*)

HPS *hps*; /* presentation-space handle */

LONG *idSegment*; /* segment identifier */

The **GpiOpenSegment** function opens the segment specified by the *idSegment* parameter. The function creates a new segment if a segment having the specified identifier does not already exist. Otherwise, it opens the segment. Once a segment is opened or created, the system stores an element in the segment for each subsequent primitive and attribute function, up to the next call to the **GpiCloseSegment** function. If the segment previously existed, the system either replaces the old elements with the new or inserts the new elements, depending on the segment editing mode.

The function can create a segment when the drawing mode is set to either **DM_RETAIN** or **DM_DRAWANDRETAIN** but can open an existing segment only when the drawing mode is **DM_RETAIN**. (The **GpiOpenSegment** function can also create a segment when the drawing mode is **DM_DRAW**, but subsequent elements are not stored.)

Parameters *hps* Identifies the presentation space.

idSegment Specifies the segment identifier. The segment identifier must be a positive integer. If the identifier is unique—that is, has not been used before with the presentation space—the function creates a new segment. Zero is reserved for unnamed segments.

Return Value The return value is **GPI_OK** if the function is successful or **GPI_ERROR** if an error occurred.

Errors Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

```
PMERR_AREA_INCOMPLETE
PMERR_DYNAMIC_SEG_ZERO_INV
PMERR_IMAGE_INCOMPLETE
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_MODE_FOR_OPEN_DYN
PMERR_INV_MODE_FOR_REOPEN_SEG
PMERR_PATH_INCOMPLETE
```

Comments If the segment identifier is zero, the function creates an unnamed segment. An unnamed segment is like any other segment except it cannot be referenced by identifiers in subsequent segment functions. For example, an unnamed segment cannot be drawn directly since the **GpiDrawSegment** function requires a segment identifier, but the unnamed segment can be drawn if it is added to the picture chain. **GpiOpenSegment** creates a new unnamed segment for each call specifying the zero identifier. Any number of unnamed segments can be created, and the unnamed segments continue to exist until all segments are deleted.

The **GpiOpenSegment** function assigns segment attributes to each new segment. The initial segment attributes are set by the **GpiSetInitialSegmentAttrs** function. If the initial attributes specify a dynamic segment, the segment can be created only in **DM_RETAIN** drawing mode.

Only one segment per presentation space can be open at a time.

Example This example uses the **GpiOpenSegment** to create a new segment. The segment is subsequently drawn by using the **GpiDrawSegment** function.

```
POINTL pt1Start = { 0, 0 };
POINTL pt1Triangle[] = { 100, 100, 200, 0, 0, 0 };

GpiOpenSegment(hps, 1L);          /* opens the segment */
GpiMove(hps, &pt1Start);         /* moves to starting point (0,0) */
GpiPolyLine(hps, 3L, pt1Triangle); /* draws triangle */
GpiCloseSegment(hps);           /* closes the segment */

GpiDrawSegment(hps, 1L);
```

See Also **GpiCloseSegment**, **GpiErrorSegmentData**, **GpiSetInitialSegmentAttrs**, **GpiSetSegmentAttrs**, **GpiSetViewingTransformMatrix**

■ GpiPaintRegion

LONG GpiPaintRegion(hps, hrgn)

HPS hps; /* presentation-space handle */

HRGN hrgn; /* region handle */

The **GpiPaintRegion** function paints the region specified by the *hrgn* parameter. The function paints a region by filling it with the current fill pattern, applying the current area colors and mix modes as it fills the region.

Parameters *hps* Identifies the presentation space.

hrgn Identifies the region.

Return Value The return value is **GPL_OK** or **GPL_HITS** if the function is successful (it is **GPL_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is **GPL_ERROR** if an error occurs.

Example This example uses the **GpiPaintRegion** function to fill a complex region consisting of three, intersecting rectangles. The region is filled with a red, diagonal pattern.

```
HRGN hrgn;
RECTL arcl[3] = { 100, 100, 200, 200, /* handle for region */
                 150, 150, 250, 250, /* 1st rectangle */
                 200, 200, 300, 300 }; /* 2nd rectangle */
                                     /* 3rd rectangle */

hrgn = GpiCreateRegion(hps, 3L, arcl);
GpiSetColor(hps, CLR_RED);
GpiSetPattern(hps, PATSYM_DIAG1);
GpiPaintRegion(hps, hrgn);
```

See Also **GpiCreateRegion**, **GpiSetAttrs**, **GpiSetColor**, **GpiSetPattern**, **GpiSetPatternRefPoint**, **GpiSetPatternSet**

■ GpiPartialArc

LONG GpiPartialArc(*hps*, *pptl*, *fxMultiplier*, *fxStartAngle*, *fxSweepAngle*)

HPS *hps*; /* presentation-space handle */
PPOINTL *pptl*; /* pointer to structure for center point */
FIXED *fxMultiplier*; /* arc-size multiplier */
FIXED *fxStartAngle*; /* start angle of arc */
FIXED *fxSweepAngle*; /* sweep angle of arc */

The **GpiPartialArc** function draws a partial arc. The function actually draws two figures: a straight line, from the current position to the start point of an arc; and the arc itself, with its center at the specified point. The function determines the start and end points of the arc by using the start and sweep angles specified by the *fxStartAngle* and *fxSweepAngle* functions.

The **GpiPartialArc** function moves the current position to the end point on the partial arc.

Parameters

hps Identifies the presentation space.

pptl Points to a **POINTL** structure that contains the center point. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

fxMultiplier Specifies the amount to scale the width and height of the arc. It must be a fixed-point value in the range 1 through 255 (or in the range 0x10000 to 0xFF0000 if expressed as a 32-bit value). This means the function can scale the arc from 1 to 255 times the current arc-parameter dimensions.

fxStartAngle Specifies the start angle in degrees. It must be a positive, fixed-point value.

fxSweepAngle Specifies the sweep angle in degrees. It must be a positive, fixed-point value.

Return Value

The return value is **GPL_OK** or **GPL_HITS** if the function is successful (it is **GPL_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is **GPL_ERROR** if an error occurs.

Errors

Use the **WinGetLastError** function to retrieve the error value, which may be the following:

PMERR_INV_MULTIPLIER

Comments

To draw the arc, the **GpiPartialArc** function first constructs an imaginary unit circle at the specified center point. The function locates the start point of the arc by measuring counterclockwise from the *x*-axis of the circle by the number of degrees in the start angle. It then locates the end point of the arc by measuring counterclockwise from the start point by the number of degrees in the sweep angle. Finally, the function draws the arc by applying the current arc parameters and the arc-size multiplier. The direction in which the function draws the arc depends on the arc parameters. The direction may affect the way a closed figure containing an arc is filled.

If the sweep angle is greater than 360 degrees, the function draws one or more complete circles or ellipses (depending on the original sweep-angle value) followed by an arc. The sweep angle of the final arc is the remainder after dividing the original sweep angle by 360.

Example

This example uses the `GpiPartialArc` function to draw a chord. A chord is an arc whose end points are connected by a straight line.

```
POINTL pt1 = { 100, 100 };           /* center point for arc */

GpiSetLineType(hps, LINETYPE_INVISIBLE);
GpiPartialArc(hps, &pt1, MAKEFIXED(50, 0), MAKEFIXED(0, 0),
              MAKEFIXED(180, 0));
GpiSetLineType(hps, LINETYPE_SOLID);
GpiPartialArc(hps, &pt1, MAKEFIXED(50, 0), MAKEFIXED(0, 0),
              MAKEFIXED(180, 0));
```

See Also

`GpiFullArc`, `GpiLine`, `GpiMove`, `GpiPointArc`, `GpiQueryArcParams`, `GpiSetArcParams`, `GpiSetAttrs`, `GpiSetColor`, `GpiSetCurrentPosition`, `GpiSetLineType`

■ GpiPlayMetaFile

```
LONG GpiPlayMetaFile(hps, hmf, cOptions, alOptions, pcSegments, cchDesc, pszDesc)
HPS hps;           /* presentation-space handle */
HMF hmf;           /* metafile handle */
LONG cOptions;     /* number of elements in array */
PLONG alOptions;  /* pointer to array of load options */
PLONG pcSegments; /* pointer to count of renumbered segments */
LONG cchDesc;     /* number of bytes in record */
PSZ pszDesc;     /* pointer to buffer for descriptive record */
```

The `GpiPlayMetaFile` function plays the metafile specified by the *hmf* parameter. The function plays the metafile file by converting the graphics data in the file to graphics operations for the given presentation space. The function uses the load options specified by the *alOptions* parameter to determine how to prepare the presentation space for playing the metafile. This may include resetting the presentation space, replacing tagged bitmaps and logical fonts, and replacing the logical color table.

Since the metafile may create segments, the application must close any open segment before calling `GpiPlayMetaFile`. If the metafile creates segments, the function retains the segments only if the current drawing mode is `DM_RETAIN` or `DM_DRAWANDRETAIN`. If chained segments are retained, the function adds them to the end of the existing segment chain.

The `GpiPlayMetaFile` function can play a metafile any number of times.

Parameters

hps Identifies a presentation space.

hmf Identifies the metafile to play. It must have been created or loaded previously by using the `DevOpenDC` or `GpiLoadMetaFile` function.

cOptions Specifies the number of elements in the array pointed to by the *alOptions* parameter.

alOptions Points to the array specifying the load options. For a full description, see the following “Comments” section.

pcSegments Points to a variable for the count of renumbered segments. This parameter is reserved and is set to zero.

cchDesc Specifies the number of bytes in the buffer pointed to by the *pszDesc* parameter.

pszDesc Points to the buffer that receives the null-terminated string describing the metafile. This descriptive record is the record set by the **DevOpenDC** function for the metafile. If the buffer is smaller than the record, the function truncates the record.

Return Value

The return value is **GPI_OK** or **GPI_HITS** if the function is successful (it is **GPI_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is **GPI_ERROR** if an error occurs.

Errors

Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

PMERR_INCOMPATIBLE_METAFILE
PMERR_INV_LENGTH
PMERR_INV_PLAY_METAFILE_OPTION
PMERR_STOP_DRAW_OCCURRED

Comments

The **GpiPlayMetaFile** function uses several options to control how a metafile is played. The options are specified in an array passed to the function by using the *alOptions* parameter. The array has at most ten elements, and there are eight predefined array indexes that can be used to access these elements. The following list describes the purpose and possible values for each element:

Index	Meaning								
PMF_SEGBASE	Specifies a reserved element. It must be zero.								
PMF_LOADTYPE	Specifies the transformation to use when playing the metafile. It can be one of the following:								
	<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>LT_DEFAULT</td> <td>Default; same as LT_NOMODIFY.</td> </tr> <tr> <td>LT_NOMODIFY</td> <td>Use the current viewing transformation as set by the application by using the GpiSetViewingTransformMatrix function. This is the default action.</td> </tr> <tr> <td>LT_ORIGINALVIEW</td> <td>Use the viewing transformations defined in the metafile.</td> </tr> </tbody> </table>	Value	Meaning	LT_DEFAULT	Default; same as LT_NOMODIFY .	LT_NOMODIFY	Use the current viewing transformation as set by the application by using the GpiSetViewingTransformMatrix function. This is the default action.	LT_ORIGINALVIEW	Use the viewing transformations defined in the metafile.
Value	Meaning								
LT_DEFAULT	Default; same as LT_NOMODIFY .								
LT_NOMODIFY	Use the current viewing transformation as set by the application by using the GpiSetViewingTransformMatrix function. This is the default action.								
LT_ORIGINALVIEW	Use the viewing transformations defined in the metafile.								
PMF_RESOLVE	Specifies a reserved element. It must be RS_DEFAULT or RS_NODISCARD .								
PMF_LCIDS	Specifies whether to use tagged bitmaps and logical fonts from the metafile or from the application. It can be one of the following:								
	<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>LC_DEFAULT</td> <td>Default; same as LC_NOLOAD.</td> </tr> <tr> <td>LC_NOLOAD</td> <td>Use the tagged bitmaps and logical fonts defined by the</td> </tr> </tbody> </table>	Value	Meaning	LC_DEFAULT	Default; same as LC_NOLOAD .	LC_NOLOAD	Use the tagged bitmaps and logical fonts defined by the		
Value	Meaning								
LC_DEFAULT	Default; same as LC_NOLOAD .								
LC_NOLOAD	Use the tagged bitmaps and logical fonts defined by the								

Value	Meaning
	application. The application must define the appropriate objects and local identifiers before playing the metafile. This is the default.
LC_LOADDISC	Use the tagged bitmaps and logical fonts defined in the metafile. The function loads the object from the metafile and assigns a local identifier. If the local identifier is already defined by the application, the function deletes the identifier before creating the new object.

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. It can be one of the following:

Value	Meaning
RES_DEFAULT	Default; same as RES_NORESET.
RES_NORESET	Does not reset the presentation space.
RES_RESET	Resets the presentation space. The function resets the page units and page size to the values specified by the metafile. It then sets up default transformations, based on page units and size, as if the presentation space had just been created with these values, and modifies the device transformation (if necessary) to ensure that the physical size of the metafile picture is preserved. Finally, it resets the presentation space as if calling the GpiResetPS function with the GRES_ALL option.

PMF_SUPPRESS

Specifies whether to continue playing the metafile after resetting the presentation space. It can be one of the following values:

Value	Meaning
SUP_DEFAULT	Default; same as SUP_NOSUPPRESS.

Value	Meaning
SUP_NOSUPPRESS	Does not suppress the metafile.
SUP_SUPPRESS	Suppresses the metafile after the presentation space is reset as specified by the PMF_RESET option. All other options are ignored.

PMF_COLORTABLES

Specifies whether to use logical color tables from the metafile or from the application. It can be one of the following:

Value	Meaning
CTAB_DEFAULT	Default; same as CTAB_NOMODIFY.
CTAB_NOMODIFY	Uses the logical color table defined by the application. This is the default.
CTAB_REPLACE	Uses the logical color tables implied by or given in the metafile. The application's existing logical color table is overwritten.

PMF_COLORREALIZABLE

Specifies whether the logical color tables defined by the metafile should be realizable. It can be one of the following values:

Value	Meaning
CREA_DEFAULT	Default; same as CREA_REALIZE.
CREA_REALIZE	Creates realizable color tables. This is the default.
CREA_NOREALIZE	Does not create realizable color tables.

PMF_PATHBASE Specifies a reserved element. It must be zero.

PMF_RESOLVEPATH Specifies a reserved element. It must be RSP_DEFAULT or RSP_NODISCARD.

Example

This example uses the GpiPlayMetaFile function to play the given metafile. The function uses all the default actions for playing the metafile.

```
HMF hmf;
LONG cSegments;
CHAR szBuffer[80];

hmf = GpiLoadMetafile(hab, "sample.met");
GpiPlayMetafile(hps, hmf, OL, NULL, &cSegments, 80L, szBuffer);
```

See Also

DevCloseDC, DevOpenDC, GpiCreateLogColorTable, GpiCreateLogFont, GpiLoadMetaFile, GpiResetPS, GpiSetDrawingMode, GpiSetViewingTransformMatrix

■ GpiPointArc

```
LONG GpiPointArc(hps, pptl)
HPS hps;          /* presentation-space handle */
PPOINTL pptl;    /* pointer to structure for points */
```

The **GpiPointArc** function draws an arc through three points. The function uses the current arc parameters to determine the shape of the arc, then starts the arc at the current position, draws it through the first point specified by *pptl*, and ends the arc at the second point specified by *pptl*.

The **GpiPointArc** function moves the current position to the end point of the point arc.

Parameters *hps* Identifies the presentation space.
pptl Points to a POINTL structure that contains intermediate and end points. The POINTL structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is GPL_OK or GPL_HITS if the function is successful (it is GPL_HITS if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is GPL_ERROR if an error occurs.

Example This example uses the **GpiPointArc** function to draw an arc through the three points of a triangle. The **GpiPolyLine** function then draws the triangle:

```
POINTL ptlTriangle[] = { 0, 0, 100, 100, 200, 0 };

GpiMove(hps, &ptlTriangle[0]);      /* moves to start point (0, 0) */
GpiPointArc(hps, &ptlTriangle[1]);  /* draws the arc */
GpiMove(hps, &ptlTriangle[0]);      /* moves to start point (0, 0) */
GpiPolyLine(hps, &ptlTriangle[1]);  /* draws the triangle */
```

See Also **GpiFullArc**, **GpiMove**, **GpiQueryArcParams**, **GpiSetArcParams**, **GpiSetAttrs**, **GpiSetColor**, **GpiSetCurrentPosition**, **GpiSetLineType**

■ GpiPolyFillet

```
LONG GpiPolyFillet(hps, cptl, aptl)
HPS hps;          /* presentation-space handle */
LONG cptl;        /* number of points in array */
PPOINTL aptl;    /* pointer to array of structures for points */
```

The **GpiPolyFillet** function draws one or more fillets. The function draws the fillets by using the points specified by the *aptl* parameter. The function needs at least two points to draw a fillet. If exactly two points are specified, the function draws the fillet from the current position to the second point, using the first point as a control point. If more than two points are given, the function uses each point (except the last) as a control point, computing the end point of each fillet as needed. The function draws each fillet by using the current line-color, line-mix, line-width, and line-type attributes.

The **GpiPolyFillet** function moves the current position to the end point of the last fillet.

Parameters

hps Identifies the presentation space.

cptl Specifies the number of points.

aptl Points to an array of **POINTL** structures that contain the points. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

The return value is **GPL_OK** or **GPL_HITS** if the function is successful (it is **GPL_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is **GPL_ERROR** if an error occurs.

Comments

A fillet is a smooth curve whose path is controlled by three points: a start point, an end point, and a control point.

When given two points, the **GpiPolyFillet** function draws a fillet by first constructing two imaginary straight lines, one from the current position to the control point (the first point) and another from the control point to the end point (the second point). The function then draws the curve from the current position to the end point. The curve is drawn so that the first imaginary line is tangent to the curve at the current position and the second imaginary line is tangent to the curve at the second point. Only the curve is drawn, never the imaginary lines.

When given more than two points, the function constructs a series of imaginary straight lines, then draws a series of curves. The function draws the first curve from the current position to the midpoint of the second imaginary line, the second curve from the midpoint of the second line to the midpoint of the third, and so on until it draws the last curve from a midpoint to the last point specified.

The maximum number of fillets allowed in the polyfillet depends on the length of coordinates, but is at least 4000.

Example

This example uses the **GpiPolyFillet** function to draw a curve with a loop. The four points are the four points of a rectangle. The curve is drawn from the lower-left corner, through the midpoint of the top edge, and back to the lower-right corner.

```
POINTL pt1Start = { 0, 0 };
POINTL apt1[3] = { 200, 100, 0, 100, 200, 0 };

GpiMove(hps, &pt1Start); /* move to the lower-left corner */
GpiPolyFillet(hps, 3L, apt1); /* draw the curve */
```

See Also

GpiMove, **GpiPolyFilletSharp**, **GpiPolyLine**, **GpiSetAttrs**, **GpiSetColor**, **GpiSetCurrentPosition**, **GpiSetLineType**

■ GpiPolyFilletSharp

```

LONG GpiPolyFilletSharp(hps, cptl, aptl, afxSharpness)
HPS hps;           /* presentation-space handle */
LONG cptl;         /* number of points */
PPOINTL aptl;     /* pointer to array of structures for points */
PFIXED afxSharpness; /* pointer of array of structures for sharpness values */

```

The **GpiPolyFilletSharp** function creates one or more fillets. The function draws the fillets by using the control and end points specified by the *aptl* parameter and the fillet sharpness values specified by the *afxSharpness* parameter. The function draws the first fillet from the current position to the first end point, by using the first control point and first sharpness value to construct the path of the fillet. The second fillet is drawn from the first end point to the second end point using the second control point and sharpness values. The function continues with each successive point, using the last end point as the starting point for the next fillet, until the function draws one fillet for each control and end-point pair.

For each fillet, the array pointed to by *aptl* contains a control and end-point pair. The first pair of points is the control and end points for the first fillet, with the control point given first. The array pointed to by *afxSharpness* contains the sharpness values for each fillet, with the sharpness value for the first fillet given first.

The **GpiPolyFilletSharp** function moves the current position to the end point of the last fillet.

Parameters

hps Identifies the presentation space.

cptl Specifies the number of points in the array pointed to by *aptl*. This must be twice the number of fillets since each fillet requires a control and end-point pair.

aptl Points to an array of **POINTL** structures that contain the points. The **POINTL** structure has the following form:

```

typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

afxSharpness Points to an array of sharpness values giving the sharpness of successive fillets. Each value must be a fixed-point value. Each value controls the type of curve drawn for the fillet. If this value is greater than 1.0, the curve is a hyperbola. If the value is 1.0, the curve is a parabola. If the value is less than 1.0, the curve is an ellipse.

Return Value

The return value is **GPL_OK** or **GPL_HITS** if the function is successful (it is **GPL_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is **GPL_ERROR** if an error occurs.

Comments

A fillet is a smooth curve whose path is controlled by three points: a start point, an end point, and a control point. The smoothness of the fillet is controlled by a sharpness value.

The **GpiPolyFilletSharp** function draws a fillet by first constructing two imaginary straight lines, one from the start point to the control point and another from the control point to the end point. The function then draws the fillet from the start to end point, such that the first imaginary line is tangent to the fillet at the current position and the second imaginary line is tangent to the fillet at the end point.

GpiPolyFilletSharp uses the control point and the sharpness value to determine the path of the curve. The function always draws the curve through a fourth point. It locates the point by constructing two more imaginary straight lines, one from the start to end point, and another from the control point to the midpoint of this first line. The fourth point lies on the imaginary line drawn from the control point to the midpoint. It is placed such that the ratio of the lengths of the bottom and top pieces of this line is equal to the sharpness value.

The maximum number of fillets allowed depends upon the length of coordinates and is not less than 2000.

Example

This example uses the **GpiPolyFilletSharp** function to draw a curve with a loop. The curve is drawn within a rectangle. The sharpness values are chosen to draw the curve close to the control points.

```
POINTL pt1Start = { 0, 0 };
POINTL apt1[4] = { 100, 100, 200, 100, 0, 100, 200, 0 };
FIXED afx[2] = { MAKEFIXED(4, 0), MAKEFIXED(4, 0) };

GpiMove(hps, &pt1Start);      /* move to first starting point      */
GpiPolyFilletSharp(hps,      /* presentation-space handle        */
    4L,                      /* 4 points in the array            */
    apt1,                    /* pointer to array of points        */
    afx);                    /* pointer to array of sharpness values */
```

See Also

GpiMove, **GpiPolyFillet**, **GpiPolyLine**, **GpiSetAttrs**, **GpiSetColor**, **GpiSetCurrentPosition**, **GpiSetLineType**

■ GpiPolyLine

```
LONG GpiPolyLine(hps, cptl, aptl)
HPS hps;      /* presentation-space handle */
LONG cptl;    /* number of points in array */
PPOINTL aptl; /* pointer to array of structures for points */
```

The **GpiPolyLine** function draws one or more straight lines. The function draws the lines by using the points specified by the *aptl* parameter. The function needs at least one point to draw a line. If a point is specified, the function draws the line from the current position to the point. For each additional line, the function needs exactly one more point, and uses the end point of the last line as the starting point for the next. The function draws the lines by using the current values of the line-color, line-mix, line-width, and line-type attributes.

The **GpiPolyLine** function moves the current position to the end point of the last line.

Parameters

hps Identifies a presentation space.

cptl Specifies the number of points. This parameter must be greater than or equal to zero.

aptl Points to an array of POINTL structures that contains the points. The POINTL structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is `GPL_OK` or `GPL_HITS` if the function is successful (it is `GPL_HITS` if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is `GPL_ERROR` if an error occurs.

Example This example uses the `GpiPolyLine` function to draw a triangle:

```
POINTL ptlTriangle[] = { 0, 0, 100, 100, 200, 0 };
GpiMove(hps, &ptlTriangle[0]); /* moves to start point (0, 0) */
GpiPolyLine(hps, &ptlTriangle[1]); /* draws the triangle */
```

See Also `GpiLine`, `GpiMove`, `GpiSetAttrs`, `GpiSetColor`, `GpiSetCurrentPosition`, `GpiSetLineType`

■ GpiPolyMarker

```
LONG GpiPolyMarker(hps, cptl, aptl)
HPS hps; /* presentation-space handle */
LONG cptl; /* number of points */
PPOINTL aptl; /* pointer to array of structures for point */
```

The `GpiPolyMarker` function draws a marker at each point specified by the *aptl* parameter. The function places the center of each marker at the given point. The current marker set and marker-symbol attributes specify the marker to draw.

The function moves the current position to the point of the last marker.

Parameters *hps* Identifies a presentation space.

cptl Specifies the number of points.

aptl Points to an array of POINTL structures that contain the points. The POINTL structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is `GPL_OK` or `GPL_HITS` if the function is successful (it is `GPL_HITS` if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is `GPL_ERROR` if an error occurs.

Example This example uses the **GpiPolyMarker** function to draw a series of markers. It then uses the **GpiPolyLine** function to connect to markers with lines.

```
POINTL pt1Start = { 0, 0 };
POINTL apt1[5] = { 10, 8, 20, 17, 30, 28, 40, 51, 50, 46 };

GpiPolyMarker(hps, 51, apt1);
GpiMove(hps, &pt1Start);
GpiPolyLine(hps, 5L, apt1);
```

See Also **GpiMarker**, **GpiMove**, **GpiSetAttrs**, **GpiSetCurrentPosition**, **GpiSetMarkerBox**, **GpiSetMarkerSet**

I **GpiPolySpline**

LONG GpiPolySpline (*hps*, *cptl*, *aptl*)

HPS *hps*; /* presentation-space handle */
LONG *cptl*; /* number of points in array */
PPOINTL *aptl*; /* pointer to array of structures for points */

The **GpiPolySpline** function creates one or more Bezier splines. The function draws the Bezier splines by using the points specified by the *aptl* parameter. The function needs at least three points to draw a spline. If exactly three points are specified, the function draws the spline from the current position to the third point, by using the first and second points as control points. For each additional spline, the function needs exactly three more points, and uses the end point of the last spline as the next starting point. The function draws each fillet by using the current line-color, line-mix, line-width, and line-type attributes.

For each Bezier spline, the array pointed to by *aptl* contains two control points and an end point. The first triplet of points are the control and end points for the first spline, with the control points given first.

The **GpiPolySpline** function moves the current position to the last specified point.

Parameters *hps* Identifies a presentation space.

cptl Specifies the number of points in the array pointed to by *aptl*. This must be three times the number of splines since each spline requires two control points and an end point.

aptl Points to an array of **POINTL** structures that contains the points. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is **GPL_OK** or **GPL_HITS** if the function is successful (it is **GPL_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is **GPL_ERROR** if an error occurs.

Comments A Bezier spline is a smooth curve whose path is controlled by four points: a start point, an end point, and two control points.

As the **GpiPolySpline** function draws a spline, all points contribute to the direction of the path, with one point having the greatest amount of control. The point with the greatest control tends to pull the path toward it. Greatest control moves from the start point, to the first control point, to the second control point, and finally to the end point as the path progresses.

If the function draws more than one spline, it does not automatically ensure continuity of the curve at the end points. If an application wants a smooth transition from one curve to the next, it must supply the appropriate end and control points.

Example

This example uses the **GpiPolySpline** function to draw a curve. The curve is drawn within a skewed rectangle, with the bottom corners being the start and end points and the top corners being the control points.

```
POINTL pt1Start = { 0, 0 };
POINTL apt1[3] = { 0, 100, 200, 150, 200, 50 };

GpiMove(hps, &pt1Start);           /* moves to start point */
GpiPolySpline(hps,                 /* presentation-space handle */
              3L,                   /* 3 points in the array */
              apt1);                /* pointer to array of points */
```

See Also

GpiMove, **GpiSetAttrs**, **GpiSetColor**, **GpiSetCurrentPosition**, **GpiSetLineType**

■ GpiPop

BOOL GpiPop(*hps*, *cAttrs*)

HPS *hps*; /* presentation-space handle */
LONG *cAttrs*; /* number of attributes to restore */

The **GpiPop** function restores one or more primitive attributes by popping the previous attribute values from the attribute stack. The system saves the previous value of a primitive attribute, such as color, line type, and fill pattern, on the attribute stack whenever an application changes an attribute while the attribute mode is **AM_PRESERVE**. The function pops the number of attributes specified by *cAttrs* from the stack in last-in, first-out order.

Parameters

hps Identifies a presentation space.

cAttrs Specifies the number of attributes to restore.

Return Value

The return value is **GPL_OK** if the function is successful or **GPL_ERROR** if an error occurred.

Errors

Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

```
PMERR_INV_MICROPS_FUNCTION
PMERR_PRIMITIVE_STACK_EMPTY
```

Comments

Although **GpiPop** can be used in an area or path bracket, an application must ensure that the attribute to be restored is valid within the bracket. Once an attribute is on the stack, there is no way to check it for validity.

The attribute stack is especially useful when you are drawing segments. Any attributes changed by the segment can be quickly restored by popping the stack. Note that a segment automatically pops the stack when it returns, so a call to the **GpiPop** function is not required.

Example

This example uses the **GpiPop** function to restore the fill pattern and color attribute after painting a path:

```
GpiSetAttrMode(hps, AM_PRESERVE); /* preserves attributes on stack */
.
.
GpiSetColor(hps, CLR_RED); /* sets color to red */
GpiSetPattern(hps, PATSYM_DIAG1); /* sets pattern to a diagonal */
GpiPaintRegion(hps, 3L);
GpiPop(hps, 2L); /* restores values of last two attributes set */
```

See Also

GpiRestorePS, GpiSavePS

GpiPtInRegion

LONG GpiPtInRegion(hps, hrgn, pptl)

HPS *hps*; /* presentation-space handle */
HRGN *hrgn*; /* region handle */
PPOINTL *pptl*; /* pointer to structure for point */

The **GpiPtInRegion** function checks whether a point lies in the region specified by the *hrgn* parameter. The function checks the region only if the device context containing the region is associated with the given presentation space.

Parameters

hps Identifies a presentation space.

hrgn Identifies a region.

pptl Points to a **POINTL** data structure that contains the coordinates of the point. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

The return value is **PRGN_OUTSIDE** or **PRGN_INSIDE** if the function is successful, or **PRGN_ERROR** if an error occurs.

See Also

GpiRectInRegion

GpiPtVisible

LONG GpiPtVisible(hps, pptl)

HPS *hps*; /* presentation-space handle */
PPOINTL *pptl*; /* pointer to structure for point */

The **GpiPtVisible** function checks whether a point is visible on the device associated with the specified presentation space. A point is visible if it lies within the intersection of the current graphics field, viewing limit, clip path, clip region, and visible region (if any).

Parameters *hps* Identifies a presentation space.
pptl Points to a **POINTL** data structure that contains the coordinates of the point. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is **PVIS_VISIBLE** or **PVIS_INVISIBLE** if the function is successful or **PVIS_ERROR** if an error occurred.

See Also **GpiConvert**, **GpiQueryPel**

■ GpiPutData

LONG GpiPutData(*hps*, *cmdFormat*, *pcb*, *pb*)

HPS *hps*; /* presentation-space handle */
LONG *cmdFormat*; /* coordinate type */
PLONG *pcb*; /* pointer to variable for length of order data */
PBYTE *pb*; /* pointer to buffer for order data */

The **GpiPutData** function draws the graphics orders given in the buffer pointed to by the *pb* parameter. The function carries out the graphics operation specified by each graphics order. The buffer can contain any number of graphics orders as long as the buffer does not exceed 63K. The *pcb* parameter specifies the number of bytes of graphics-order data in the buffer. The function converts the coordinates in the graphics orders if the format specified by the *cmdFormat* parameter is not the same as the format used by the presentation space.

The **GpiPutData** function is used typically with the **GpiGetData** function to copy graphics orders from one segment to another. For convenience, the last order in the buffer does not have to be complete. If the order is not complete, the function does not copy the order. Instead, it replaces the count in *pcb* with the count of bytes copied. This count can be used to locate the incomplete order in the buffer.

Parameters *hps* Identifies a presentation space.
cmdFormat Specifies the type of coordinates used in the graphics orders. It can be one of the following values:

Value	Meaning
DFORM_S370SHORT	Uses S/370-format short (2-byte) integers.
DFORM_PCSHORT	Uses PC-format short (2-byte) integers.
DFORM_PCLONG	Uses PC-format long (4-byte) integers.

pcb Points to the count of bytes in the buffer pointed to by the *pb* parameter. After copying the data, the function replaces the count with the number of bytes copied.

pb Points to the buffer that contains the graphics-order data. It must not contain more than 63K of data.

Return Value The return value is `GPL_OK` or `GPL_HITS` if the function is successful (it is `GPL_HITS` if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is `GPL_ERROR` if an error occurs.

Errors Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_DATA_TOO_LONG
PMERR_INV_EDIT_MODE
PMERR_INV_ELEMENT_POINTER
PMERR_INV_LENGTH_OR_COUNT
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_ORDER_LENGTH
```

Comments The current drawing mode determines whether the function draws the graphics orders or copies them to a segment. To copy the graphics orders to the currently open segment, an application must set the current segment editing mode to `SEGEM_INSERT` and move the element pointer to the last element in the segment. The function can be used in an element bracket if the graphics-order data does not contain an element bracket.

Example This example uses the `GpiPutData` function to copy graphics orders from one segment to another:

```
LONG fFormat = DFORM_NOCONV; /* do not convert coordinates */
LONG offSegment = 0L; /* offset in segment */
LONG offNextElement = 0L; /* offset in segment to next element */
LONG cb = 0L; /* bytes retrieved */
BYTE abBuffer[512];

GpiOpenSegment(hps, 3L); /* open segment to receive the data */
do {
    offSegment += cb;
    offNextElement = offSegment;
    cb = GpiGetData(hps, 2L, &offNextElement, fFormat, 512L, abBuffer);

    /* Put data in other segment. */

    if (cb > 0L) GpiPutData(hps, /* presentation-space handle */
        fFormat, /* format of coordinates */
        &cb, /* number of bytes in buffer */
        abBuffer); /* buffer with graphics-order data */

} while (cb > 0L);
GpiCloseSegment(hps); /* close segment that received data */
```

See Also `GpiBeginElement`, `GpiEndElement`, `GpiGetData`, `GpiSetDrawingMode`, `GpiSetEditMode`

■ GpiQueryArcParams

BOOL GpiQueryArcParams(*hps*, *parcp*)

HPS *hps*; /* presentation-space handle */

ARCPARAMS *parcp*; /* pointer to structure for arc parameters */

The **GpiQueryArcParams** function retrieves the current arc parameters used to draw arcs, circles, and ellipses. The function cannot be used in an open segment when the drawing mode is **DM_RETAIN**.

Parameters *hps* Identifies the presentation space.
 parcp Points to the **ARCPARAMS** structure that receives the current arc parameters. The **ARCPARAMS** structure has the following form:

```
typedef struct _ARCPARAMS {
    LONG iP;
    LONG IQ;
    LONG IR;
    LONG IS;
} ARCPARAMS;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is **GPI_OK** if the function is successful or **GPI_ERROR** if an error occurred.

See Also **GpiSetArcParams**, **GpiSetDrawingMode**

■ GpiQueryAttrMode

LONG GpiQueryAttrMode(*hps*)

HPS *hps*; /* presentation-space handle */

The **GpiQueryAttrMode** function retrieves the current value of the attribute mode, as set by the **GpiSetAttrMode** function.

Parameters *hps* Identifies the presentation space.

Return Value The return value is the current attribute mode if the function is successful or **AM_ERROR** if an error occurs.

Errors Use the **WinGetLastError** function to retrieve the error value, which may be the following:

PMERR_INV_MICROPS_FUNCTION

See Also **GpiSetAttrMode**

■ GpiQueryAttrs

LONG GpiQueryAttrs (*hps*, *lPrimType*, *flAttrsMask*, *pbunAttrs*)

HPS *hps*; /* presentation-space handle */
LONG *lPrimType*; /* primitive type */
ULONG *flAttrsMask*; /* attribute mask */
PBUNDLE *pbunAttrs*; /* pointer to buffer for nondefault attributes */

The **GpiQueryAttrs** function retrieves the current attributes for the specified primitive type. The function copies the attribute values specified by the *flAttrsMask* parameter to the buffer pointed to by the *pbunAttrs* parameter, then returns a mask that specifies which attributes have the default values. The function sets the bit in the mask if the corresponding attribute has its default value.

The **GpiQueryAttrs** function cannot be used in an open segment when the drawing mode is **DM_RETAIN**.

Parameters

hps Identifies the presentation space.

lPrimType Specifies which primitive type to retrieve attributes for. It can be one of the following values:

Value	Meaning
PRIM_AREA	Area primitives
PRIM_CHAR	Character primitives
PRIM_IMAGE	Image primitives
PRIM_LINE	Line and arc primitives
PRIM_MARKER	Marker primitives

flAttrsMask Specifies which attributes to retrieve. The values for this parameter depend on the primitive type specified by the *lPrimType* parameter. This parameter can be any combination of the following values for a specific type:

Type	Values
PRIM_AREA	ABB_COLOR, ABB_BACK_COLOR, ABB_MIX_MODE, ABB_BACK_MIX_MODE, ABB_SET, ABB_SYMBOL, ABB_REF_POINT
PRIM_CHAR	CBB_COLOR, CBB_BACK_COLOR, CBB_MIX_MODE, CBB_BACK_MIX_MODE, CBB_SET, CBB_MODE, CBB_BOX, CBB_ANGLE, CBB_SHEAR, CBB_DIRECTION
PRIM_IMAGE	IBB_COLOR, IBB_BACK_COLOR, IBB_MIX_MODE, IBB_BACK_MIX_MODE
PRIM_LINE	LBB_COLOR, LBB_MIX_MODE, LBB_WIDTH, LBB_GEOM_WIDTH, LBB_TYPE, LBB_END, LBB_JOIN
PRIM_MARKER	MBB_COLOR, MBB_BACK_COLOR, MBB_MIX_MODE, MBB_BACK_MIX_MODE, MBB_SET, MBB_SYMBOL, MBB_BOX

If this parameter is zero, the function does not retrieve attributes but still returns a mask specifying the attributes with default values.

pbunAttrs Points to the buffer that receives the attribute values specified by the *fAttrsMask* parameter. The buffer format depends on the primitive type specified by the *lPrimType* parameter. The following structures can be used for the specified primitive types:

Type	Structure
PRIM_AREA	AREABUNDLE
PRIM_CHAR	CHARBUNDLE
PRIM_IMAGE	IMAGEBUNDLE
PRIM_LINE	LINEBUNDLE
PRIM_MARKER	MARKERBUNDLE

Return Value The return value is the default mask if the function is successful or `GPI_ALTError` if an error occurred.

Errors Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_PRIMITIVE_TYPE`

Example This example uses the `GpiQueryAttrs` function to retrieve the current attributes for the line primitive:

```
LINEBUNDLE lbnd;
LONG flDefMask;

flDefMask = GpiQueryAttrs(hps, /* presentation-space handle */
    PRIM_LINE, /* line primitive */
    LBB_COLOR | /* line color */
    LBB_MIX_MODE | /* color-mix mode */
    LBB_WIDTH | /* line width */
    LBB_GEOM_WIDTH | /* geometric-line width */
    LBB_TYPE | /* line style */
    LBB_END | /* line-end style */
    LBB_JOIN, /* line-join style */
    &lbnd); /* buffer for attributes */

if (flDefMask & LBB_COLOR)
    /* The line color has the default value. */
```

See Also `GpiSetAttrs`, `GpiSetDrawingMode`

■ GpiQueryBackColor

LONG GpiQueryBackColor(hps)

HPS hps; /* presentation-space handle */

The `GpiQueryBackColor` function retrieves the current value of the line background-color attribute, as set by the `GpiSetBackColor` function. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

Parameters *hps* Identifies the presentation space.

Return Value The return value is the background color if the function is successful or CLR_ERROR if an error occurred.

See Also GpiQueryBackMix, GpiQueryColor, GpiSetBackColor

■ GpiQueryBackMix

LONG GpiQueryBackMix(*hps*)

HPS *hps*; /* presentation-space handle */

The **GpiQueryBackMix** function retrieves the current value of the line background-color mix mode, as set by the **GpiSetBackMix** function.

Parameters *hps* Identifies the presentation space.

Return Value The return value is the background-color mix mode if the function is successful or BM_ERROR if an error occurred.

See Also GpiQueryBackColor, GpiQueryMix, GpiSetBackMix

■ GpiQueryBitmapBits

LONG GpiQueryBitmapBits(*hps, lScanStart, cScan, pbBuffer, pbmi*)

HPS *hps*; /* presentation-space handle */

LONG *lScanStart*; /* number for first scan line to retrieve */

LONG *cScan*; /* number of scan lines to retrieve */

PBYTE *pbBuffer*; /* pointer to buffer for bitmap image data */

PBITMAPINFO *pbmi*; /* pointer to structure for bitmap info */

The **GpiQueryBitmapBits** function copies image data from a bitmap to the buffer pointed to by the *pbBuffer* parameter. The function copies the image data from the bitmap currently set for the presentation space. The presentation space must be associated with a memory device context.

To copy the image data, the function needs the count of planes and adjacent color bits specified in the fields of the structure pointed to by the *pbmi* parameter. That is, the *cPlanes* and *cBitCount* fields must be set before you call the function. Also, the *cbFix* field must be set to 12. The function then copies the image data to the buffer. The buffer must have sufficient space to hold all the bytes of image data being copied. The number of bytes for the buffer is equal to the number of scan lines to copy, multiplied by the width of the bitmap in bytes (rounded up to the next multiple of 4), multiplied by the number of color planes. The width has to be a multiple of 4, since the function rounds the length of each scan line to a multiple of 4 bytes before copying. Also, the width must be multiplied by the number of adjacent color bits before rounding.

After copying the image data, the **GpiQueryBitmapBits** function fills the remaining fields in the structure pointed to by *pbmi*. These fields are the width and height of the bitmap and the array of RGB color values for the bitmap pels. An application must make sure there is sufficient space in the structure to receive all elements of the array of RGB color values. The number of elements in the array depends on the format of the bitmap.

Parameters

hps Identifies the presentation space.

lScanStart Specifies the number of the first scan line to copy to the buffer. If this parameter is zero, the function copies the first scan line in the bitmap.

cScan Specifies the number of scan lines to copy.

pbBuffer Points to the buffer that receives the bitmap image data. It must be large enough to hold all the bytes of the image data, from the scan line specified by the *lScanStart* parameter to the end of the bitmap.

pbmi Points to the BITMAPINFO structure that receives the bitmap information table. The BITMAPINFO structure has the following form:

```
typedef struct _BITMAPINFO {
    ULONG   cbFix;
    USHORT  cx;
    USHORT  cy;
    USHORT  cPlanes;
    USHORT  cBitCount;
    RGB     argbColor[1];
} BITMAPINFO;
```

Depending on the format of the given bitmap, an application may need to allocate extra bytes for the structure to hold the additional elements for the *argbColor* field. For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

The return value is the number of scan lines retrieved if the function is successful or *GPI_ALTEERROR* if an error occurred.

Errors

Use the *WinGetLastError* function to retrieve the error value, which may be one of the following:

```
PMERR_INV_DC_TYPE
PMERR_INV_INFO_TABLE
```

Comments

If the requested color format is not the same as the bitmap’s color format, the function converts the bitmap image data to the requested format.

Example

This example uses the *GpiQueryBitmapBits* to copy the image data of a bitmap from a presentation space associated with a memory device context.

```
BITMAPINFOHEADER bmp = { 12, 640, 350, 1, 1 };
LONG cbBuffer, cbBitmapInfo;
SEL selBuffer, selBitmapInfo;
PBYTE pbBuffer;
PBITMAPINFO pbmi;

/*
 * Compute the size of the image-data buffer
 * and the bitmap information structure.
 */

cbBuffer = (((bmp.cBitCount * bmp.cx) + 31) / 32)
           * 4 * bmp.cy * bmp.cPlanes;
cbBitmapInfo = sizeof(BITMAPINFO) +
               (sizeof(RGB) * (1 << bmp.cBitCount));

/* Allocate memory for the image data-buffer
 * and the bitmap information structure.
 */

DosAllocSeg(cbBuffer, &selBuffer, SEG_NONSHARED);
pbBuffer = MAKEP(selBuffer, 0);
DosAllocSeg(cbBitmapInfo, &selBitmapInfo, SEG_NONSHARED);
pbmi = MAKEP(selBitmapInfo, 0);
```

```

/* Copy the image data. */
pbmi->cbFix = 12; pbmi->cPlanes = 1; pbmi->cBitCount = 1;
GpiQueryBitmapBits(hps, OL, (LONG) bmp.cy, pbBuffer, pbmi);

```

See Also **GpiLoadBitmap, GpiQueryBitmapParameters, GpiSetBitmapBits**

I GpiQueryBitmapDimension

```

BOOL GpiQueryBitmapDimension(hbm, psizl)
HBITMAP hbm;        /* bitmap handle                            */
PSIZEL psizl;      /* pointer to structure for bitmap size info */

```

The **GpiQueryBitmapDimension** function retrieves the width and height of a bitmap, as specified by a previous call to the **GpiSetBitmapDimension** function. If the bitmap dimensions have not been set by **GpiSetBitmapDimension**, the width and height are zero.

Parameters *hbm* Identifies the bitmap.
psizl Points to the **SIZEL** structure that receives the width and height of the bitmap in 0.1 millimeter units. The **SIZEL** structure has the following form:

```

typedef struct _SIZEL {
    LONG cx;
    LONG cy;
} SIZEL;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is **GPL_OK** if the function is successful or **GPL_ERROR** if an error occurred.

See Also **GpiQueryBitmapParameters, GpiSetBitmapDimension**

I GpiQueryBitmapHandle

```

HBITMAP GpiQueryBitmapHandle(hps, lcid)
HPS hps;            /* presentation-space handle */
LONG lcid;         /* local identifier            */

```

The **GpiQueryBitmapHandle** function retrieves the handle of the bitmap currently tagged with the specified local identifier. The function returns a null handle if a bitmap is not currently tagged with the specified local identifier.

Parameters *hps* Identifies the presentation space.
lcid Specifies the local identifier.

Return Value The return value is a handle to the bitmap if the function is successful or **GPL_ERROR** if an error occurred.

Errors Use the **WinGetLastError** function to retrieve the error value, which may be the following:

PMERR_BITMAP_NOT_FOUND

See Also **GpiSetBitmapId**

■ GpiQueryBitmapParameters

```

BOOL GpiQueryBitmapParameters(hbm, pbmp)
HBITMAP hbm; /* bitmap handle */
PBITMAPINFOHEADER pbmp; /* pointer to structure for bitmap info */

```

The **GpiQueryBitmapParameters** function retrieves information about the bitmap identified by the *hbm* parameter. The function copies the bitmap width, height, and number of color planes and adjacent color bits to the structure pointed to by the *pbmp* parameter.

Parameters *hbm* Identifies the bitmap.

pbmp Points to the **BITMAPINFOHEADER** structure that receives the information for the specified bitmap. The **BITMAPINFOHEADER** structure has the following form:

```

typedef struct _BITMAPINFOHEADER {
    ULONG    cbFix;
    USHORT   cx;
    USHORT   cy;
    USHORT   cPlanes;
    USHORT   cBitCount;
} BITMAPINFOHEADER;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is **GPL_OK** if the function is successful or **GPL_ERROR** if an error occurred.

See Also **GpiCreateBitmap**, **GpiLoadBitmap**, **GpiQueryBitmapDimension**

■ GpiQueryBoundaryData

```

BOOL GpiQueryBoundaryData(hps, prcl)
HPS hps; /* presentation-space handle */
PRECTL prcl; /* pointer to structure for boundary data */

```

The **GpiQueryBoundaryData** function retrieves the current boundary data. Boundary data is the smallest rectangle in model space that encloses previous graphics output. The previous output includes all output since the application reset the boundary data by using the **GpiResetBoundaryData** function or started accumulating boundary data by using the **DCTL_BOUNDARY** option of the **GpiSetDrawControl** function.

The boundary data is inclusive, meaning some output may lie on one or more edges of the given rectangle. If there has been no output, the boundary data is empty. In this case, the values for the upper-right corner in the given rectangle will be less than the values for the lower-left corner.

Parameters *hps* Identifies the presentation space.

prcl Points to the **RECTL** structure that receives the boundary data. The **RECTL** structure has the following form:

```

typedef struct _RECTL {
    LONG    xLeft;
    LONG    yBottom;
    LONG    xRight;
    LONG    yTop;
} RECTL;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

- Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_MICROPS_FUNCTION`

- Example** This example uses the `GpiQueryBoundaryData` function to retrieve the rectangle enclosing the output. The boundary data is then used to draw a border around the output.

```
POINTL ptlStart = { 0, 0 };
POINTL ptlTriangle[3] = { 100, 100, 200, 0, 0, 0 };
RECTL rcl;

GpiSetDrawControl(hps,
    DCTL_BOUNDARY, DCTL_ON);           /* accumulate boundary data */

GpiMove(hps, &ptlStart);               /* produce output */
GpiPolyLine(hps, 3L, &ptlTriangle);

GpiQueryBoundaryData(hps, &rcl);       /* copy boundary data to rcl */
if (rcl.xLeft < rcl.xRight) {        /* verify output exists */
    ptlStart.x = rcl.xLeft; ptlStart.y = rcl.yBottom;
    GpiMove(hps, &ptlStart);         /* move to lower-right corner */
    ptlStart.x = rcl.xRight; ptlStart.y = rcl.yTop;
    GpiBox(hps, DRO_OUTLINE, &ptlStart, 0L, 0L); /* draw border */
}
```

- See Also** `GpiResetBoundaryData`, `GpiSetDrawControl`

■ GpiQueryCharAngle

```
BOOL GpiQueryCharAngle(hps, pgradlAngle)
HPS hps;           /* presentation-space handle */
PGRADIENTL pgradlAngle; /* pointer to structure for baseline-angle point */
```

The `GpiQueryCharAngle` function retrieves the current value of the character-baseline angle. The character-baseline angle is set by the `GpiSetCharAngle` function.

The `GpiQueryCharAngle` function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

- Parameters** *hps* Identifies the presentation space.

pgradlAngle Points to the `GRADIENTL` structure that receives a point that specifies the end of a vector defining the baseline angle. The `GRADIENTL` structure has the following form:

```
typedef struct _GRADIENTL {
    LONG x;
    LONG y;
} GRADIENTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

See Also `GpiQueryAttrs`, `GpiQueryCharMode`, `GpiQueryCharShear`, `GpiSetCharAngle`

■ GpiQueryCharBox

BOOL GpiQueryCharBox(*hps*, *psizfxBox*)

HPS *hps*; /* presentation-space handle */

PSIZEF *psizfxBox*; /* pointer to structure for character-box size */

The `GpiQueryCharBox` function retrieves the current value of the character-box attribute, as set by the `GpiSetCharBox` function. This function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

Parameters *hps* Identifies the presentation space.

psizfxBox Points to the `SIZEF` structure that receives the character-box size. The `SIZEF` structure has the following form:

```
typedef struct _SIZEF {
    FIXED cx;
    FIXED cy;
} SIZEF;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

See Also `GpiQueryAttrs`, `GpiQueryCharMode`, `GpiQueryDefCharBox`, `GpiSetCharBox`, `GpiSetDrawingMode`

■ GpiQueryCharDirection

LONG GpiQueryCharDirection(*hps*)

HPS *hps*; /* presentation-space handle */

The `GpiQueryCharDirection` function retrieves the current value of the character-direction attribute. This function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

Parameters *hps* Identifies the presentation space.

Return Value The return value is the current character-direction attribute if the function is successful or `CHDIRN_ERROR` if an error occurred.

Comments Under MS OS/2 version 1.1, the only character directions available are `CHDIRN_DEFAULT` and `CHDIRN_LEFTRIGHT`.

See Also `GpiSetCharDirection`, `GpiSetDrawingMode`

■ GpiQueryCharMode

LONG GpiQueryCharMode (*hps*)

HPS *hps*; /* presentation-space handle */

The **GpiQueryCharMode** function retrieves the current value of the character-mode attribute, as set by the **GpiSetCharMode** function. This function cannot be used in an open segment when the drawing mode is **DM_RETAIN**.

Parameters *hps* Identifies the presentation space.

Return Value The return value is the current character-mode attribute if the function is successful or **CM_ERROR** if an error occurred.

See Also **GpiQueryAttrs**, **GpiQueryCharAngle**, **GpiQueryCharShear**, **GpiSetCharMode**, **GpiSetDrawingMode**

■ GpiQueryCharSet

LONG GpiQueryCharSet (*hps*)

HPS *hps*; /* presentation-space handle */

The **GpiQueryCharSet** function retrieves the character-set local identifier. This function cannot be used in an open segment when the drawing mode is set to **DM_RETAIN**.

Parameters *hps* Identifies the presentation space.

Return Value The return value is the local identifier for the current character set if the function is successful or **LCID_ERROR** if an error occurred.

See Also **GpiQueryAttrs**, **GpiSetCharSet**, **GpiSetDrawingMode**

■ GpiQueryCharShear

BOOL GpiQueryCharShear (*hps*, *pptlShear*)

HPS *hps*; /* presentation-space handle */

PPOINTL *pptlShear*; /* pointer to structure for shear-vector point */

The **GpiQueryCharShear** function retrieves the value of the current character-shear angle. This function cannot be used in an open segment when the drawing mode is **DM_RETAIN**.

Parameters *hps* Identifies the presentation space.

pptlShear Points to the **POINTL** structure that receives the point defining the character-shear vector. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

- Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.
- See Also** `GpiQueryAttrs`, `GpiQueryCharAngle`, `GpiQueryCharMode`, `GpiSetCharShear`, `GpiSetDrawingMode`

■ GpiQueryCharStringPos

BOOL GpiQueryCharStringPos(*hps*, *flOptions*, *cchString*, *pchString*, *adx*, *aptl*)

HPS *hps*; /* presentation-space handle */
ULONG *flOptions*; /* option flags */
LONG *cchString*; /* length of the string */
PCH *pchString*; /* pointer to string to examine */
PLONG *adx*; /* pointer to array for increment values */
PPOINTL *aptl*; /* pointer to array of structures for points */

The `GpiQueryCharStringPos` function determines a position for each character in the string pointed to by the *pchString* parameter. Each position is the position of the character in world coordinates as if it were drawn by using the `GpiCharStringPos` function.

The `GpiQueryCharStringPos` function copies the character positions to the array of structures pointed to by the *aptl* parameter. It uses the current character attributes or the array of vector increments specified by the *adx* parameter to determine the positions. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

Parameters

hps Identifies the presentation space.

flOptions Specifies whether to use the vector increments specified by the *adx* parameter. It can be one of the following values:

Value	Meaning
0	Advances the current position after each character by using the width of the character. The <i>adx</i> parameter is ignored.
<code>CHS_VECTOR</code>	Advances the current position after each character by using the next value in the array <i>adx</i> . The current character direction defines the direction in which the current position is advanced.

cchString Specifies the length of the string pointed to by the *pchString* parameter.

pchString Points to the character string to examine.

adx Points to an array of increment values. Each value is a 4-byte signed integer specifying the distance (in world coordinates) to advance the current position for each character. There must be one value for each character in the string. The first element specifies the distance for the first character, the second element for the second character, and so on.

aptl Points to the array of `POINTL` structures that receives the position (in world coordinates) of each character in the string. The `POINTL` structure has the following form:


```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

Example This example uses the `GpiQueryCharStringPos` function to determine the location of each character in the string. Vector increments are not used.

```
POINTL aptl[11];

GpiQueryCharStringPos(hps, /* presentation-space handle */
                     OL, /* do not use vector increments */
                     11L, /* 11 characters in string */
                     "This string", /* character string */
                     NULL, /* no vector increments */
                     aptl); /* array of structures for points */
```

See Also `GpiCharStringPos`, `GpiQueryCharStringPosAt`, `GpiSetDrawingMode`

■ GpiQueryCharStringPosAt

```
BOOL GpiQueryCharStringPosAt(hps, pptlStart, flOptions, cchString, pchString, adx, aptl)
```

```
HPS hps; /* presentation-space handle */
PPOINTL pptlStart; /* pointer to structure for starting point */
ULONG flOptions; /* option flag */
LONG cchString; /* length of the string */
PCH pchString; /* pointer to string to examine */
PLONG adx; /* pointer to array for increment values */
PPOINTL aptl; /* pointer to array of structures for points */
```

The `GpiQueryCharStringPosAt` function determines a position for each character in the character string pointed to by the `pchString` parameter. The positions are determined as if the application had called the `GpiCharStringPosAt` function and are specified in world coordinates.

The `GpiQueryCharStringPosAt` function copies the character positions to the array of structures pointed to by the `aptl` parameter. It uses the current character attributes or the array of vector increments specified by the `adx` parameter to determine the positions. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

Parameters *hps* Identifies the presentation space.

pptlStart Points to the `POINTL` structure that specifies the starting point (in world coordinates) of the character string. The `POINTL` structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

flOptions Specifies whether to use the vector increments specified by the *adx* parameter. It can be one of the following values:

Value	Meaning
0	Advances the current position after each character by using the width of the character. The <i>adx</i> parameter is ignored.
CHS_VECTOR	Advances the current position after each character by using the next value in the array <i>adx</i> . The current character direction defines the direction in which the current position is advanced.

cchString Specifies the length (in bytes) of the string pointed to by the *pchString* parameter.

pchString Points to the character string to examine.

adx Points to an array of increment values. Each value is a 4-byte signed integer specifying the distance (in world coordinates) to advance the current position for each character. There must be one value for each character in the string. The first element specifies the distance for the first character, the second element for the second character, and so on.

aptl Points to the array of POINTL structures that receives the position (in world coordinates) of each character in the string.

Return Value The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

Example This example uses the `GpiQueryCharStringPosAt` function to determine the location of each character in the string. Vector increments are not used.

```
POINTL ptlStart = { 100, 100 };
POINTL aptl[11];

GpiQueryCharStringPosAt(hps, /* presentation-space handle */
                        &ptlStart, /* starting point for string */
                        0L, /* do not use vector increments */
                        11L, /* 11 characters in string */
                        "This string", /* character string */
                        NULL, /* no vector increments */
                        aptl); /* array of structures for points */
```

See Also `GpiCharStringPosAt`, `GpiQueryCharStringPos`, `GpiSetDrawingMode`

■ GpiQueryClipBox

```
LONG GpiQueryClipBox(hps, prcl)
```

```
HPS hps; /* presentation-space handle */
```

```
PRECTL prcl; /* pointer to structure for clip box */
```

The `GpiQueryClipBox` function retrieves the world coordinates of the smallest rectangle that encloses the intersection of the current graphics field, viewing limit, clip path, clip region, and visible region (if any). If the clip box is empty, the function sets the left and right sides of the rectangle to equal values.

- Parameters** *hps* Identifies the presentation space.
prcl Points to the **RECTL** structure that receives the coordinates of the clip box. The **RECTL** structure has the following form:
- ```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```
- For a full description, see Chapter 4, “Types, Macros, Structures.”
- Return Value**     The return value is **RGN\_NULL**, **RGN\_RECT**, or **RGN\_COMPLEX** if the function is successful or **RGN\_ERROR** if an error occurred.
- See Also**           **GpiQueryClipRegion**, **GpiQueryGraphicsField**, **GpiQueryRegionBox**, **GpiQueryViewingLimits**

## ■ GpiQueryClipRegion

---

**HRGN GpiQueryClipRegion**(*hps*)

**HPS** *hps*;     /\* presentation-space handle \*/

The **GpiQueryClipRegion** function retrieves the handle of the currently selected clip region.

- Parameters**     *hps* Identifies the presentation space.
- Return Value**     The return value is the handle of the clip region, if one is selected, or **NULL** if no clip region is selected. The return value is **HRGN\_ERROR** if an error occurred.
- See Also**           **GpiQueryClipBox**, **GpiQueryGraphicsField**, **GpiQueryViewingLimits**, **GpiSetClipRegion**

## ■ GpiQueryColor

---

**LONG GpiQueryColor**(*hps*)

**HPS** *hps*;     /\* presentation-space handle \*/

The **GpiQueryColor** function returns the current value of the line-color attribute, as set by the **GpiSetColor** function. The function cannot be used in an open segment when the drawing mode is **DM\_RETAIN**.

- Parameters**     *hps* Identifies the presentation space.
- Return Value**     The return value is the current line-color attribute if the function is successful or **CLR\_ERROR** if an error occurred.
- See Also**           **GpiQueryAttrs**, **GpiSetColor**

## ■ GpiQueryColorData

**BOOL GpiQueryColorData**(*hps, clData, alData*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *clData*; /\* number of elements \*/

**PLONG** *alData*; /\* three-element array \*/

The **GpiQueryColorData** function retrieves the format flag and index-value range for the current logical color table. The format flag specifies whether the color table has the default, indexed-RGB, or RGB format. The index-value range specifies the first and last indexes in the table. (These apply to color tables having the default or indexed RGB formats.)

The function typically copies the format flags and the first and last index values to the array pointed to by the *alData* parameter; however, the function uses the *clData* parameter to determine the number of values to copy. The *clData* parameter must be set to 3 in order to copy all values.

### Parameters

*hps* Identifies the presentation space.

*clData* Specifies the number of values to copy to the array pointed to by the *alData* parameter.

*alData* Points to the array that receives the format flag and index-value range. The elements of the array have the following meanings:

| Element index   | Meaning                                                                                                                                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| QCD_LCT_FORMAT  | Specifies the format of color table. It is LCOLF_DEFAULT if the current color table is the default color table; LCOLF_INDRGB if the color table translates indices to RGB color values; or LCOLF_RGB if the color-table indices and RGB color values are equal. |
| QCD_LCT_LOINDEX | Specifies the smallest color index. For the default color table, the smallest index is zero.                                                                                                                                                                    |
| QCD_LCT_HIINDEX | Specifies the largest color index. For the default color table, the largest index is one less than the maximum number of entries in the table.                                                                                                                  |

### Return Value

The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

### Example

This example uses the **GpiQueryColorData** function to retrieve the smallest color-table index. The **GpiQueryLogColorTable** function is then used to retrieve the RGB color value for this index.

```
LONG alData[3];
LONG alColor[1];
```

```
GpiQueryColorData(hps, 3L, alData);
GpiQueryLogColorTable(hps, 0L, alData[QCD_LCT_LOINDEX], 1L, alColor);
```

### See Also

**GpiQueryLogColorTable**, **GpiQueryNearestColor**, **GpiQueryRealColors**, **GpiQueryRGBColor**

## ■ GpiQueryColorIndex

---

**LONG GpiQueryColorIndex**(*hps*, *flClrType*, *IRgbColor*)

**HPS** *hps*; /\* presentation-space handle \*/

**ULONG** *flClrType*; /\* color type \*/

**LONG** *IRgbColor*; /\* RGB color value \*/

The **GpiQueryColorIndex** function returns a color index for the specified RGB color value. The function maps the RGB color value to the closest match in the physical palette for the device associated with the presentation space. It then returns the index in the current logical color table that corresponds to this physical-palette color.

If the current logical color table is realizable but has not been realized, the **LCOLOPT\_REALIZED** option maps the RGB color value as if the logical color table has been realized. Since realizing a logical color table affects the contents of the physical palette, the color index value returned with this option may be different than the value returned without the option.

**Parameters**     *hps*    Identifies the presentation space.

*flClrType*    Specifies whether the color index is based on realized colors. If it is **LCOLOPT\_REALIZED**, the function returns a color index based on the colors in the physical palette after the current logical color table is realized. If the parameter is zero, the function returns a color index based on the colors in the current physical palette.

*IRgbColor*    Specifies the RGB color value.

**Return Value**    The return value is a color index that is the closest possible match of the specified color if the function is successful, or **QLCT\_ERROR** if an error occurred.

**See Also**        **GpiQueryLogColorTable**, **GpiQueryRGBColor**

## ■ GpiQueryCp

---

**USHORT GpiQueryCp**(*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The **GpiQueryCp** function returns the identifier of the current graphics code page, as set by the **GpiSetCp** function. The system uses the current graphics code-page identifier as the default code-page identifier when the **GpiCreateLogFont** function selects fonts.

**Parameters**     *hps*    Identifies the presentation space.

**Return Value**    The return value is the currently selected graphics code-page identifier if the function is successful, or **GPL\_ERROR** if an error occurred.

**See Also**        **GpiCreateLogFont**, **GpiQueryFontMetrics**, **GpiSetCp**

## ■ GpiQueryCurrentPosition

---

```

BOOL GpiQueryCurrentPosition(hps, pptl)
HPS hps; /* presentation-space handle */
PPOINTL pptl; /* pointer to structure for current position */

```

The **GpiQueryCurrentPosition** function returns the value of the current position. The function cannot be used in an open segment when the drawing mode is **DM\_RETAIN**.

### Parameters

*hps* Identifies the presentation space.

*pptl* Points to a **POINTL** structure that receives the current position. The **POINTL** structure has the following form:

```

typedef struct _POINTL {
 LONG x;
 LONG y;
} POINTL;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

### Return Value

The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

### See Also

**GpiMove**, **GpiSetCurrentPosition**

## ■ GpiQueryDefaultViewMatrix

---

```

BOOL GpiQueryDefaultViewMatrix(hps, cElements, pmatlf)
HPS hps; /* presentation-space handle */
LONG cElements; /* number of elements */
PMATRIXLF pmatlf; /* pointer to structure for transformation matrix */

```

The **GpiQueryDefaultViewMatrix** function retrieves the current default viewing transformation.

### Parameters

*hps* Identifies the presentation space.

*cElements* Specifies the number of elements in the transformation to retrieve. It must be an integer in the range 0 through 9.

*pmatlf* Points to a **MATRIXLF** structure that receives the elements of the default viewing transformation matrix. The **MATRIXLF** structure has the following form:

```

typedef struct _MATRIXLF {
 FIXED fxM11;
 FIXED fxM12;
 LONG lM13;
 FIXED fxM21;
 FIXED fxM22;
 LONG lM23;
 LONG lM31;
 LONG lM32;
 LONG lM33;
} MATRIXLF;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**See Also** `GpiSetDefaultViewMatrix`

## ■ GpiQueryDefCharBox

---

`BOOL GpiQueryDefCharBox(hps, psizl)`

`HPS hps; /* presentation-space handle */`

`PSIZEL psizl; /* pointer to structure for character-box size */`

The `GpiQueryDefCharBox` function retrieves the size of the default graphics-character box in world coordinates.

**Parameters** *hps* Identifies the presentation space.

*psizl* Points to a `SIZEL` structure that receives the default character-box size. The `SIZEL` structure has the following form:

```
typedef struct _SIZEL {
 LONG cx;
 LONG cy;
} SIZEL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**See Also** `GpiQueryCharBox`

## ■ GpiQueryDevice

---

`HDC GpiQueryDevice(hps)`

`HPS hps; /* presentation-space handle */`

The `GpiQueryDevice` function retrieves a device-context handle for the currently associated presentation space.

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is a handle to a device context if the function is successful or `NULL` if no device context is currently associated with the presentation space. The return value is `HDC_ERROR` if an error occurred.

**Example** This example uses the `GpiQueryDevice` function to retrieve a device-context handle for the presentation space of the desktop window. The handle is used in the `DevQueryCaps` function to determine the width and height of the Presentation Manager screen.

```
HPS hps;
HDC hdc;
LONG lwidth, lheight;
```

```

hps = WinGetScreenPS (HWND_DESKTOP);
hdc = GpiQueryDevice (hps);
DevQueryCaps (hdc, CAPS_WIDTH, 1L, &lWidth);
DevQueryCaps (hdc, CAPS_HEIGHT, 1L, &lHeight);

```

See Also

DevOpenDC, GpiAssociate, WinQueryWindowDC

## ■ GpiQueryDeviceBitmapFormats

**BOOL GpiQueryDeviceBitmapFormats** (*hps*, *clData*, *alData*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *clData*; /\* number of elements \*/  
**PLONG** *alData*; /\* array of elements \*/

The **GpiQueryDeviceBitmapFormats** function retrieves the bitmap formats for the raster or memory device associated with the given presentation space. The function copies the formats to the array pointed to by the *alData* parameter. A bitmap format consists of two 32-bit values, the first specifying the number of color planes, and the second specifying the number of color bits per pel. The first format copied to the array is the format that most closely matches the device.

The **GpiQueryDeviceBitmapFormats** function uses the *clData* parameter to determine how many formats to return. Since each format fills two elements in the array, the *clData* must be a multiple of 2. Although there are several standard bitmap formats, most devices use just a few. If *clData* specifies more formats than the device supports, the functions fills the extra elements with zero. The **DevQueryCaps** function can be used to retrieve the actual number of bitmap formats for the device.

### Parameters

*hps* Identifies the presentation space.

*clData* Specifies the number of elements to copy to the array. Since each bitmap format fills two elements, this parameter must be a multiple of 2.

*alData* Points to the array to receive the bitmap formats.

### Return Value

The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

### Example

This example uses the **GpiQueryDeviceBitmapFormats** function to retrieve a bitmap format for the given device. The format is then used to create a bitmap that is compatible with the device.

```

LONG alFormats[2];
BITMAPINFOHEADER bmp;
HBITMAP hbm;

/* Retrieve the format that most closely matches the device. */
GpiQueryDeviceBitmapFormats(hps, 2L, alFormats);

/* Set the bitmap dimensions and format and create the bitmap. */

bmp.cbFlix = 12;
bmp.cx = 100;
bmp.cy = 100;
bmp.cPlanes = alFormats[0];
bmp.cBitCount = alFormats[1];
hbm = GpiCreateBitmap(hps, &bmp, 0L, NULL, NULL);

```

See Also

DevQueryCaps, GpiBitBlt, GpiCreateBitmap, GpiLoadBitmap



**■ GpiQueryDrawControl**

---

**LONG GpiQueryDrawControl**(*hps*, *flDraw*)**HPS** *hps*; /\* presentation-space handle \*/**LONG** *flDraw*; /\* drawing-control flag \*/

The **GpiQueryDrawControl** function checks the state of the drawing control specified by *flDraw*. The function returns **DCTL\_ON** or **DCTL\_OFF** to specify whether the control is on or off.

**Parameters**     *hps*   Identifies the presentation space.

*flDraw*   Specifies the drawing control to check. It can be one of the following values:

| Value                 | Meaning                                       |
|-----------------------|-----------------------------------------------|
| <b>DCTL_BOUNDARY</b>  | Accumulates boundary data.                    |
| <b>DCTL_CORRELATE</b> | Correlates output with pick aperture.         |
| <b>DCTL_DISPLAY</b>   | Allows drawing to occur on the output medium. |
| <b>DCTL_DYNAMIC</b>   | Draws dynamic segments.                       |
| <b>DCTL_ERASE</b>     | Erases before drawing.                        |

**Return Value**     The return value is **DCTL\_ON** or **DCTL\_OFF** if the function is successful, or a negative number if an error occurred.

**Errors**             Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

**PMERR\_INV\_DRAW\_CONTROL**  
**PMERR\_INV\_MICROPS\_FUNCTION**

**See Also**           **GpiDrawDynamics**, **GpiRemoveDynamics**, **GpiSetDrawControl**

**■ GpiQueryDrawingMode**

---

**LONG GpiQueryDrawingMode**(*hps*)**HPS** *hps*; /\* presentation-space handle \*/

The **GpiQueryDrawingMode** function returns the drawing mode, as set by **GpiSetDrawingMode**.

**Parameters**     *hps*   Identifies the presentation space.

**Return Value**     The return value is the current drawing mode if the function is successful, or **DM\_ERROR** if an error occurred.

**See Also**           **GpiSetDrawingMode**

## ■ GpiQueryEditMode

---

**LONG GpiQueryEditMode(*hps*)**

**HPS *hps*;** /\* presentation-space handle \*/

The **GpiQueryEditMode** function returns the current editing mode.

**Parameters**     *hps*   Identifies the presentation space.

**Return Value**     The return value is **SEGEM\_INSERT** or **SEGEM\_REPLACE** if the function is successful, or **SEGEM\_ERROR** if an error occurred.

**Errors**             Use the **WinGetLastError** function to retrieve the error value, which may be the following:

**PMERR\_INV\_MICROPS\_FUNCTION**

**See Also**            **GpiQueryElementPointer**, **GpiQueryElementType**, **GpiSetEditMode**

## ■ GpiQueryElement

---

**LONG GpiQueryElement(*hps*, *off*, *cbMax*, *pb*)**

**HPS *hps*;**            /\* presentation-space handle            \*/

**LONG *off*;**            /\* offset to first byte to copy        \*/

**LONG *cbMax*;**        /\* size of the buffer                    \*/

**PBYTE *pb*;**           /\* pointer to buffer for graphics-order data \*/

The **GpiQueryElement** function retrieves the graphics-order data for an element in the currently open segment. The element pointer must point to the element to retrieve. The function copies the data to the buffer pointed to by the *pb* parameter, copying all bytes in the current element or the number of bytes specified by *cbMax*, whichever is smaller. If *off* is not zero, the function uses this parameter as an offset from the beginning of the element to the first byte to copy to the buffer.

The function can be used only in an open segment and only when the drawing mode is **DM\_RETAIN**. It cannot be used in an element bracket.

**Parameters**     *hps*   Identifies the presentation space.

*off*   Specifies the offset from the beginning of the segment to the first byte of graphics-order data for the element.

*cbMax*   Specifies the size in bytes of the *pb* buffer.

*pb*   Points to a buffer that receives the graphics-order data for the element.

**Return Value**     The return value specifies the number of bytes returned if the function is successful or **GPI\_ALTEERROR** if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_INV_LENGTH
PMERR_INV_MICROPS_FUNCTION
PMERR_NO_CURRENT_ELEMENT
PMERR_NOT_IN_RETAIN_MODE
```

**Example** This example uses the `GpiQueryElement` function to retrieve the graphics-order data for an element.

```
BYTE abElement[512];
/* Move pointer to first element in segment. */

GpiSetElementPointer(hps, 1L);
GpiQueryElement(hps, /* presentation space */
 0L, /* start with first byte in element */
 512L, /* copy no more than 512 bytes */
 abElement); /* buffer to receive data */
```

**See Also** `GpiBeginElement`, `GpiDeleteElement`, `GpiEndElement`, `GpiQueryElementPointer`, `GpiSetElementPointer`

## ■ **GpiQueryElementPointer**

**LONG GpiQueryElementPointer(hps)**

**HPS hps;** /\* presentation-space handle \*/

The `GpiQueryElementPointer` function retrieves the current element pointer. The function can be used only in an open segment and only when the drawing mode is `DM_RETAIN`.

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is the current element pointer if the function is successful or `GPI_ALTError` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_INV_MICROPS_FUNCTION
PMERR_NOT_IN_RETAIN_MODE
```

**See Also** `GpiBeginElement`, `GpiDeleteElement`, `GpiEndElement`, `GpiQueryElement`, `GpiSetElementPointer`

## ■ GpiQueryElementType

**LONG GpiQueryElementType**(*hps, plType, cbDesc, pszDesc*)

**HPS** *hps*; /\* presentation-space handle \*/  
**PLONG** *plType*; /\* pointer to variable for type \*/  
**LONG** *cbDesc*; /\* length in bytes of buffer \*/  
**PSZ** *pszDesc*; /\* pointer to buffer for description \*/

The **GpiQueryElementType** function retrieves the type and description for an element in current open segment. It also returns the size in bytes of the element. The element pointer must point to the element to retrieve. The function copies the type to the variable pointed to by the *plType* parameter, and copies the description, up to the number of bytes specified by *cbDesc*, to the buffer pointed to by the *pszDesc* parameter.

The function can be used only in an open segment and only when the drawing mode is **DM\_RETAIN**. It cannot be used in an element bracket.

### Parameters

*hps* Identifies the presentation space.

*plType* Points to the variable that receives the element type.

*cbDesc* Specifies the maximum number of bytes of description to copy to the buffer specified by the *pszDesc* parameter.

*pszDesc* Points to the buffer that receives the description, a null-terminated string, for the element. The buffer must have the number of bytes specified by *cbDesc*. If the description is longer than the buffer, it is truncated.

### Return Value

The return value is the size of the data required to hold the element if the function is successful or **GPI\_ALTError** if an error occurred.

### Errors

Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

**PMERR\_INV\_LENGTH**  
**PMERR\_INV\_MICROPS\_FUNCTION**  
**PMERR\_NO\_CURRENT\_ELEMENT**  
**PMERR\_NOT\_IN\_RETAIN\_MODE**

### Example

This example uses the **GpiQueryElementType** function to retrieve the size of the current element. The size is used to retrieve the graphics-order data in the element.

```
BYTE abElement[512];
LONG cbElement;
LONG lType;

/* move pointer to first element in segment */

GpiSetElementPointer(hps, 1L);
cbElement = GpiQueryElementType(
 hps, /* presentation space */
 &lType, /* variable to receive type */
 OL, /* copy zero bytes of description */
 NULL); /* no buffer for description */

GpiQueryElement(hps, OL, cbElement, abElement);
```

### See Also

**GpiBeginElement**, **GpiElement**, **GpiQueryElement**, **GpiQueryElementPointer**

## ■ GpiQueryFontFileDescriptions

---

**LONG GpiQueryFontFileDescriptions** (*hab*, *pszFileName*, *pcFonts*, *pffdescs*)

**HAB** *hab*; /\* anchor-block handle \*/  
**PSZ** *pszFileName*; /\* pointer to the font-resource filename \*/  
**PLONG** *pcFonts*; /\* pointer to variable with number of fonts \*/  
**PFFDESCS** *pffdescs*; /\* array of names \*/

The **GpiQueryFontFileDescriptions** function retrieves the typeface family and names contained in the specified file if the file is a font-resource file. The function copies the names to the array pointed to by the *pffdescs* parameter. Each name is a null-terminated string up to 32 characters long. The function copies all names in the file up to the number of names specified by the *pcFonts* parameter.

### Parameters

*hab* Identifies the anchor block.

*pszFileName* Points to a null-terminated string. This string must be a valid MS OS/2 filename. If it does not specify a path and the *.fon* filename extension, the function appends the default extension (*.dll*) and looks for the font-resource file in the directories specified by the *libpath* command in the *config.sys* file.

*pcFonts* Points to a variable specifying the maximum number of typeface family and name pairs to retrieve. The function copies the actual number of descriptions it retrieved to this variable.

*pffdescs* Points to the array to receive the typeface family and names for each font. Each array element is itself a two-element array of type **FFDESCS**.

### Return Value

The return value is the number of fonts for which details were not returned if the function is successful, or **GPLALTERROR** if an error occurred.

### Example

This example uses the **GpiQueryFontFileDescriptions** to retrieve the typeface family and names for the fonts in the *helv.fon* file. The function is called twice, once to determine the actual number of fonts in the file, and again to retrieve the descriptions.

```
PFFDESCS pffdescs;
SEL sel;
LONG cFonts;

/* Retrieve a count of all fonts in the file. */
cFonts = GpiQueryFontFileDescriptions(hab, "helv", &cFonts, NULL);

/* Allocate space for the descriptions. */
DosAllocSeg((USHORT) (cFonts * sizeof(FFDESCS)), &sel, SEG_NONSHARED);
pffdescs = MAKEP(sel, 0);

/* Retrieve the descriptions. */
GpiQueryFontFileDescriptions(hab, "helv", &cFonts, pffdescs);
```

### See Also

**GpiQueryFonts**

## ■ GpiQueryFontMetrics

```

BOOL GpiQueryFontMetrics(hps, cbMetrics, pfm)
HPS hps; /* presentation-space handle */
LONG cbMetrics; /* length of the structure */
PFONTMETRICS pfm; /* pointer to structure for font metrics */

```

The **GpiQueryFontMetrics** function retrieves the font metrics for the currently selected logical font. The font metrics describe the typeface family, name, maximum height, average width, and other information about the font. All sizes the function retrieves are in world coordinates.

### Parameters

*hps* Identifies the presentation space.

*cbMetrics* Specifies the length of the font metrics.

*pfm* Points to a **FONTMETRICS** structure that receives font metrics for the logical font. This **FONTMETRICS** structure has the following form:

```

typedef struct _FONTMETRICS {
 CHAR szFamilyname[FACESIZE];
 CHAR szFacename[FACESIZE];
 SHORT idRegistry;
 SHORT usCodePage;
 LONG lEmHeight;
 LONG lXHeight;
 LONG lMaxAscender;
 LONG lMaxDescender;
 LONG lLowerCaseAscent;
 LONG lLowerCaseDescent;
 LONG lInternalLeading;
 LONG lExternalLeading;
 LONG lAveCharWidth;
 LONG lMaxCharInc;
 LONG lEmInc;
 LONG lMaxBaselineExt;
 SHORT sCharSlope;
 SHORT sInlineDir;
 SHORT sCharRot;
 USHORT usWeightClass;
 USHORT usWidthClass;
 SHORT sXDeviceRes;
 SHORT sYDeviceRes;
 SHORT sFirstChar;
 SHORT sLastChar;
 SHORT sDefaultChar;
 SHORT sBreakChar;
 SHORT sNominalPointSize;
 SHORT sMinimumPointSize;
 SHORT sMaximumPointSize;
 USHORT fsType;
 USHORT fsDefn;
 USHORT fsSelection;
 USHORT fsCapabilities;
 LONG lSubscriptXSize;
 LONG lSubscriptYSize;
 LONG lSubscriptXOffset;
 LONG lSubscriptYOffset;
 LONG lSuperscriptXSize;
 LONG lSuperscriptYSize;
 LONG lSuperscriptXOffset;
 LONG lSuperscriptYOffset;
 LONG lUnderscoreSize;
 LONG lUnderscorePosition;
 LONG lStrikeoutSize;
 LONG lStrikeoutPosition;
 SHORT sKerningPairs;
 SHORT sReserved;
 LONG lMatch;
} FONTMETRICS;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

- Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:  
**PMERR\_INV\_LENGTH**
- Example** This example uses the `GpiQueryFontMetrics` function to retrieve the font metrics for the current font.  

```
FONTMETRICS fm;
GpiQueryFontMetrics(hps, sizeof(FONTMETRICS), &fm);
```
- See Also** `GpiQueryCp`, `GpiQueryFonts`, `GpiQueryKerningPairs`

## ■ GpiQueryFonts

**LONG GpiQueryFonts** (*hps*, *fOptions*, *pszFacename*, *pcFonts*, *cbMetrics*, *pfm*)

**HPS** *hps*; /\* presentation-space handle \*/  
**ULONG** *fOptions*; /\* type of fonts to retrieve \*/  
**PSZ** *pszFacename*; /\* pointer to typeface name of the fonts \*/  
**PLONG** *pcFonts*; /\* number of fonts to retrieve \*/  
**LONG** *cbMetrics*; /\* length of the structure \*/  
**PFONTMETRICS** *pfm*; /\* pointer to structure(s) for font metrics \*/

The `GpiQueryFonts` function retrieves the font metrics for loaded fonts. The function copies the font metrics for the fonts that have the typeface name specified by the *pszFacename* parameter to the structure or array of structures pointed to by the *pfm* parameter. The function retrieves font metrics for all fonts up to the number specified by the *pcFonts* parameters. If the function does not retrieve font metrics for all the fonts, it returns the number of remaining fonts.

All sizes retrieved by the function are in world coordinates.

- Parameters**
- hps* Identifies the presentation space.
- fOptions* Specifies the type of fonts to retrieve. It can be a combination of the following values:

| Value      | Meaning                                                                                              |
|------------|------------------------------------------------------------------------------------------------------|
| QF_PUBLIC  | Retrieve public fonts. Public fonts were loaded by the system and are available to all applications. |
| QF_PRIVATE | Retrieve private fonts. Private fonts were loaded by the application and are available only to it.   |

*pszFacename* Points to the typeface name of the fonts. If the *pszFacename* parameter is a NULL pointer, the function retrieves metrics for all available fonts regardless of their typeface names.

*pcFonts* Points to a variable containing the number of fonts for which to retrieve metrics. The function copies to this variable the actual number of fonts it retrieved.

**cbMetrics** Specifies the length of one FONTMETRICS structure.

**pfm** Points to one or more FONTMETRICS structures that receive the metrics of the fonts. The number of structure required is specified by the *pcFonts* parameter. This FONTMETRICS structure has the following form:

```
typedef struct _FONTMETRICS {
 CHAR szFamilyname[FACESIZE];
 CHAR szFacename[FACESIZE];
 USHORT idRegistry;
 USHORT usCodePage;
 LONG lEmHeight;
 LONG lXHeight;
 LONG lMaxAscender;
 LONG lMaxDescender;
 LONG lLowerCaseAscent;
 LONG lLowerCaseDescent;
 LONG lInternalLeading;
 LONG lExternalLeading;
 LONG lAveCharWidth;
 LONG lMaxCharInc;
 LONG lEmInc;
 LONG lMaxBaselineExt;
 SHORT sCharSlope;
 SHORT sInlineDir;
 SHORT sCharRot;
 USHORT usWeightClass;
 USHORT usWidthClass;
 SHORT sXDeviceRes;
 SHORT sYDeviceRes;
 SHORT sFirstChar;
 SHORT sLastChar;
 SHORT sDefaultChar;
 SHORT sBreakChar;
 SHORT sNominalPointSize;
 SHORT sMinimumPointSize;
 SHORT sMaximumPointSize;
 USHORT fsType;
 USHORT fsDefn;
 USHORT fsSelection;
 USHORT fsCapabilities;
 LONG lSubscriptXSize;
 LONG lSubscriptYSize;
 LONG lSubscriptXOffset;
 LONG lSubscriptYOffset;
 LONG lSuperscriptXSize;
 LONG lSuperscriptYSize;
 LONG lSuperscriptXOffset;
 LONG lSuperscriptYOffset;
 LONG lUnderscoreSize;
 LONG lUnderscorePosition;
 LONG lStrikeoutSize;
 LONG lStrikeoutPosition;
 SHORT sKerningPairs;
 SHORT sReserved;
 LONG lMatch;
} FONTMETRICS;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

### Return Value

The return value is the number of fonts *not* returned if the function is successful, or `GPL_ALTError` if an error occurred.

### Example

This example uses the `GpiQueryFonts` function to retrieve the font metrics for all private fonts having the “Helv” typeface name. The function is called twice, once to determine the number of fonts available, again to retrieve the font metrics for all the fonts.



```

LONG cFonts, lTemp = 0L;
SEL sel;
PFONMETRICS pfm;

/* Determine the number of fonts. */
cFonts = GpiQueryFonts(hps, QF_PRIVATE, "Helv", &lTemp,
 (ULONG) sizeof(FONMETRICS), NULL);

/* Allocate space for the font metrics. */
DosAllocSeg((USHORT) (sizeof(FONMETRICS) * cFonts),
 &sel, SEG_NONSHARED);
pfm = MAKEP(sel, 0);

/* Retrieve the font metrics. */
cFonts = GpiQueryFonts(hps, QF_PRIVATE, "Helv", &cFonts,
 (ULONG) sizeof(FONMETRICS), pfm);

```

**See Also** GpiCreateLogFont, GpiQueryFontMetrics

## ■ GpiQueryGraphicsField

```

BOOL GpiQueryGraphicsField(hps, prclField)
HPS hps; /* presentation-space handle */
PRECTL prclField; /* pointer to structure for the graphics field */

```

The **GpiQueryGraphicsField** function retrieves the coordinates, in presentation-page units, of the graphics field. The function copies the coordinates of the lower-left and upper-right corners of the field to the structure pointed to by the *prclField* parameter.

**Parameters** *hps* Identifies the presentation space.

*prclField* Points to the **RECTL** structure that receives the graphics field. This **RECTL** structure has the following form:

```

typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is **GPL\_OK** if the function is successful or **GPL\_ERROR** if an error occurred.

**See Also** GpiQueryClipBox, GpiQueryClipRegion, GpiQueryGraphicsField, GpiQueryViewingLimits, GpiSetGraphicsField

## ■ GpiQueryInitialSegmentAttrs

**LONG GpiQueryInitialSegmentAttrs** (*hps, lAttribute*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *lAttribute*; /\* attribute \*/

The **GpiQueryDrawControl** function checks whether the initial segment attribute specified by *lAttribute* is enabled. The function returns **ATTR\_ON** or **ATTR\_OFF** to specify whether the given attribute is enabled or disabled.

### Parameters

*hps* Identifies the presentation space.

*lAttribute* Specifies the attributes to check. It can be one of the following values:

| Value                       | Meaning                 |
|-----------------------------|-------------------------|
| <b>ATTR_CHAINED</b>         | Chained                 |
| <b>ATTR_DETECTABLE</b>      | Detectability           |
| <b>ATTR_DYNAMIC</b>         | Dynamic                 |
| <b>ATTR_FASTCHAIN</b>       | Fast chaining           |
| <b>ATTR_PROP_DETECTABLE</b> | Propagate detectability |
| <b>ATTR_PROP_VISIBLE</b>    | Propagate visibility    |
| <b>ATTR_VISIBLE</b>         | Visibility              |

### Return Value

The return value is **ATTR\_ON** or **ATTR\_OFF** if the function is successful, or **ATTR\_ERROR** if an error occurred.

### Errors

Use the **WinGetLastError** function to retrieve the error value, which may be the following:

**PMERR\_INV\_MICROPS\_FUNCTION**

### See Also

**GpiSetInitialSegmentAttrs**

## ■ GpiQueryKerningPairs

**BOOL GpiQueryKerningPairs** (*hps, ckrnpr, akrnpr*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *ckrnpr*; /\* number of kerning pairs \*/

**PKERNINGPAIRS** *akrnpr*; /\* pointer to array of kerning-pair structures \*/

The **GpiQueryKerningPairs** function retrieves kerning-pair information for the current font for the presentation space. The kerning-pair information specifies the characters to be kerned and the amount of space in world coordinates to kern. The number of kerned pairs for a given font is specified in the font metrics for that font.

### Parameters

*hps* Identifies the presentation space.

*ckrnpr* Specifies the number of kerning pairs to retrieve.

*akrnpr* Points to the array of **KERNINGPAIRS** structures that receives the kerning-pair information. The array must have the number of elements specified by the *ckrnpr* parameter. The **KERNINGPAIRS** structure has the following form:

```
typedef struct _KERNINGPAIRS {
 SHORT sFirstChar;
 SHORT sSecondChar;
 SHORT sKerningAmount;
} KERNINGPAIRS;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is **GPL\_OK** if the function is successful or **GPL\_ERROR** if an error occurred.

**Example** This example uses the **GpiQueryKerningPairs** function to retrieve the kerning information for the current font.

```
FONTMETRICS fm;
SEL sel;
PKERNINGPAIRS akrnpr;

GpiQueryFontMetrics(hps, (LONG) sizeof(FONTMETRICS), &fm);
DosAllocSeg(fm.sKerningPairs * sizeof(KERNINGPAIRS), &sel,
 SEG_NONSHARED);
akrnpr = MAKEP(sel, 0);
GpiQueryKerningPairs(hps, (LONG) fm.sKerningPairs, akrnpr);
```

**See Also** **GpiQueryFontMetrics**

## ■ **GpiQueryLineEnd**

---

**LONG GpiQueryLineEnd(hps)**

**HPS hps;** /\* presentation-space handle \*/

The **GpiQueryLineEnd** function returns the current line-end attribute. The function cannot be used in an open segment when the drawing mode is **DM\_RETAIN**.

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is the current line-end attribute if the function is successful, or **LINEEND\_ERROR** if an error occurred.

**See Also** **GpiQueryAttrs**, **GpiSetDrawingMode**, **GpiSetLineEnd**

## ■ **GpiQueryLineJoin**

---

**LONG GpiQueryLineJoin(hps)**

**HPS hps;** /\* presentation-space handle \*/

The **GpiQueryLineJoin** function returns the current line-join attribute. The function cannot be used in an open segment when the drawing mode is **DM\_RETAIN**.

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is the current line-join attribute if the function is successful, or `LINEJOIN_ERROR` if an error occurred.

**See Also** `GpiQueryAttrs`, `GpiSetDrawingMode`, `GpiSetLineJoin`

## ■ GpiQueryLineType

---

**LONG GpiQueryLineType**(*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The `GpiQueryLineType` function returns the current cosmetic line-type attribute, as set by the `GpiSetLineType` function. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is the current cosmetic line-type attribute if the function is successful, or `LINETYPE_ERROR` if an error occurred.

**See Also** `GpiQueryAttrs`, `GpiSetDrawingMode`, `GpiSetLineType`

## ■ GpiQueryLineWidth

---

**FIXED GpiQueryLineWidth**(*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The `GpiQueryLineWidth` function returns the current value of the cosmetic line-width attribute, as set by the `GpiSetLineWidth` function. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is the current value of the cosmetic line-width attribute if the function is successful, or `LINEWIDTH_ERROR` if an error occurred.

**See Also** `GpiQueryAttrs`, `GpiSetDrawingMode`, `GpiSetLineWidth`

## ■ GpiQueryLineWidthGeom

---

**LONG GpiQueryLineWidthGeom**(*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The `GpiQueryLineWidthGeom` function returns the current geometric line-width attribute. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is the current geometric line-width attribute if the function is successful, or `LINEWIDTHGEOM_ERROR` if an error occurred.

**See Also** `GpiQueryAttrs`, `GpiSetDrawingMode`, `GpiSetLineWidthGeom`

## ■ GpiQueryLogColorTable

---

**LONG GpiQueryLogColorTable** (*hps, flOptions, iStart, clTable, alTable*)

**HPS** *hps*; /\* presentation-space handle \*/  
**ULONG** *flOptions*; /\* color type to retrieve \*/  
**LONG** *iStart*; /\* starting index \*/  
**LONG** *clTable*; /\* maximum number of values to copy \*/  
**PLONG** *alTable*; /\* pointer to array for elements \*/

The **GpiQueryLogColorTable** function retrieves the current logical-color-table entries. This function copies the RGB color values from the current logical color table to the array pointed to by *alTable*. It may also copy the color-table index values for each RGB color value, depending on the value of *flOptions*.

**GpiQueryLogColorTable** uses *iStart* as the index of the first color table entry to copy. It continues to copy subsequent entries until it reaches the end of the table or it has copied the number of values specified by *clTable*. If a logical color-table entry has no corresponding RGB color value, the function either copies - 1 to the array or skips the entry, depending on the value of *flOptions*. It skips the entry only if it is copying both the color-table index and the RGB value to the array.

### Parameters

*hps* Identifies the presentation space.

*flOptions* Specifies whether to retrieve indexes and RGB color values or just RGB values. If *flOptions* is LCOLOPT\_INDEX, the function retrieves indexes and RGB color values. If it is 0x0000, the function retrieves RGB color values only.

*iStart* Specifies the color-table index for the first entry to copy. The function copies this entry and all subsequent entries up to the number of values specified by *clTable* or the end of the logical color table.

*clTable* Specifies the maximum number of values to copy to the array pointed to by *alTable*. If *flOptions* is 0x0000, this parameter specifies the number of RGB color values to retrieve. If *flOptions* is LCOLOPT\_INDEX, this parameter specifies the combined total of index and RGB values to retrieve and must be an even value.

*alTable* Points to the array to receive the color-table entries. If *flOptions* is equal to 0x0000, each element in the array receives an RGB color value. If *flOptions* is LCOLOPT\_INDEX, each pair of array elements receives an index and a corresponding RGB value, with the first element in the pair receiving the index.

### Return Value

The return value is the number of values copied to the array if the function is successful. The return value is QLCT\_ERROR if an error occurred. The function returns QLCT\_RGB if *flOptions* is LCOLOPT\_INDEX and the current logical color table does not use indexes.

### Errors

Use the WinGetLastError function to retrieve the error value, which may be the following:

PMERR\_INV\_COLOR\_OPTIONS

**Example**

This example uses the `GpiQueryLogColorTable` function to retrieve all the entries in the current logical color table.

```

LONG cColors;
SEL sel;
PLONG alColor;

/* Find out how many colors are in the color table. */
DevQueryCaps(GpiQueryDevice(hps), CAPS_COLORS, 1L, &cColors);

/* Allocate space for the color values and indexes. */
DosAllocSeg((USHORT)(cColors * 2 * sizeof(LONG)), &sel, SEG_NONSHARED);
alColor = MAKEP(sel, 0);

/* Retrieve the values. */
GpiQueryLogColorTable(hps, /* presentation space */
 LCOLOPT_INDEX, /* retrieve indexes and RGB values */
 OL, /* start with first entry */
 cColors * 2, /* copy 2 values for each entry */
 alColor); /* array to receive values */

```

**See Also**

`GpiCreateLogColorTable`, `GpiQueryColorData`, `GpiQueryNearestColor`, `GpiQueryRealColors`, `GpiQueryRGBColor`

## ■ GpiQueryMarker

---

**LONG GpiQueryMarker**(*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The `GpiQueryMarker` function returns the current value of the marker-symbol attribute. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is the current marker symbol if the function is successful, or `MARKSYM_ERROR` if an error occurred.

**See Also** `GpiMarker`, `GpiPolyMarker`, `GpiQueryAttrs`, `GpiSetDrawingMode`

## ■ GpiQueryMarkerBox

---

**BOOL GpiQueryMarkerBox**(*hps*, *psizfxBox*)

**HPS** *hps*; /\* presentation-space handle \*/

**PSIZEF** *psizfxBox*; /\* pointer to structure for marker-box size \*/

The `GpiQueryMarkerBox` function retrieves the current value of the marker-box attribute, set by the `GpiSetMarkerBox` function. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

**Parameters** *hps* Identifies the presentation space.

*psizfxBox* Points to the `SIZEF` structure that receives the size of the marker box. The `SIZEF` structure has the following form:

```

typedef struct _SIZEF {
 FIXED cx;
 FIXED cy;
} SIZEF;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**See Also** `GpiQueryAttrs`, `GpiSetDrawingMode`, `GpiSetMarkerBox`

## ■ GpiQueryMarkerSet

---

**LONG GpiQueryMarkerSet**(*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The `GpiQueryMarkerSet` function returns the current value of the marker-set attribute, as set by the `GpiSetMarkerSet` function. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is the local identifier for the marker set if the function is successful, or `LCID_ERROR` if an error occurred.

**See Also** `GpiQueryAttrs`, `GpiSetDrawingMode`, `GpiSetMarkerSet`

## ■ GpiQueryMetaFileBits

---

**BOOL GpiQueryMetaFileBits**(*hmf*, *off*, *cbBuffer*, *pbBuffer*)

**HMF** *hmf*; /\* metafile handle \*/

**LONG** *off*; /\* offset to the first metafile byte to copy \*/

**LONG** *cbBuffer*; /\* length in bytes of buffer \*/

**PBYTE** *pbBuffer*; /\* pointer to buffer for metafile data \*/

The `GpiQueryMetaFileBits` function copies data from the metafile specified by *hmf* to the buffer pointed to by the *pbBuffer* parameter. The function copies the bytes of the metafile, up to the number of bytes specified by *cbBuffer*, starting at the byte whose offset from the beginning of the metafile is specified by the *off* parameter.

**Parameters** *hmf* Identifies the memory metafile.

*off* Specifies the offset in bytes from the beginning of the metafile to the first byte to copy.

*cbBuffer* Specifies the number of bytes of metafile data to copy.

*pbBuffer* Points to the buffer to receive the metafile data. It must have the number of bytes specified by the *cbBuffer* parameter.

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

`PMERR_INV_METAFILE_LENGTH`  
`PMERR_INV_METAFILE_OFFSET`

**Example**

This example uses the `GpiQueryMetaFileBits` function to retrieve the graphics-order data from the specified metafile. The `GpiQueryMetaFileLength` function returns the length of the metafile.

```
HMF hmf;
LONG cBytes;
SEL sel;
LONG off;

hmf = GpiLoadMetaFile(hps, "sample.met");

/* Allocate the buffer for the metafile data. */

DosAllocSeg(0, &sel, SEG_NONSHARED);
pbBuffer = MAKEP(sel, 0);

cBytes = GpiQueryMetaFileLength(hmf); /* get length of metafile */

/* Continue to retrieve data in 64K blocks. */

for (off = 0L; off < cBytes; off += 65536L)
 GpiQueryMetaFileBits(
 hps, /* presentation space */
 off, /* offset of next byte to retrieve */
 65536L, /* retrieve as much as possible */
 pbBuffer); /* buffer to receive metafile data */
```

**See Also**

`GpiQueryMetaFileLength`, `GpiSetMetaFileBits`

## ■ GpiQueryMetaFileLength

---

**LONG** `GpiQueryMetaFileLength`(*hmf*)

**HMF** *hmf*; /\* metafile handle \*/

The `GpiQueryMetaFileLength` function returns the total length, in bytes, of the metafile specified by *hmf*. The function is typically used to determine the number of bytes of data to retrieve using the `GpiQueryMetaFileBits` function.

**Parameters**     *hmf*   Identifies the metafile.

**Return Value**     The return value is the metafile length if the function is successful, or `GPI_ALTError` if an error occurred.

**See Also**         `GpiQueryMetaFileBits`, `GpiSetMetaFileBits`

## ■ GpiQueryMix

---

**LONG** `GpiQueryMix`(*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The `GpiQueryMix` function returns the current value of the (line) foreground color-mixing mode, as set by the `GpiSetMix` function. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

**Parameters**     *hps*   Identifies the presentation space.



**Return Value** The return value is the current foreground mix mode if the function is successful, or `FM_ERROR` if an error occurred.

**See Also** `GpiQueryAttrs`, `GpiQueryBackMix`, `GpiSetDrawingMode`, `GpiSetMix`

## ■ GpiQueryModelTransformMatrix

---

**BOOL** `GpiQueryModelTransformMatrix(hps, cElements, pmatlf)`

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *cElements*; /\* number of elements \*/  
**PMATRIXLF** *pmatlf*; /\* pointer to structure for transformation matrix \*/

The `GpiQueryModelTransformMatrix` function retrieves the matrix for the current model transformation. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

**Parameters** *hps* Identifies the presentation space.

*cElements* Specifies the number of elements of the matrix to retrieve. It must be an integer in the range 0 through 9.

*pmatlf* Points to the `MATRIXLF` structure the receives the model transformation matrix. The `MATRIXLF` structure has the following form:

```
typedef struct _MATRIXLF {
 FIXED fxM11;
 FIXED fxM12;
 LONG lM13;
 FIXED fxM21;
 FIXED fxM22;
 LONG lM23;
 LONG lM31;
 LONG lM32;
 LONG lM33;
} MATRIXLF;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**See Also** `GpiSetDrawingMode`, `GpiSetModelTransformMatrix`

## ■ GpiQueryNearestColor

---

**LONG** `GpiQueryNearestColor(hps, flOptions, lRgbColorIn)`

**HPS** *hps*; /\* presentation-space handle \*/  
**ULONG** *flClrType*; /\* color type \*/  
**LONG** *lRgbColorIn*; /\* RGB color value \*/

The `GpiQueryColorIndex` function returns the RGB color value from the physical palette that most closely matches the RGB color value specified by the `lRgbColorIn` parameter. The function uses the physical palette of the device associated with the given presentation space.

If the current logical color table is realizable but has not been realized, the `LCOLOPT_REALIZED` option maps the RGB color value as if the logical color table has been realized. Since realizing a logical color table affects the contents

of the physical palette, the RGB color value returned with this option may be different than the value returned without the option.

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b>   | <p><i>hps</i> Identifies the presentation space.</p> <p><i>fClrType</i> Specifies whether the RGB color value is based on realized colors. If it is <code>LCOLOPT_REALIZED</code>, the function returns a RGB color based on the colors in the physical palette after the current logical color table is realized. If the parameter is zero, the function returns a RGB color based on the colors in the current physical palette.</p> <p><i>lRgbColorIn</i> Specifies a RGB color value.</p> |
| <b>Return Value</b> | The return value is the nearest available color if the function is successful or <code>GPI_ALTEERROR</code> if an error occurred.                                                                                                                                                                                                                                                                                                                                                             |
| <b>Errors</b>       | Use the <code>WinGetLastError</code> function to retrieve the error value, which may be one of the following:                                                                                                                                                                                                                                                                                                                                                                                 |
|                     | <pre>PMERR_INV_COLOR_OPTIONS PMERR_INV_RGBCOLOR</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <b>See Also</b>     | <code>GpiCreateLogColorTable</code> , <code>GpiQueryColorData</code> , <code>GpiQueryRealColors</code> , <code>GpiQueryRGBColor</code>                                                                                                                                                                                                                                                                                                                                                        |

## ■ GpiQueryNumberSetIds

`LONG GpiQueryNumberSetIds(hps)`

`HPS hps; /* presentation-space handle */`

The `GpiQueryNumberSetIds` function returns the number of local identifiers currently in use that refer to fonts or bitmaps. The function is typically used before calling the `GpiQuerySetIds` function.

|                     |                                                                                                                                          |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b>   | <i>hps</i> Identifies the presentation space.                                                                                            |
| <b>Return Value</b> | The return value is the number of local identifiers in use if the function is successful or <code>GPI_ERROR</code> if an error occurred. |
| <b>See Also</b>     | <code>GpiQuerySetIds</code> , <code>GpiSetBitmapId</code> , <code>GpiSetCharSet</code>                                                   |

## ■ GpiQueryPageViewport

`BOOL GpiQueryPageViewport(hps, prclViewport)`

`HPS hps; /* presentation-space handle */`

`PRECTL prclViewport; /* pointer to structure for viewport */`

The `GpiQueryPageViewport` function retrieves the page viewport. The function cannot be used if no device context is associated with the presentation space.

|                   |                                                                                                                                                                                                                      |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b> | <p><i>hps</i> Identifies the presentation space.</p> <p><i>prclViewport</i> Points to the <code>RECTL</code> structure that receives the page viewport. The <code>RECTL</code> structure has the following form:</p> |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

**See Also** `GpiCreatePS`, `GpiSetPageViewport`

## ■ GpiQueryPattern

---

**LONG GpiQueryPattern**(*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The `GpiQueryPattern` function returns the current value of the shading-pattern symbol, as set by the `GpiSetPattern` function. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is the current shading-pattern symbol if the function is successful or `PATSYM_ERROR` if an error occurred.

**See Also** `GpiQueryPatternRefPoint`, `GpiQueryPatternSet`, `GpiSetDrawingMode`, `GpiSetPattern`

## ■ GpiQueryPatternRefPoint

---

**BOOL GpiQueryPatternRefPoint**(*hps*, *pptlRefPoint*)

*point*"

**HPS** *hps*; /\* presentation-space handle \*/

**PPOINTL** *pptlRefPoint*; /\* pointer to structure for pattern-reference \*/

The `GpiQueryPatternRefPoint` function retrieves the current pattern-reference point. The function cannot be used in an open segment when the drawing mode is `DM_RETAIN`.

**Parameters** *hps* Identifies the presentation space.

*pptlRefPoint* Points to the `POINTL` structure that receives the pattern-reference point. The `POINTL` structure has the following form:

```
typedef struct _POINTL {
 LONG x;
 LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

**See Also** `GpiQueryPattern`, `GpiQueryPatternSet`, `GpiSetDrawingMode`

## ■ GpiQueryPatternSet

---

**LONG GpiQueryPatternSet(*hps*)**

**HPS *hps*;** /\* presentation-space handle \*/

The **GpiQueryPatternSet** function returns the current value of the pattern-set identifier, as set by the **GpiSetPatternSet** function. The function cannot be used in an open segment when the drawing mode is **DM\_RETAIN**.

**Parameters**      *hps*    Identifies the presentation space.

**Return Value**    The return value is the current pattern-set identifier if the function is successful or **LCID\_ERROR** if an error occurred.

**See Also**        **GpiQueryPattern**, **GpiQueryPatternRefPoint**, **GpiSetDrawingMode**, **GpiSetPatternSet**

## ■ GpiQueryPel

---

**LONG GpiQueryPel(*hps*, *pptl*)**

**HPS *hps*;**        /\* presentation-space handle        \*/

**PPOINTL *pptl*;**   /\* pointer to structure with point to query \*/

The **GpiQueryPel** function returns the color of a pel at the specified point. The point, given in world coordinates, must be in any of the current clipping objects: clip path, viewing limits, graphics field, clip region, or visible region.

**Parameters**      *hps*    Identifies the presentation space.

*pptl*    Points to the **POINTL** structure that contains the world coordinates of the point. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
 LONG x;
 LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value**    The return value is the color index of the pel if the function is successful or **GPL\_ALTERROR** if an error occurred.

**See Also**        **GpiPtVisible**, **GpiSetPel**

## ■ GpiQueryPickAperturePosition

---

**BOOL GpiQueryPickAperturePosition**(*hps*, *pptl*)

**HPS** *hps*; /\* presentation-space handle \*/

**PPOINTL** *pptl*; /\* pointer to structure for center point \*/

The **GpiQueryPickAperturePosition** function retrieves the position of the center of the pick aperture.

**Parameters** *hps* Identifies the presentation space.

*pptl* Points to the **POINTL** structure that receives the coordinates, in presentation page units, of the center. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
 LONG x;
 LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**See Also** **GpiQueryPickApertureSize**, **GpiSetPickAperturePosition**

## ■ GpiQueryPickApertureSize

---

**BOOL GpiQueryPickApertureSize**(*hps*, *psizl*)

**HPS** *hps*; /\* presentation-space handle \*/

**PSIZEL** *psizl*; /\* pointer to structure for pick-aperture size \*/

The **GpiQueryPickApertureSize** function retrieves the width and height of the the pick aperture. The pick aperture is set using the **GpiSetPickApertureSize** function.

**Parameters** *hps* Identifies the presentation space.

*psizl* Points to a **SIZEL** structure that receives the pick-aperture size. The **SIZEL** structure has the following form:

```
typedef struct _SIZEL {
 LONG cx;
 LONG cy;
} SIZEL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**See Also** **GpiQueryPickAperturePosition**, **GpiSetPickApertureSize**

## ■ GpiQueryPS

```
ULONG GpiQueryPS(hps, psizlPage)
HPS hps; /* presentation-space handle */
PSIZEL psizlPage; /* pointer to structure for page size */
```

The **GpiQueryPS** function retrieves the page parameters and returns the presentation-space options for the presentation space. The page parameters specify the dimensions of the presentation page. The presentation-space options specify the page unit, storage format, and presentation type for the presentation space. These are the values set for the presentation space when it is created using the **GpiCreatePS** function.

**Parameters**     *hps*   Identifies the presentation space.  
                   *psizlPage*   Points to a **SIZEL** structure that receives the presentation-page size. The **SIZEL** structure has the following form:

```
typedef struct _SIZEL {
 LONG cx;
 LONG cy;
} SIZEL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value**     The return value is the presentation-space options if the function is successful or **GPI\_ERROR** if an error occurred.

**Comment**             The individual presentation-space options can be extracted from the return value by using the bitwise AND operator and the constants defined for the *flOptions* parameter of the **GpiCreatePS** function.

**See Also**             **GpiCreatePS**, **GpiQueryPageViewport**

## ■ GpiQueryRealColors

```
LONG GpiQueryRealColors(hps, flClrType, lStart, cclr, aclr)
HPS hps; /* presentation-space handle */
ULONG flClrType; /* color options */
LONG lStart; /* ordinal number of first color */
LONG cclr; /* number of colors */
PLONG aclr; /* pointer to array of colors */
```

The **GpiQueryRealColors** function retrieves the RGB values in the physical palette of the device associated with the presentation space. These colors represent the only device colors currently available to the application. The function copies the RGB color values, up to the number specified by the *cclr* parameter, to the array pointed to by the *aclr* parameter. The function uses the *lStart* parameter to determine which physical palette color to start copying. If this parameter is zero, the function copies from the start of the physical palette. The function returns the number of colors actually retrieved.

An application can change the contents of the physical palette by realizing a color table, for devices that permit realizable color tables. If the current logical color table is realizable but has not been realized, the **LCOLOPT\_REALIZED**

option retrieves the RGB color values as if the logical color table has been realized. Since realizing a logical color table affects the contents of the physical palette, the RGB color values retrieved with this option may be different from the values retrieved without the option.

The **GpiQueryRealColors** function can also map the colors from the physical palette to the color-index values in the current logical color table. If the **LCOLOPT\_INDEX** option is specified, the function copies a color-index and RGB-color pair to the array pointed to by *aclr*. Each value in the pair fills one element in the array, with the color index appearing first.

## Parameters

*hps* Identifies the presentation space.

*flClrType* Specifies whether the RGB color values are realized colors and whether color-index values are retrieved. It can be one of the following:

| Value            | Meaning                                                                                                                                                                       |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x0000           | Retrieves RGB colors based on the colors in the current physical palette.                                                                                                     |
| LCOLOPT_REALIZED | Retrieves the RGB colors corresponding to the realized logical color table. This option does not realize the table, but does return values as if the table had been realized. |
| LCOLOPT_INDEX    | Retrieves indexes and RGB color values.                                                                                                                                       |

The **LCOLOPT\_REALIZED** and **LCOLOPT\_INDEX** options can be combined.

*lStart* Specifies the ordinal number of the first color to copy.

*cclr* Specifies the number of RGB color values to retrieve. If the **LCOLOPT\_INDEX** option is specified, this parameter must specify the total number of color indexes and RGB colors to retrieve; this value must be a multiple of 2.

*aclr* Points to the array that receives the RGB color values. It must have the number of elements specified by the *cclr* parameter. If the **LCOLOPT\_INDEX** option is given, the first element in each pair of elements is the color index.

## Return Value

The return value is the number of elements returned if the function is successful or **GPI\_ALTEERROR** if an error occurred.

## Errors

Use the **WinGetLastError** function to retrieve the error value, which may be the following:

**PMERR\_INV\_COLOR\_OPTIONS**

## See Also

**GpiCreateLogColorTable**, **GpiQueryColorData**, **GpiQueryNearestColor**, **GpiQueryRGBColor**

## ■ GpiQueryRegionBox

```
LONG GpiQueryRegionBox(hps, hrgn, prcl)
HPS hps; /* presentation-space handle */
HRGN hrgn; /* region handle */
PRECTL prcl; /* pointer to structure for enclosing rectangle */
```

The **GpiQueryRegionBox** function retrieves the dimensions of the smallest rectangle that encloses the region identified by *hrgn*. If the region is empty, the function sets the left and right coordinates and top and bottom coordinates to equal values. The function cannot be used if no device context is associated with the presentation space.

**Parameters**     *hps*   Identifies the presentation space.

*hrgn*   Identifies the region.

*prcl*   Points to the **RECTL** structure that receives the coordinates of the enclosing rectangle. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value**     The return value is **RGN\_NULL**, **RGN\_RECT**, or **RGN\_COMPLEX** if the function is successful, or **RGN\_ERROR** if an error occurred.

**See Also**            **GpiQueryClipBox**

## ■ GpiQueryRegionRects

```
BOOL GpiQueryRegionRects(hps, hrgn, prclBound, prgnrc, arcl)
HPS hps; /* presentation-space handle */
HRGN hrgn; /* region handle */
PRECTL prclBound; /* pointer to structure for enclosing rectangle */
PRGNRECT prgnrc; /* pointer to structure controlling processing */
PRECTL arcl; /* pointer to array of rectangle structures */
```

The **GpiQueryRegionRects** function retrieves the rectangles that define the region identified by the *hrgn* parameter. The rectangles can be used to recreate the region, by using the **GpiCreateRegion** function. The function copies the coordinates of one or more of the defining rectangles to the array of structures pointed to by the *arcl* parameter. It uses the rectangle pointed to by the *prclBound* parameter to determine which rectangles to retrieve. Only rectangles within this rectangle are retrieved. If the *prclBound* parameter is **NULL**, the function retrieves all rectangles in the region.

The **GpiQueryRegionRects** function uses the fields of the **RGNRECT** structure pointed to by the *prgnrc* parameter to control how the defining rectangles are retrieved. Since a region may comprise several rectangles, the **RGNRECT** structure lets an application retrieve a few rectangles at a time. The structure specifies which rectangle to start with and how many to retrieve. The function copies



the actual number of rectangles retrieved to the structure. Also, a field in this structure specifies the direction through the region the function is to take as it retrieves rectangles.

The **GpiQueryRegionRects** function cannot be used if no device context is associated with the presentation space.

**Parameters** *hps* Identifies the presentation space.

*hrgn* Identifies the region.

*prclBound* Points to the **RECTL** structure that contains the enclosing rectangle. Only rectangles within this rectangle are retrieved. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

*prgnrc* Points to the **RGNRECT** structure that specifies how to retrieve the rectangles. The **RGNRECT** structure has the following form:

```
typedef struct _RGNRECT {
 USHORT ircStart;
 USHORT crc;
 USHORT crcReturned;
 USHORT usDirection;
} RGNRECT;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

*arcl* Points to the array of **RECTL** structures that receives the defining rectangles.

**Return Value** The return value is **GPL\_OK** if the function is successful or **GPL\_ERROR** if an error occurred.

**Errors** Use the **WinGetLastError** function to retrieve the error value, which may be the following:

**PMERR\_INV\_REGION\_CONTROL**

**See Also** **GpiCombineRegion**, **GpiCreateRegion**

## ■ **GpiQueryRGBColor**

**LONG GpiQueryRGBColor**(*hps*, *flClrType*, *IColorIndex*)

```
HPS hps; /* presentation-space handle */
ULONG flClrType; /* color type */
LONG IColorIndex; /* color-index value */
```

The **GpiQueryRGBColor** function returns the RGB color value in the physical palette that corresponds to the color-index value specified by the *IColorIndex* parameter.

If the current logical color table is realizable but has not been realized, the `LCOLOPT_REALIZED` option maps the color-index value as if the logical color table has been realized. Since realizing a logical color table affects the contents of the physical palette, the RGB color value returned with this option may be different from the value returned without the option.

If the current logical color table was created using the `LCOLF_RGB` option, the `IColorIndex` parameter is interpreted as an RGB color value. In this case, the function is identical to the `GpiQueryNearestColor` function.

### Parameters

*hps* Identifies the presentation space.

*flClrType* Specifies whether the RGB color value is based on realized colors. If it is `LCOLOPT_REALIZED`, the function returns an RGB color based on the colors in the physical palette after the current logical color table is realized. If the parameter is zero, the function returns an RGB color based on the colors in the current physical palette.

*IColorIndex* Specifies the color index. This may be any valid color-index value except `CLR_DEFAULT`.

### Return Value

The return value is the RGB color that is the closest match if the function is successful or `GPI_ALTEERROR` if an error occurred.

### Errors

Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

`PMERR_INV_COLOR_INDEX`  
`PMERR_INV_COLOR_OPTIONS`

### See Also

`GpiCreateLogColorTable`, `GpiQueryColorData`, `GpiQueryNearestColor`, `GpiQueryRealColors`, `GpiQueryRGBColor`, `GpiSetColor`

## ■ GpiQuerySegmentAttrs

**LONG** `GpiQuerySegmentAttrs` (*hps*, *idSegment*, *IColorIndex*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *idSegment*; /\* segment identifier \*/

**LONG** *IColorIndex*; /\* attribute \*/

The `GpiQuerySegmentAttrs` function checks a segment for the specified attribute. The function returns `ATTR_ON` or `ATTR_OFF` to specify whether the segment has or does not have the given attribute. The function can be used to check the attributes of any segment, including the currently open segment (if any).

### Parameters

*hps* Identifies the presentation space.

*idSegment* Specifies the segment identifier. It must be greater than zero.

*IColorIndex* Specifies the attribute to check. It must be one of the following values:

| Value                        | Meaning       |
|------------------------------|---------------|
| <code>ATTR_CHAINED</code>    | Chained       |
| <code>ATTR_DETECTABLE</code> | Detectability |
| <code>ATTR_DYNAMIC</code>    | Dynamic       |

| Value                | Meaning                 |
|----------------------|-------------------------|
| ATTR_FASTCHAIN       | Fast chaining           |
| ATTR_PROP_DETECTABLE | Propagate detectability |
| ATTR_PROP_VISIBLE    | Propagate visibility    |
| ATTR_VISIBLE         | Visibility              |

**Return Value** The return is ATTR\_ON or ATTR\_OFF if the function is successful, or ATTR\_ERROR if an error occurred.

**Errors** Use the WinGetLastError function to retrieve the error value, which may be one of the following:

PMERR\_INV\_MICROPS\_FUNCTION  
PMERR\_INV\_SEG\_NAME

**See Also** GpiSetSegmentAttrs

## ■ GpiQuerySegmentNames

**LONG GpiQuerySegmentNames** (*hps, idFirstSegment, idLastSegment, cidMax, aidSegments*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *idFirstSegment*; /\* first segment \*/  
**LONG** *idLastSegment*; /\* last segment \*/  
**LONG** *cidMax*; /\* maximum number of segments \*/  
**PLONG** *aidSegments*; /\* pointer to array for segments \*/

The **GpiQuerySegmentNames** function retrieves the identifiers of all existing segments whose identifiers are in the range specified by the *idFirstSegment* and *idLastSegment* parameters.

If the *idFirstSegment* parameter is equal to or greater than the *idLastSegment* parameter, the function only checks for the segment having the identifier specified by *idFirstSegment*.

**Parameters**

- hps* Identifies the presentation space.
- idFirstSegment* Specifies the first segment in the range; it must be greater than zero.
- idLastSegment* Specifies the last segment; it must be greater than zero.
- cidMax* Specifies the maximum number of segment identifiers to retrieve.
- aidSegments* Points to the array to receive the segment identifiers. It must have the number of elements specified by the *cidMax* parameter.

**Return Value** The return value is the number of segment identifiers returned if the function is successful or GPL\_ALTEERROR if an error occurred.

**Errors** Use the WinGetLastError function to retrieve the error value, which may be one of the following:

PMERR\_INV\_MICROPS\_FUNCTION  
PMERR\_INV\_SEG\_NAME

**See Also** GpiOpenSegment

## ■ GpiQuerySegmentPriority

```
LONG GpiQuerySegmentPriority(hps, idRefSegment, cmdOrder)
HPS hps; /* presentation-space handle */
LONG idRefSegment; /* reference-segment identifier */
LONG cmdOrder; /* segment order */
```

The **GpiQuerySegmentPriority** function returns the identifier of the segment having the next-highest or next-lowest priority, relative to the segment specified by *idRefSegment*. The priority of a segment affects how segments in the picture chain are drawn.

The function uses the *cmdOrder* parameter to determine whether to look for the segment with next-highest or next-lowest priority. The function returns zero if there is no segment with next-highest or next-lowest priority.

### Parameters

*hps* Identifies the presentation space.

*idRefSegment* Specifies the identifier of the segment whose priority is compared, or is zero to specify the segment with lowest or highest priority.

*cmdOrder* Specifies whether to check for a segment with higher or lower priority. It can be one of the following values:

| Value      | Meaning                                                                                                                                                                        |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LOWER_PRI  | Return the identifier for the segment with next-lowest priority. If <i>idRefSegment</i> is zero, the function returns the identifier of the segment with the lowest priority.  |
| HIGHER_PRI | Return the identifier of the segment with next-highest priority. If <i>idRefSegment</i> is zero, the function returns the identifier of the segment with the highest priority. |

### Return Value

The return value is the identifier of the segment with the next-highest or next-lowest priority if the function is successful. The return value is **GPI\_ALTError** if an error occurred.

### Errors

Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

```
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_SEG_NAME
```

### See Also

**GpiDrawChain**, **GpiDrawFrom**, **GpiSetSegmentPriority**

## ■ GpiQuerySegmentTransformMatrix

---

**BOOL GpiQuerySegmentTransformMatrix(*hps, idSegment, cElements, pmatlf*)**

**HPS *hps*;** /\* presentation-space handle \*/  
**LONG *idSegment*;** /\* segment identifier \*/  
**LONG *cElements*;** /\* number of elements \*/  
**PMATRIXLF *pmatlf*;** /\* pointer to structure for matrix elements \*/

The **GpiQuerySegmentTransformMatrix** function retrieves one or more elements of the segment-transformation matrix for the segment specified by *idSegment*.

### Parameters

*hps* Identifies the presentation space.

*idSegment* Specifies the segment identifier.

*cElements* Specifies the number of elements to retrieve. It must be an integer value in the range 0 through 9.

*pmatlf* Points to the **MATRIXLF** structure that receives the transformation matrix. The **MATRIXLF** structure has the following form:

```
typedef struct _MATRIXLF {
 FIXED fxM11;
 FIXED fxM12;
 LONG lM13;
 FIXED fxM21;
 FIXED fxM22;
 LONG lM23;
 LONG lM31;
 LONG lM32;
 LONG lM33;
} MATRIXLF;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

### Return Value

The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

### Errors

Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

**PMERR\_INV\_MICROPS\_FUNCTION**  
**PMERR\_INV\_SEG\_NAME**

### See Also

**GpiSetSegmentTransformMatrix**

## ■ GpiQuerySetIds

---

**BOOL GpiQuerySetIds(*hps, cSets, alTypes, pstr8, alcid*)**

**HPS *hps*;** /\* presentation-space handle \*/  
**LONG *cSets*;** /\* number of objects to query \*/  
**PLONG *alTypes*;** /\* pointer to array of types \*/  
**PSTR8 *pstr8*;** /\* pointer to array for names \*/  
**PLONG *alcid*;** /\* pointer to array for local identifiers \*/

The **GpiQuerySetIds** function retrieves a list of types, names, and local identifiers for all current logical fonts and tagged bitmaps. The function copies the information to the arrays pointed to by the *alTypes*, *pstr8*, and *alcid* parameters.

The type specifies whether the object is a logical font or tagged bitmap. The name is an 8-character string that uniquely identifies the object. Not all objects have names.

The **GpiQuerySetIds** function retrieves information for only the number of objects specified by the *cSets* parameter, starting with the object having local identifier 1. If information for all objects is needed, the **GpiQueryNumberSetIds** function returns a count of all local identifiers in use.

## Parameters

*hps* Identifies the presentation space.

*cSets* Specifies the number of objects to retrieve. It must not be greater than the number of local identifiers currently in use.

*alTypes* Points to the array to receive the type for each object. The function sets each element in this array to either `LCIDT_FONT` or `LCIDT_BITMAP`. The array must have the number of elements specified by *cSets*.

*pstr8* Points to the array to receive the name for each object. Each element, itself an array, receives an object name of up to eight characters. If an object has no name, the element is set to zero. The array must have the number of elements specified by *cSet*.

*alcid* Points to the array that receives the local identifiers. The array must have the number of elements specified by *cSets*.

## Return Value

The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

## Example

This example uses the **GpiQuerySetIds** function to retrieve the local identifier for all logical fonts. It then uses the identifiers to delete the logical fonts.

```
LONG cIds;
SEL sel;
PLONG plType;
PSTR8 pstr8;
PLONG plcid;

cIds = GpiQueryNumberSetIds(hps); /* get number of local identifiers */
/* Allocate space for type, name, and local-identifier arrays. */

DosAllocSeg((USHORT) cIds * sizeof(LONG), &sel, SEG_NONSHARED);
plType = MAKEP(sel, 0);
DosAllocSeg((USHORT) cIds * sizeof(STR8), &sel, SEG_NONSHARED);
pstr8 = MAKEP(sel, 0);
DosAllocSeg((USHORT) cIds * sizeof(LONG), &sel, SEG_NONSHARED);
plcid = MAKEP(sel, 0);

/* Get the types, names, and local identifiers. */
GpiQuerySetIds(hps, cIds, plType, pstr8, plcid);

/* Delete each local identifier that has LCIDT_FONT type. */
for (i = 1; i < cIds; i++)
 if (plTypes[i] == LCIDT_FONT)
 GpiDeleteSetId(hps, plcid[i]);
```

## See Also

**GpiCreateLogFont**, **GpiQueryNumberSetIds**, **GpiSetBitmapId**

**■ GpiQueryStopDraw**

---

**LONG GpiQueryStopDraw**(*hps*)**HPS** *hps*; /\* presentation-space handle \*/

The **GpiQueryStopDraw** function returns the state of the stop-draw condition.

**Parameters**     *hps*   Identifies the presentation space.

**Return Value**   The return value is **TRUE** if the stop-draw condition is on or **FALSE** if it is not. The return value is **GPI\_ALTErrOR** if an error occurred.

**Errors**         Use the **WinGetLastError** function to retrieve the error value, which may be the following:

**PMERR\_INV\_MICROPS\_FUNCTION**

**See Also**        **GpiSetStopDraw**

**■ GpiQueryTag**

---

**BOOL GpiQueryTag**(*hps*, *pITag*)**HPS** *hps*;         /\* presentation-space handle \*/**PLONG** *pITag*;    /\* tag identifier         \*/

The **GpiQueryTag** function retrieves the current value of the tag identifier, as set by the **GpiSetTag** function. The function cannot be used in an open segment when the drawing mode is **DM\_RETAIN**.

**Parameters**     *hps*   Identifies the presentation space.  
                  *pITag*   Points to the variable to receive the tag.

**Return Value**   The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**Errors**         Use the **WinGetLastError** function to retrieve the error value, which may be the following:

**PMERR\_INV\_MICROPS\_FUNCTION**

**See Also**        **GpiSetDrawingMode**, **GpiSetTag**

## ■ GpiQueryTextBox

```

BOOL GpiQueryTextBox(hps, cchString, pchString, cptl, aptl)
HPS hps; /* presentation-space handle */
LONG cchString; /* number of characters */
PCH pchString; /* pointer to string */
LONG cptl; /* number of points */
PPOINTL aptl; /* pointer to array of point structures */

```

The **GpiQueryTextBox** function retrieves the text box and concatenation point for the string pointed to by the *pchString* parameter. The text box is four points specifying the parallelogram that, if drawn, encloses the given string when the string is displayed on the device. The concatenation point is the point the current position advances to after the string is drawn. All coordinates are relative to the start point of the string—that is, the text box and concatenation point are given as if the string were drawn at the world-space origin.

The **GpiQueryTextBox** function computes the text box and concatenation point using the current character attributes. It then copies the computed points to the array pointed to by the *aptl* parameter. In most cases, the function copies the upper-left, lower-left, upper-right, and lower-right corners of the text box first, followed by the concatenation point, but not all points need to be copied at all times. The function uses the *cptl* parameter to determine how many of these points to retrieve and copies only that number.

Depending on the character attributes, the “upper-left” corner of the text box may not seem so when the text box is actually drawn. For this reason, the function copies the coordinates of the text box to the array prior to applying character attributes, such as base-line angle, that affect the orientation of the points.

The function cannot be used in an open segment when the drawing mode is **DM\_RETAIN**.

### Parameters

*hps* Identifies the presentation space.

*cchString* Specifies the number of characters in the string pointed to by *pchString*.

*pchString* Points to the character string.

*cptl* Specifies the number of points to retrieve. If it is **TXTBOX\_COUNT**, the function retrieves the maximum number of points for the text box.

*aptl* Points to the array of **POINTL** structures that receives a list of points. The list of points contains the relative coordinates of the character box. The elements of the array are defined as follows:

| Value                     | Meaning             |
|---------------------------|---------------------|
| <b>TXTBOX_TOPLEFT</b>     | Upper-left corner   |
| <b>TXTBOX_BOTTOMLEFT</b>  | Lower-left corner   |
| <b>TXTBOX_TOPRIGHT</b>    | Upper-right corner  |
| <b>TXTBOX_BOTTOMRIGHT</b> | Lower-right corner  |
| <b>TXTBOX_CONCAT</b>      | Concatenation point |



The **POINTL** structure has the following form:

```
typedef struct _POINTL {
 LONG x;
 LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is **GPL\_OK** if the function is successful or **GPL\_ERROR** if an error occurred.

**Example** This example uses the **GpiQueryTextBox** function to draw a line under the string. The **GpiCharString** function draws the string at the point (100,100). Since the points retrieved by **GpiQueryTextBox** are relative to the start of the string, the starting point needs to be added to the points that are used to draw the underline.

```
POINTL aptl[TEXTBOX_COUNT];
POINTL ptl = { 100, 100 };

GpiQueryTextBox(hps, 11L, "This string", TEXTBOX_COUNT, aptl);
aptl[1].x += ptl.x;
aptl[1].y += ptl.y;
GpiMove(hps, &aptl[1]);
aptl[3].x += ptl.x;
aptl[3].y += ptl.y;
GpiLine(hps, &aptl[3]);
GpiMove(hps, &ptl);
GpiCharString(hps, 11L, "This string");
```

**See Also** **GpiCharStringAt**, **GpiSetDrawingMode**

## ■ **GpiQueryViewingLimits**

**BOOL GpiQueryViewingLimits** (*hps*, *prclLimits*)

**HPS** *hps*; /\* presentation-space handle \*/  
**PRECTL** *prclLimits*; /\* pointer to structure for viewing limits \*/

The **GpiQueryViewingLimits** function retrieves the current value of the viewing limits, as set by the **GpiSetViewingLimits** function. The function cannot be used in an open segment when the drawing mode is **DM\_RETAIN**.

**Parameters** *hps* Identifies the presentation space.  
*prclLimits* Points to the **RECTL** structure that receives the viewing limits. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is **GPL\_OK** if the function is successful or **GPL\_ERROR** if an error occurred.

**See Also** **GpiQueryClipBox**, **GpiQueryClipRegion**, **GpiQueryGraphicsField**, **GpiQueryViewingLimits**, **GpiSetDrawingMode**, **GpiSetViewingLimits**

## ■ GpiQueryViewingTransformMatrix

---

```

BOOL GpiQueryViewingTransformMatrix(hps, cElements, pmatlf)
HPS hps; /* presentation-space handle */
LONG cElements; /* number of elements */
PMATRIXLF pmatlf; /* pointer to structure for transformation matrix */

```

The **GpiQueryViewingTransformMatrix** function retrieves the current viewing-transformation matrix.

### Parameters

***hps*** Identifies the presentation space.

***cElements*** Specifies the number of elements to retrieve. It must be an integer in the range 0 through 9.

***pmatlf*** Points to the **MATRIXLF** structure that receives the elements of the viewing-transformation matrix. The **MATRIXLF** structure has the following form:

```

typedef struct _MATRIXLF {
 FIXED fxM11;
 FIXED fxM12;
 LONG lM13;
 FIXED fxM21;
 FIXED fxM22;
 LONG lM23;
 LONG lM31;
 LONG lM32;
 LONG lM33;
} MATRIXLF;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

### Return Value

The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

### Errors

Use the **WinGetLastError** function to retrieve the error value, which may be the following:

**PMERR\_INV\_MICROPS\_FUNCTION**

### See Also

**GpiSetViewingTransformMatrix**

## ■ GpiQueryWidthTable

---

```

BOOL GpiQueryWidthTable(hps, lFirstChar, clWidths, alWidths)
HPS hps; /* presentation-space handle */
LONG lFirstChar; /* code point of first character */
LONG clWidths; /* number of elements */
PLONG alWidths; /* pointer to array for width table */

```

The **GpiQueryWidthTable** function retrieves the widths of one or more characters in the current font. A character width is the distance (in world coordinates) the system advances along the baseline after drawing the character. The function copies the widths, starting with the width for the character specified by the ***lFirstChar*** parameter, to the array pointed to by the ***alWidths*** parameter. The function uses the ***clWidths*** parameter to determine how many widths to retrieve.

If the widths for all characters in the font are desired, the **GpiQueryFontMetrics** function can be used to retrieve the number of characters in the font.

|                     |                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b>   | <p><i>hps</i> Identifies the presentation space.</p> <p><i>lFirstChar</i> Specifies the code point of the first character for which a width is retrieved.</p> <p><i>clWidths</i> Specifies the number of widths to retrieve.</p> <p><i>alWidths</i> Points to the array that receives the character widths. The array must have the number of elements specified by <i>clWidths</i>.</p> |
| <b>Return Value</b> | The return value is <code>GPL_OK</code> if the function is successful or <code>GPL_ERROR</code> if an error occurred.                                                                                                                                                                                                                                                                    |
| <b>Errors</b>       | Use the <code>WinGetLastError</code> function to retrieve the error value, which may be the following:<br><br><code>PMERR_INV_FIRST_CHAR</code>                                                                                                                                                                                                                                          |
| <b>See Also</b>     | <code>GpiQueryFontMetrics</code>                                                                                                                                                                                                                                                                                                                                                         |

## ■ **GpiRealizeColorTable**

---

**BOOL GpiRealizeColorTable** (*hps*)

HPS *hps*; /\* presentation-space handle \*/

The `GpiRealizeColorTable` function realizes the logical color table. The function realizes the color table by replacing the colors in the physical palette. It replaces the physical palette colors with the device colors that most closely match the RGB color values given in the logical color table.

To realize a logical color table, the application must create the table using the `LCOL_REALIZABLE` option of the `GpiCreateLogColorTable` function and the device must be capable of realizing logical color tables. The `DevQueryCaps` function and `CAPS_COLOR` option can be used to determine if logical color tables can be realized.

If the presentation space is currently associated with a screen window device, this function should be used only when the associated window is maximized. Changing the physical palette colors for the screen affects output for all visible windows.

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b>   | <i>hps</i> Identifies the presentation space.                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Return Value</b> | The return value is <code>GPL_OK</code> if the function is successful or <code>GPL_ERROR</code> if an error occurred.                                                                                                                                                                                                                                                                                                                                   |
| <b>Example</b>      | <p>This example uses the <code>GpiRealizeColorTable</code> function to realize the logical color table. The <code>GpiUnrealizeColorTable</code> function is used to restore the physical palette after the drawing is complete.</p> <pre> RGB argb[16];          /* RGB color values for new logical color table */ /* Create a realizable logical color table. */ GpiCreateLogColorTable(hps, LCOL_REALIZABLE, LCOLF_CONSECRGB, OL, 16L, argb); </pre> |

```
GpiRealizeColorTable(hps); /* realizes the logical color table */
:
GpiUnrealizeColorTable(hps); /* restores the physical palette */
```

**See Also**      `DevQueryCaps`, `GpiUnrealizeColorTable`

## ■ GpiRectInRegion

**LONG GpiRectInRegion**(*hps*, *hrgn*, *prcl*)

**HPS** *hps*;      /\* presentation-space handle \*/

**HRGN** *hrgn*;    /\* region handle \*/

**PRECTL** *prcl*; /\* pointer to rectangle structure \*/

The `GpiRectInRegion` function checks whether any part of a rectangle lies within the specified region. The function cannot be used if no device context is associated with the presentation space.

**Parameters**      *hps*    Identifies the presentation space.  
                       *hrgn*    Identifies the region.  
                       *prcl*    Points to the **RECTL** structure that contains the rectangle to check. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value**      The return value is **RRGN\_OUTSIDE**, **RRGN\_PARTIAL**, or **RRGN\_INSIDE** if the function is successful, or **RRGN\_ERROR** if an error occurred.

**See Also**          `GpiPtInRegion`

## ■ GpiRectVisible

**LONG GpiRectVisible**(*hps*, *prcl*)

**HPS** *hps*;      /\* presentation-space handle \*/

**PRECTL** *prcl*; /\* pointer to rectangle structure \*/

The `GpiRectVisible` function checks whether any part of a rectangle is visible on the device associated with the specified presentation space. A point in the rectangle is visible if it lies within the intersection of the current graphics field, viewing limit, clip path, clip region, and visible region (if any).

**Parameters**      *hps*    Identifies the presentation space.  
                       *prcl*    Points to a **RECTL** structure that contains the rectangle in world coordinates. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is `RVIS_INVISIBLE`, `RVIS_PARTIAL`, or `RVIS_VISIBLE` if the function is successful, or `RVIS_ERROR` if an error occurred.

**See Also** GpiPtVisible

## ■ GpiRemoveDynamics

**BOOL GpiRemoveDynamics** (*hps, idFirstSegment, idLastSegment*)

```
HPS hps; /* presentation-space handle */
LONG idFirstSegment; /* first segment identifier */
LONG idLastSegment; /* last segment identifier */
```

The **GpiRemoveDynamics** function removes from the display any images drawn using dynamic segments. The function removes the image for any dynamic segment that is in the current picture chain and whose segment identifier is in the range specified by the *idFirstSegment* and *idLastSegment* parameters. The function removes only the image drawn using the dynamic segment and by the dynamic segment’s segment calls.

The **GpiRemoveDynamics** function checks for the segments specified by *idFirstSegment* and *idLastSegment*. If they do not exist, or are not in the chain, or *idLastSegment* is less than *idFirstSegment*, the function returns without removing segments and without an error value.

**Parameters**

- hps* Identifies the presentation space.
- idFirstSegment* Specifies the name of the first segment in the section. It must be greater than zero.
- idLastSegment* Specifies the last segment in the section. It must be greater than zero.

**Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_AREA_INCOMPLETE
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_SEG_NAME
PMERR_PATH_INCOMPLETE
```

**Comments** An application should remove dynamic segments before associating a new device context with the presentation space.

**Example**

This example uses the **GpiRemoveDynamics** function to remove the image drawn by the dynamic segment whose segment identifier is 4. It then edits the segment and redraws it, using the **GpiDrawDynamics** function.

```
POINTL pt1 = {30, 40};

/* Remove the image for dynamic segment #4. */
GpiRemoveDynamics(hps, 4L, 4L);

/* Edit the segment. */
GpiSetDrawingMode(DM_RETAIN);
GpiOpenSegment(hps, 4L);
GpiSetElementPointer(hps, 1L);
GpiMove(hps, &pt1);
GpiCloseSegment(hps);

GpiDrawDynamics(hps); /* redraws the edited segment */
```

**See Also**

**GpiCloseSegment**, **GpiDrawDynamics**

**■ GpiResetBoundaryData**

---

**BOOL GpiResetBoundaryData**(hps)

HPS hps; /\* presentation-space handle \*/

The **GpiResetBoundaryData** function resets the boundary data, copying the maximum boundary value to the lower corner and the minimum boundary value to the upper corner of the boundary-data rectangle.

The function is only necessary when accumulating boundary data in **DM\_DRAW** drawing mode. For other drawing modes, drawing functions automatically reset the boundary data. (However, the **GpiOpenSegment** function does not reset the boundary data.)

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**Errors** Use the **WinGetLastError** function to retrieve the error value, which may be the following:

PMERR\_INV\_MICROPS\_FUNCTION

**See Also** **GpiSetDrawControl**

**■ GpiResetPS**

---

**BOOL GpiResetPS**(hps, flOption)

HPS hps; /\* presentation-space handle \*/

ULONG flOption; /\* reset option \*/

The **GpiResetPS** function resets the presentation space. In general, resetting the presentation space restores attributes to their default values—that is, the values given to the attributes when the presentation space was created. The function can reset the presentation space in three ways: as if a segment were closed; as if the presentation space had just been created, but without deleting any resources;

and as if the presentation space had just been created. It uses the *fOption* parameter to determine how to reset the presentation space.

The **GpiResetPS** function does not draw or erase the device. It is up to the application to erase the screen, if this is required. Also, the function does not affect the association between the specified presentation space and a device context.

**Parameters** *hps* Identifies the presentation space.

*fOption* Specifies the reset option. It can be one of the following:

| Value         | Meaning                                                                                                                                                                                                                                                                                                                                                 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GRES_ATTRS    | Sets all current attributes to their default values, the current model transform to unity, and the current position to (0,0). The option also ends any open path, area, or element brackets and closes any open segment. Finally, it sets the current clip path and viewing limits to their widest possible values.                                     |
| GRES_SEGMENTS | Resets as described for GRES_ATTRS, plus it deletes all retained segments, clears any boundary data, releases the clip region (if any), enables kerning (if the device supports it), and sets the default values for initial segment attributes, default viewing transform, graphics field, drawing mode, draw controls, edit mode, and attribute mode. |
| GRES_ALL      | Resets as described for GRES_ATTRS and GRES_SEGMENTS, plus it deletes any logical fonts and local identifiers for bitmaps and sets the logical color table to its default value.                                                                                                                                                                        |

**Return Value** The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**Errors** Use the **WinGetLastError** function to retrieve the error value, which may be the following:

**PMERR\_INV\_RESET\_OPTIONS**

**See Also** **GpiAssociate**, **GpiCreatePS**

## ■ GpiRestorePS

**BOOL** GpiRestorePS(*hps*, *idPS*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *idPS*; /\* identifier for the presentation space \*/

The **GpiRestorePS** function restores the state of the presentation space by popping the state from the presentation space (PS) stack. The function sets the attributes and resources of the presentation space to the values that were saved previously by using the **GpiSavePS** function.

The PS stack, maintained internally by the system, can contain one or more saved presentation spaces. Each saved presentation space has a unique identifier. The **GpiRestorePS** function restores a specific presentation space if the *idPS*

parameter contains the corresponding identifier. The function also accepts negative identifiers. In this case, the function uses the absolute value of the identifier to determine how many presentation spaces to pop from the PS stack. For example, if it is -2, the function pops two presentation spaces from the stack. In either case, identifier or negative number, the function discards any presentation spaces that are skipped over on the PS stack.

|                     |                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b>   | <i>hps</i> Identifies the presentation space.<br><i>idPS</i> Specifies the identifier of the saved presentation space to restore, or a negative number indicating the number of saved presentation spaces to pop. If it is an identifier, it must have been returned previously by the <b>GpiSavePS</b> function. It must not be zero.                                                             |
| <b>Return Value</b> | The return value is <b>GPI_OK</b> if the function is successful or <b>GPI_ERROR</b> if an error occurred.                                                                                                                                                                                                                                                                                          |
| <b>Errors</b>       | Use the <b>WinGetLastError</b> function to retrieve the error value, which may be the following:<br><br>PMERR_INV_ID                                                                                                                                                                                                                                                                               |
| <b>Comments</b>     | The function can be used in an open segment only if the drawing mode is <b>DM_DRAW</b> . The function can also be used in an element, area, or path bracket. If it is in an area or path bracket, the corresponding <b>GpiSavePS</b> function must have been called in the same bracket.<br><br>If an error occurs, the function leaves the PS stack and the current presentation space unchanged. |
| <b>See Also</b>     | <b>GpiPop</b> , <b>GpiSavePS</b> , <b>GpiSetDrawingMode</b>                                                                                                                                                                                                                                                                                                                                        |

## ■ GpiSaveMetaFile

---

```
BOOL GpiSaveMetaFile(hmf, pszFilename)
HMF hmf; /* metafile handle */
PSZ pszFilename; /* pointer to filename */
```

The **GpiSaveMetaFile** function saves a metafile to disk. The function deletes the metafile from memory and invalidates the metafile handle. The application can load the metafile by using the **GpiLoadMetaFile** function.

|                     |                                                                                                                                                                                                                      |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b>   | <i>hmf</i> Identifies the metafile.<br><i>pszFilename</i> Points to a null-terminated string. This string must be a valid MS OS/2 filename that specifies the path and filename of the file to receive the metafile. |
| <b>Return Value</b> | The return value is <b>GPI_OK</b> if the function is successful or <b>GPI_ERROR</b> if an error occurred.                                                                                                            |
| <b>See Also</b>     | <b>GpiLoadMetaFile</b> , <b>GpiPlayMetaFile</b>                                                                                                                                                                      |



## ■ GpiSavePS

---

**LONG GpiSavePS** (*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The **GpiSavePS** function saves the state of the presentation space on the presentation-space (PS) stack. The PS stack is a last-in, first-out stack, maintained by the system, on which an application can save one or more presentation-space states. The function saves the state and returns a unique identifier. The identifier can be used with the **GpiRestorePS** function to restore the given state.

The **GpiSavePS** function saves the current position of the presentation space, the viewing limits, all attributes, and transformation matrices. It also saves the clip path, clip region, logical color table, and references to any loaded logical fonts and the regions created on the associated device context. It does not save the visible region. The function does not actually copy resources such as logical fonts and regions to the PS stack. Instead, it copies a reference to the resource. The application must ensure that the resources are available when these references are restored.

**Parameters**     *hps* Identifies the presentation space. If it identifies a micro presentation space, the drawing mode must be **DM\_DRAW**.

**Return Value**     The return value is an identifier of the saved presentation space if the function is successful or **GPI\_ERROR** if an error occurred. The identifier is equal to the depth of the saved presentation space on the save/restore stack, with 1 as the base level.

**Comments**         The function can be used in an open segment, but only if the drawing mode is **DM\_DRAW**. It can also be used in an element bracket. When it occurs within an open area or path bracket, then the corresponding call to the **GpiRestorePS** function should take place before the bracket is closed. Although the function can be used when creating a metafile, the drawing mode must be **DM\_DRAW** when replaying the metafile.

The PS stack is not the same as the attribute stack (that is, the stack used to save attributes when the attribute mode is **AM\_PRESERVE**).

**See Also**           **GpiPop**, **GpiRestorePS**, **GpiSetDrawingMode**

## ■ GpiSetArcParams

---

**BOOL GpiSetArcParams** (*hps*, *parcp*)

**HPS** *hps*; /\* presentation-space handle \*/

**PARCPARAMS** *parcp*; /\* pointer to structure for arc parameters \*/

The **GpiSetArcParams** function sets the current arc parameters. The arc parameters define the shape and orientation of the ellipse used by the **GpiPointArc**, **GpiFullArc**, and **GpiPartialArc** functions to construct arcs.

The arc parameters define a four-element transformation that maps the unit circle to the ellipse. The transformation has the form:

$$\begin{aligned}x' &= \text{IP} \times x + \text{IR} \times y \\y' &= \text{IS} \times x + \text{IQ} \times y\end{aligned}$$

In the transformation, *IP*, *IR*, *IS*, and *IQ* are the fields of the structure pointed to by the *parcp* parameter. The *IP* and *IQ* fields determine the width and height of ellipse, and the *IS* and *IR* fields determine the shear of the ellipse.

The fields also determine the direction of drawing for arcs drawn using the *GpiFullArc* and *GpiPartialArc* functions. If  $IP \times IQ$  is greater than  $IR \times IS$ , the direction is counterclockwise. If  $IP \times IQ$  is less than  $IR \times IS$ , the direction is clockwise. If  $IP \times IQ$  is equal to  $IR \times IS$ , a straight line is drawn.

If the attribute mode is *AM\_PRESERVE*, the function saves the previous arc parameters on the attribute stack when it sets the new parameters. The previous arc parameters can be retrieved by using the *GpiPop* function.

## Parameters

*hps* Identifies the presentation space.

*parcp* Points to an *ARCPARAMS* structure that contains the arc parameters. The *ARCPARAMS* structure has the following form:

```
typedef struct _ARCPARAMS {
 LONG IP;
 LONG IQ;
 LONG IR;
 LONG IS;
} ARCPARAMS;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

## Return Value

The return value is *GPL\_OK* if the function is successful or *GPL\_ERROR* if an error occurred.

## Comments

Although the arc parameters define the shape and orientation of arcs drawn by the *GpiPointArc*, *GpiFullArc*, and *GpiPartialArc* functions, the functions define the center point and the size of each arc. The *GpiFullArc* and *GpiPartialArc* functions use an explicit center point and a multiplier to scale the ellipse to a desired size. The *GpiPointArc* function computes the center point and size of the final ellipse so that it fits the given points.

Orthogonal transformation provides the most accurate arcs. A transformation is orthogonal if  $IP \times IR$  plus  $IS \times IQ$  equals zero. By default, the arc parameters are as follows:

```
IP = 1 IR = 0
IS = 0 IQ = 1
```

These values produce a unit circle.

Arc-parameter transformation takes place in world coordinates. Any other transformations in force will change the shape of the figure accordingly.

## See Also

*GpiFullArc*, *GpiPartialArc*, *GpiPointArc*, *GpiQueryArcParams*

## ■ GpiSetAttrMode

```
BOOL GpiSetAttrMode(hps, cmdMode)
```

```
HPS hps; /* presentation-space handle */
```

```
LONG cmdMode; /* attribute mode */
```

The *GpiSetAttrMode* function sets the current attribute mode. If the attribute mode is *AM\_PRESERVE*, the system saves the old value of a primitive attribute whenever the attribute is changed to a new value. The saved value of an attribute

can be restored by using the **GpiPop** function. Any attributes that have been saved in a called segment are automatically restored on return to the caller.

**Parameters**

*hps* Identifies the presentation space.

*cmdMode* Specifies the attribute mode. It can be one of the following values:

| Value         | Meaning                     |
|---------------|-----------------------------|
| AM_PRESERVE   | Preserve attributes.        |
| AM_NOPRESERVE | Do not preserve attributes. |

**Return Value**

The return value is **GPL\_OK** if the function is successful or **GPL\_ERROR** if an error occurred.

**Errors**

Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

PMERR\_INV\_ATTR\_MODE  
PMERR\_INV\_MICROPS\_FUNCTION

**Comments**

The attribute mode is an API mode, meaning it does not affect attribute-setting functions that have been stored in a segment. The mode used for functions stored in a segment is the mode in effect when the function was stored. This is also true for any attribute-setting functions that are part of the graphics-order data in a **GpiPutData**, **GpiElement**, or **GpiPlayMetaFile** function.

**See Also**

**GpiElement**, **GpiPlayMetaFile**, **GpiPop**, **GpiPutData**, **GpiSetAttrs**, **GpiSetColor**

## ■ **GpiSetAttrs**

---

**BOOL** **GpiSetAttrs**(*hps*, *flPrimType*, *flAttrsMask*, *flDefsMask*, *pbunAttrs*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *flPrimType*; /\* primitive type \*/  
**ULONG** *flAttrsMask*; /\* attribute mask \*/  
**ULONG** *flDefsMask*; /\* defaults mask \*/  
**PBUNDLE** *pbunAttrs*; /\* pointer to structure for attributes \*/

The **GpiSetAttrs** function sets attributes for the specified primitive type. The function uses the *flPrimType* parameter to determine the type of primitive attribute to set, then uses the *flAttrsMask* parameter to determine which attributes to set for that primitive. If the *flDefsMask* parameter specifies an attribute that is also specified by *flAttrsMask*, the function sets the attribute to its default value. Otherwise, the function uses the value found in the appropriate field of the buffer pointed to by *pbunAttrs*.

The **GpiSetAttrs** function does not accept default values in the buffer fields. To set an attribute to its default value, both the *flAttrsMask* and *flDefsMask* parameters must specify the attribute. Any attribute not specified by *flAttrsMask* remains unchanged, regardless of the *flDefsMask* setting. If the attribute mode is **AM\_PRESERVE** (as set by the **GpiSetAttrMode** function), the system saves the previous value of each attribute that is changed.

**Parameters**

*hps* Identifies the presentation space.

*flPrimType* Specifies which primitive type to set attributes for. It can be one of the following values:

| Value       | Meaning                 |
|-------------|-------------------------|
| PRIM_AREA   | Area primitives         |
| PRIM_CHAR   | Character primitives    |
| PRIM_IMAGE  | Image primitives        |
| PRIM_LINE   | Line and arc primitives |
| PRIM_MARKER | Marker primitives       |

*flAttrsMask* Specifies which attributes to set. The values for this parameter depend on the primitive type specified by *flPrimType*. It can be any combination of the following values for a specific type:

| Type        | Values                                                                                                                      |
|-------------|-----------------------------------------------------------------------------------------------------------------------------|
| PRIM_AREA   | ABB_COLOR, ABB_BACK_COLOR, ABB_MIX_MODE, ABB_BACK_MIX_MODE, ABB_SET, ABB_SYMBOL, ABB_REF_POINT                              |
| PRIM_CHAR   | CBB_COLOR, CBB_BACK_COLOR, CBB_MIX_MODE, CBB_BACK_MIX_MODE, CBB_SET, CBB_MODE, CBB_BOX, CBB_ANGLE, CBB_SHEAR, CBB_DIRECTION |
| PRIM_IMAGE  | IBB_COLOR, IBB_BACK_COLOR, IBB_MIX_MODE, IBB_BACK_MIX_MODE                                                                  |
| PRIM_LINE   | LBB_COLOR, LBB_MIX_MODE, LBB_WIDTH, LBB_GEOM_WIDTH, LBB_TYPE, LBB_END, LBB_JOIN                                             |
| PRIM_MARKER | MBB_COLOR, MBB_BACK_COLOR, MBB_MIX_MODE, MBB_BACK_MIX_MODE, MBB_SET, MBB_SYMBOL, MBB_BOX                                    |

If this parameter is zero, no attributes are set, regardless of the value of the *flDefsMask* and *pbunAttrs* parameters.

*flDefsMask* Specifies which attributes to set to default values. The values for this parameter depend on the primitive type specified by *flPrimType*. Although it can be any combination of the values specified for the *flAttrsMask* parameter, only the attributes that are also specified by *flAttrsMask* are set.

*pbunAttrs* Points to a buffer that contains attribute values for each attribute specified by *flAttrsMask* and not also specified by *flDefsMask*. The buffer format depends on the primitive type specified by *flPrimType*. The following structures can be used for the specified primitive types:

| Type        | Structure    |
|-------------|--------------|
| PRIM_AREA   | AREABUNDLE   |
| PRIM_CHAR   | CHARBUNDLE   |
| PRIM_IMAGE  | IMAGEBUNDLE  |
| PRIM_LINE   | LINEBUNDLE   |
| PRIM_MARKER | MARKERBUNDLE |

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```

PMERR_INV_BACKGROUND_MIX_ATTR
PMERR_INV_CHAR_ANGLE_ATTR
PMERR_INV_CHAR_DIRECTION_ATTR
PMERR_INV_CHAR_MODE_ATTR
PMERR_INV_CHAR_SET_ATTR
PMERR_INV_CHAR_SHEAR_ATTR
PMERR_INV_COLOR_ATTR
PMERR_INV_GEOM_LINE_WIDTH_ATTR
PMERR_INV_LINE_END_ATTR
PMERR_INV_LINE_JOIN_ATTR
PMERR_INV_LINE_TYPE_ATTR
PMERR_INV_LINE_WIDTH_ATTR
PMERR_INV_MARKER_BOX_ATTR
PMERR_INV_MARKER_SET_ATTR
PMERR_INV_MIX_ATTR
PMERR_INV_PATTERN_ATTR
PMERR_INV_PATTERN_SET_ATTR
PMERR_INV_PRIMITIVE_TYPE

```

**See Also** `GpiQueryAttrs`, `GpiSetBackMix`, `GpiSetCharAngle`, `GpiSetCharBox`, `GpiSetColor`, `GpiSetCp`, `GpiSetLineType`, `GpiSetLineWidth`, `GpiSetMarkerBox`, `GpiSetMarkerSet`, `GpiSetMix`, `GpiSetPattern`, `GpiSetPatternRefPoint`, `GpiSetPatternSet`

## ■ **GpiSetBackColor**

**BOOL** `GpiSetBackColor`(*hps*, *clr*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *clr*; /\* background color \*/

The `GpiSetBackColor` function sets the current background color for all primitive types. The background color specifies the color used to fill the background of the primitive, such as the gaps between dashes in a styled line. The function sets the background color to the color specified by *clr*. The *clr* parameter is either an RGB color value or a color-index value, depending on the current logical color table. The actual color the *clr* parameter represents also depends on the current logical color table.

If the attribute mode is `AM_PRESERVE`, the function saves the previous background color on the attribute stack when it sets the new color. The previous background color can be retrieved by using the `GpiPop` function.

**Parameters** *hps* Identifies the presentation space.

*clr* Specifies the color. The values depend on the current logical color table. If the logical color table has been created using the `LCOF_RGB` constant, the values must be RGB color values. Otherwise, the values must be color-index

values. If the default logical color table is used, the parameter can be any one of the following standard color-index values:

| Value          | Meaning                                      |
|----------------|----------------------------------------------|
| CLR_FALSE      | All color planes are zero.                   |
| CLR_TRUE       | All color planes are 1.                      |
| CLR_DEFAULT    | Set to default value; same as zero.          |
| CLR_WHITE      | White.                                       |
| CLR_BLACK      | Black.                                       |
| CLR_BACKGROUND | Reset color (used by the GpiErase function). |
| CLR_BLUE       | Blue.                                        |
| CLR_RED        | Red.                                         |
| CLR_PINK       | Pink.                                        |
| CLR_GREEN      | Green.                                       |
| CLR_CYAN       | Cyan.                                        |
| CLR_YELLOW     | Yellow.                                      |
| CLR_NEUTRAL    | Neutral.                                     |

- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:
- `PMERR_INV_BACKGROUND_COL_ATTR`
- Comments** The functions that draw primitives use the background-mix mode to combine the background color with colors already on the display surface. If the background-mix mode is `BM_LEAVEALONE`, the background color is ignored and existing colors remain unchanged.
- See Also** `GpiCreateLogColorTable`, `GpiErase`, `GpiPop`, `GpiSetBackMix`

## ■ GpiSetBackMix

```

BOOL GpiSetBackMix(hps, IMixMode)
HPS hps; /* presentation-space handle */
LONG IMixMode; /* background-mix mode */

```

The `GpiSetBackMix` function sets the current background-mix mode for all primitive types. The background-mix mode specifies how the background color is combined with colors in underlying primitives. The available background-mix modes depend on the device associated with the presentation space, but all devices support the `BM_LEAVEALONE` and `BM_OVERPAINT` mix modes. If the mix mode specified by *IMixMode* is not supported, the function uses `FM_LEAVEALONE` instead. The `DevQueryCaps` function can be used to determine which mix modes are supported.

If the attribute mode is `AM_PRESERVE`, the function saves the previous background-mix mode on the attribute stack when it sets the new mode. The previous background-mix mode can be retrieved by using the `GpiPop` function.

| <b>Parameters</b>   | <i>hps</i> Identifies the presentation space.<br><i>lMixMode</i> Specifies the background-mix mode. It can be one of the following values:                                                                                                                                                                                                                                                                                                                                                                                                                                                            |       |         |            |                                    |               |                                                                        |       |                                                               |              |                                                   |        |                                                                |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|---------|------------|------------------------------------|---------------|------------------------------------------------------------------------|-------|---------------------------------------------------------------|--------------|---------------------------------------------------|--------|----------------------------------------------------------------|
|                     | <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>BM_DEFAULT</td> <td>The default value (BM_LEAVEALONE).</td> </tr> <tr> <td>BM_LEAVEALONE</td> <td>The background color is ignored. The existing color remains unchanged.</td> </tr> <tr> <td>BM_OR</td> <td>The individual pel colors are combined using the OR operator.</td> </tr> <tr> <td>BM_OVERPAINT</td> <td>The background color replaces the existing color.</td> </tr> <tr> <td>BM_XOR</td> <td>The individual pel colors are combined using the XOR operator.</td> </tr> </tbody> </table> | Value | Meaning | BM_DEFAULT | The default value (BM_LEAVEALONE). | BM_LEAVEALONE | The background color is ignored. The existing color remains unchanged. | BM_OR | The individual pel colors are combined using the OR operator. | BM_OVERPAINT | The background color replaces the existing color. | BM_XOR | The individual pel colors are combined using the XOR operator. |
| Value               | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |       |         |            |                                    |               |                                                                        |       |                                                               |              |                                                   |        |                                                                |
| BM_DEFAULT          | The default value (BM_LEAVEALONE).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |       |         |            |                                    |               |                                                                        |       |                                                               |              |                                                   |        |                                                                |
| BM_LEAVEALONE       | The background color is ignored. The existing color remains unchanged.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |       |         |            |                                    |               |                                                                        |       |                                                               |              |                                                   |        |                                                                |
| BM_OR               | The individual pel colors are combined using the OR operator.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |       |         |            |                                    |               |                                                                        |       |                                                               |              |                                                   |        |                                                                |
| BM_OVERPAINT        | The background color replaces the existing color.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |       |         |            |                                    |               |                                                                        |       |                                                               |              |                                                   |        |                                                                |
| BM_XOR              | The individual pel colors are combined using the XOR operator.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |         |            |                                    |               |                                                                        |       |                                                               |              |                                                   |        |                                                                |
| <b>Return Value</b> | The return value is <code>GPI_OK</code> if the function is successful or <code>GPI_ERROR</code> if an error occurred.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |       |         |            |                                    |               |                                                                        |       |                                                               |              |                                                   |        |                                                                |
| <b>Errors</b>       | Use the <code>WinGetLastError</code> function to retrieve the error value, which may be the following:<br><br><code>PMERR_INV_BACKGROUND_MIX_ATTR</code>                                                                                                                                                                                                                                                                                                                                                                                                                                              |       |         |            |                                    |               |                                                                        |       |                                                               |              |                                                   |        |                                                                |
| <b>Comments</b>     | The background-mix mode is used when drawing areas, images, markers, and characters. For areas and images, the mix mode applies to pels that are not set by the shading pattern. For markers, the mix mode applies to pels not set by the marker pattern. For characters, the mix mode applies to pels not set by the character pattern.                                                                                                                                                                                                                                                              |       |         |            |                                    |               |                                                                        |       |                                                               |              |                                                   |        |                                                                |
| <b>See Also</b>     | <code>DevQueryCaps</code> , <code>GpiPop</code> , <code>GpiSetBackColor</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |       |         |            |                                    |               |                                                                        |       |                                                               |              |                                                   |        |                                                                |

## ■ GpiSetBitmap

**HBITMAP GpiSetBitmap**(*hps*, *hbm*)

**HPS** *hps*; /\* presentation-space handle \*/

**HBITMAP** *hbm*; /\* bitmap handle \*/

The `GpiSetBitmap` function sets a bitmap as the current bitmap in a memory device context. The function sets the bitmap specified by *hbm*. This bitmap must not be set for any other device context and must not be set for area shading. If another bitmap is already set in the device context, the function releases the old bitmap and returns its handle.

The specified presentation space must be currently associated with a memory device context. If the bitmap format is not the same as the device context, the bitmap format must be convertible to one supported by the device. This is guaranteed if the bitmap has one of the standard formats.

|                   |                                                                                                                                                                                                 |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b> | <i>hps</i> Identifies the presentation space.<br><i>hbm</i> Identifies the bitmap to set. If it is <code>NULL</code> , the function releases the bitmap currently set in the associated device. |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Return Value** The return value is the old bitmap handle, NULL for a null handle, or HBM\_ERROR if an error occurred.

**See Also** DevOpenDC, GpiCreateBitmap, GpiLoadBitmap, GpiSetBitmapId

## ■ GpiSetBitmapBits

**LONG GpiSetBitmapBits**(*hps, iScanStart, cScans, pbBuffer, pbmi*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *iScanStart*; /\* index of first scan line \*/  
**LONG** *cScans*; /\* number of scan lines \*/  
**PBYTE** *pbBuffer*; /\* pointer to buffer with bitmap data \*/  
**BITMAPINFO** *pbmi*; /\* pointer to structure with bitmap header table \*/

The **GpiSetBitmapBits** function copies image data to a bitmap from the buffer pointed to by *pbBuffer*. The function copies the image data to the bitmap currently set for the presentation space. The presentation space must be associated with a memory device context.

To copy the image data, the function needs the width and height of the bitmap, the count of planes and adjacent color bits, and the array of RGB color values for the bitmap pels. These values must be specified in the fields of the structure pointed to by *pbmi*. An application must make sure there is sufficient space in the structure to hold all elements of the array of RGB color values. The number of elements in the array depends on the format of the bitmap.

The buffer holding the image data must have sufficient image data to set all pels in the bitmap. The number of bytes for the buffer is equal to the number of scan lines to be copied, multiplied by the width of the bitmap in bytes (multiplied by the number of adjacent color bits and rounded up to the next multiple of 4), multiplied by the number of color planes. If the bitmap width (in bytes) is not an exact multiple of 4, the function discards any extra bits. If the format of the bitmap does not match the device format, the function converts the bitmap. The function can convert standard formats only.

### Parameters

*hps* Identifies the presentation space.

*iScanStart* Specifies the number of the first scan line to copy to the buffer. If it is zero, the function copies the first scan line in the bitmap.

*cScans* Specifies the number of scan lines to copy.

*pbBuffer* Points to the buffer that contains the image data for the bitmap.

*pbmi* Points to the **BITMAPINFO** structure that contains the bitmap information table. The **BITMAPINFO** structure has the following form:

```
typedef struct _BITMAPINFO {
 ULONG cbFix;
 USHORT cx;
 USHORT cy;
 USHORT cPlanes;
 USHORT cBitCount;
 RGB argbColor[1];
} BITMAPINFO;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

### Return Value

The return value is the number of scan lines set, or **GPI\_ALTEERROR** if an error occurred.



**Errors** Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

```
PMERR_INV_DC_TYPE
PMERR_INV_INFO_TABLE
```

**See Also** **GpiAssociate**, **GpiCreateBitmap**, **GpiLoadBitmap**

## I **GpiSetBitmapDimension**

---

**BOOL** **GpiSetBitmapDimension**(*hbm*, *psizlBitmap*)

**HBITMAP** *hbm*; /\* bitmap handle \*/

**PSIZEL** *psizlBitmap*; /\* pointer to structure with size of bitmap \*/

The **GpiSetBitmapDimension** function sets the width and height of a bitmap (in 0.1 millimeter units). Although the system does not use the values set by this function, an application can retrieve the values by using the **GpiQueryBitmapDimension** function.

**Parameters** *hbm* Identifies the bitmap to be set.

*psizlBitmap* Points to the **SIZEL** structure that contains the width and height of the bitmap, in 0.1 millimeter units. The **SIZEL** structure has the following form:

```
typedef struct _SIZEL {
 LONG cx;
 LONG cy;
} SIZEL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is **GPL\_OK** if the function is successful or **GPL\_ERROR** if an error occurred.

**See Also** **GpiCreateBitmap**, **GpiQueryBitmapDimension**, **GpiQueryBitmapParameters**

## I **GpiSetBitmapId**

---

**BOOL** **GpiSetBitmapId**(*hps*, *hbm*, *lcid*)

**HPS** *hps*; /\* presentation-space handle \*/

**HBITMAP** *hbm*; /\* bitmap handle \*/

**LONG** *lcid*; /\* local identifier \*/

The **GpiSetBitmapId** function tags the bitmap with the local identifier specified by *lcid*. The tagged bitmap can subsequently be used for area shading by specifying the local identifier in a call to the **GpiSetPatternSet** function. The bitmap must have a format supported by the device associated with the presentation space, and it must not be set in any other device.

The **GpiDeleteSetId** function can subsequently be used to release the identifier.

**Parameters** *hps* Identifies the presentation space.

*hbm* Identifies the bitmap to tag.

*lcid* Specifies the local identifier for the bitmap. It can be any integer in the range 1 through 254 that has not already been used as a bitmap tag or local identifier for a logical font.

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**See Also** `GpiBitBlt`, `GpiDeleteSetId`, `GpiSetBitmap`, `GpiSetPatternSet`

## ■ GpiSetCharAngle

**BOOL** GpiSetCharAngle (*hps*, *pgradlAngle*)

**HPS** *hps*; /\* presentation-space handle \*/

**PGRADIENTL** *pgradlAngle*; /\* pointer to structure with baseline angle \*/

The `GpiSetCharAngle` function sets the character angle. The character angle specifies the angle at which characters are drawn, relative to the *x*-axis. The function uses the point specified by the *pgradlAngle* parameter to compute the character angle. Any characters drawn are set on a baseline that is parallel to a line drawn through the specified point and the origin.

If the attribute mode is `AM_PRESERVE`, the function saves the previous character angle on the attribute stack when it sets the new angle. The previous character angle can be retrieved by using the `GpiPop` function.

**Parameters** *hps* Identifies the presentation space.

*pgradlAngle* Points to the `GRADIENTL` structure that contains a point that defines the character angle. If both fields in the structure are zero, the function sets the character angle to the default value. The `GRADIENTL` structure has the following form:

```
typedef struct _GRADIENTL {
 LONG x;
 LONG y;
} GRADIENTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_CHAR_ANGLE_ATTR`

**Comments** The character angle is used only when the character mode is `CM_MODE2` or `CM_MODE3`, as set by the `GpiSetCharMode` function. In `CM_MODE2`, the system uses the character angle to compute the position of each character along the baseline. However, when the system draws a character, it places only the lower-left corner of the character box at the computed position. The orientation of the character box remains relative to the *x*-axis instead of the baseline. In `CM_MODE3`, the character boxes are rotated to be 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.

**See Also** GpiQueryCharAngle, GpiSetAttrMode, GpiSetAttrs, GpiSetCharMode

## ■ GpiSetCharBox

**BOOL GpiSetCharBox**(*hps, psizfxBox*)

**HPS** *hps*; /\* presentation-space handle \*/  
**PSIZEF** *psizfxBox*; /\* pointer to structure with character-box size \*/

The **GpiSetCharBox** function sets the current character-box attribute to the specified value. The character-box attribute specifies the width and height of the character box. The character box determines the spacing of consecutive characters along the baseline and the orientation of characters relative to the baseline.

Both width and height can be positive, negative, or zero. When either value is negative, the spacing occurs in the opposite direction to normal and each character is drawn reflected in character-mode 3. For example, a negative height in the standard direction in mode 3 means that the characters are drawn upside down and the string is drawn below the baseline (assuming no other transformations cause inversion). A zero character width or height is also valid; in this case, the string of characters collapses into a line. If both values are zero, the string is drawn as a single point.

If the attribute mode is **AM\_PRESERVE**, the function saves the previous character-box attribute on the attribute stack when it sets the new character box. The previous character-box attribute can be retrieved by using the **GpiPop** function.

**Parameters** *hps* Identifies the presentation space.

*psizfxBox* Points to a **SIZEF** structure that contains the width and height of the character box in world coordinates. The **SIZEF** structure has the following form:

```
typedef struct _SIZEF {
 FIXED cx;
 FIXED cy;
} SIZEF;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

**Return Value** The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**See Also** GpiQueryCharBox, GpiSetAttrMode, GpiSetAttrs

## ■ GpiSetCharDirection

**BOOL GpiSetCharDirection**(*hps, flDirection*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *flDirection*; /\* character direction \*/

The **GpiSetCharDirection** function sets the character direction for drawing characters. The character direction specifies the direction to advance after drawing a character, relative to the baseline.

If the attribute mode is `AM_PRESERVE`, the function saves the previous character direction on the attribute stack when it sets the new direction. The previous character direction can be retrieved by using the `GpiPop` function.

- Parameters** *hps* Identifies the presentation space.  
*flDirection* Specifies the character direction. If it is `CHDIRN_LEFTRIGHT`, the character direction is from left to right. If it is `CHDIRN_DEFAULT`, the function sets the default character direction. The default is from left to right.
- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:  
`PMERR_INV_CHAR_DIRECTION_ATTR`
- See Also** `GpiQueryCharDirection`, `GpiSetAttrMode`, `GpiSetAttrs`

## ■ GpiSetCharMode

```

BOOL GpiSetCharMode(hps, flMode)
HPS hps; /* presentation-space handle */
LONG flMode; /* character mode */

```

The `GpiSetCharMode` function sets the character mode. The character mode specifies which character attributes to use when drawing characters.

If the attribute mode is `AM_PRESERVE`, the function saves the previous character mode on the attribute stack when it sets the new mode. The previous character mode can be retrieved by using the `GpiPop` function.

- Parameters** *hps* Identifies the presentation space.  
*flMode* Specifies the character mode. It can be one of the following values:

| Value                   | Meaning                                                                                                                                                                                                                                                                                                                                         |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>CM_DEFAULT</code> | Use the default.                                                                                                                                                                                                                                                                                                                                |
| <code>CM_MODE1</code>   | Use an image font, as determined by the character-set attribute. The positioning of characters is influenced only by the character-direction attribute; other character attributes are ignored.                                                                                                                                                 |
| <code>CM_MODE2</code>   | Use an image font, as determined by the character-set attribute. The character box, character angle, character direction, character shear, character spacing, character extra, and character break extra values are taken into consideration for positioning successive characters. Individual character definitions are not scaled or rotated. |
| <code>CM_MODE3</code>   | Use a vector font, as determined by the character-set attribute. All character attributes are followed exactly for positioning individual characters, scaling, rotating, and shearing.                                                                                                                                                          |

If the specified mode is not valid, the default is used.

- Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:  
`PMERR_INV_CHAR_MODE_ATTR`
- See Also** `GpiQueryAttrMode`, `GpiSetAttrMode`

## ■ **GpiSetCharSet**

---

**BOOL** `GpiSetCharSet(hps, lcid)`

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *lcid*; /\* local identifier \*/

The `GpiSetCharSet` function sets the current value of the character-set attribute. The character-set attribute specifies the logical font to use for drawing character strings. The logical font, specified by the *lcid* parameter, must have been previously created using the `GpiCreateLogFont` function.

If the attribute mode is `AM_PRESERVE`, the function saves the previous character set on the attribute stack when it sets the new character set. The previous character set can be retrieved by using the `GpiPop` function.

- Parameters** *hps* Identifies the presentation space.  
*lcid* Specifies the local identifier for a logical font. It can be any value in the range 1 through 254 that has been previously set as a local identifier for a logical font. If it is zero, the function sets the character-set attribute to the default character set.
- Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:  
`PMERR_INV_CHAR_SET_ATTR`
- See Also** `GpiQueryCharSet`, `GpiSetAttrMode`, `GpiSetAttrs`

## ■ **GpiSetCharShear**

---

**BOOL** `GpiSetCharShear(hps, pptlShear)`

**HPS** *hps*; /\* presentation-space handle \*/

**PPOINTL** *pptlShear*; /\* pointer to structure with shear angle \*/

The `GpiSetCharShear` function sets the character-shear attribute. The character shear specifies how much to shear (tilt) characters from their normal vertical orientation. The function uses the coordinates of the point specified by *pptlShear* as the end point of a line originating at (0,0). The vertical strokes in subsequent character strings are drawn parallel to the line. The top of the character box remains parallel to the character baseline.

The system draws upright characters if *pptlShear* specifies the point (0,1). This is the default character-shear attribute. If coordinates in the point are both positive or both negative, the characters slope from bottom-left to top-right. If the coordinates have opposite signs (one is positive and one is negative), the characters slope from top-left to bottom-right. Zero should not be used for the *y*-coordinate since it implies an infinite shear. However, if both coordinates are zero, the attribute is set to the default value.

If the attribute mode is `AM_PRESERVE`, the function saves the previous character-shear attribute on the attribute stack when it sets the new character shear. The previous character-shear attribute can be retrieved by using the `GpiPop` function.

**Parameters** *hps* Identifies the presentation space.

*pptlShear* Points to a `POINTL` structure that contains a point that defines the character shear. The `POINTL` structure has the following form:

```
typedef struct _POINTL {
 LONG x;
 LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_CHAR_SHEAR_ATTR`

**See Also** `GpiQueryCharShear`, `GpiSetAttrMode`, `GpiSetAttrs`

## ■ GpiSetClipPath

**BOOL** `GpiSetClipPath`(*hps*, *idPath*, *cmdOptions*)

```
HPS hps; /* presentation-space handle */
LONG idPath; /* clip path identifier */
LONG cmdOptions; /* options */
```

The `GpiSetClipPath` function sets the clip path. The clip path specifies a path in device space that the system uses to clip output. The clip path includes all points inside and on the boundary of the path specified by *idPath*. Since the path coordinates are assumed to be device coordinates, no conversion is applied.

The function creates the clip path by closing any open figures. It then releases any existing clip path (deleting the previous path, if any), and sets the specified path as the clip path. After a path is set as the clip path, it cannot be used again. However, its identifier is free to use for another path.

**Parameters** *hps* Identifies the presentation space.

*idPath* Specifies the identifier of the path to set to the clip path. It can be 1 to specify a path or zero to specify no clip path.

*cmdOptions* Specifies the filling and combining modes. It can be one or two of the following values:

| Value         | Meaning                                                                                                                                                                                        |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SCP_ALTERNATE | Computes the interior of the clip path, using alternate mode. This is the default if neither SCP_ALTERNATE nor SCP_WINDING is given.                                                           |
| SCP_AND       | Intersects the specified path with the current clip path. This value must be specified if the <i>idPath</i> parameter is 1.                                                                    |
| SCP_RESET     | Resets the clip path, releasing the current clip path if any. This value must be specified if the <i>idPath</i> parameter is 0. This is the default if neither SCP_AND nor SCP_RESET is given. |
| SCP_WINDING   | Computes the interior of the clip path, using winding mode.                                                                                                                                    |

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

PMERR\_INV\_PATH\_ID  
PMERR\_PATH\_UNKNOWN

**Comments** Unless the segments in the picture chain have the fast-chaining attribute, the system releases the clip path when it draws each segment. Also, the `GpiResetPS` function releases the current clip path.

**See Also** `GpiBeginArea`, `GpiQueryClipBox`, `GpiResetPS`

## ■ GpiSetClipRegion

**LONG GpiSetClipRegion**(*hps, hrgn, phrgn*)

**HPS** *hps*; /\* presentation-space handle \*/  
**HRGN** *hrgn*; /\* region handle \*/  
**PHRGN** *phrgn*; /\* pointer to variable for previous region handle \*/

The `GpiSetClipRegion` function sets the clip region. The clip region specifies a region in device space that the system uses to clip output. The clip region includes all points inside and on the boundary of the region specified by *hrgn*. Since the region coordinates are assumed to be device coordinates, no conversion is applied.

The function creates the clip region by releasing the previous clip region, copying the handle of this region to the variable pointed to by *phrgn*, and setting the specified region as the clip region. Once a region is set as the clip region, it cannot be used in subsequent region operations. Once released from the clipping region, a region can be used again to restore the previous clip region.

**Parameters** *hps* Identifies the presentation space. The presentation space must be currently associated with the device context for which the specified regions were created.

*hrgn* Identifies the region. If the *hrgn* parameter is `NULL`, the function releases any previous clip region and sets no clip region.

*phrgn* Points to the variable that receives the handle of the previous region. If no previous clip region exists, the function copies NULL to the variable.

**Return Value** The return value is RGN\_NULL, RGN\_RECT, or RGN\_COMPLEX if the function is successful, or RGN\_ERROR if an error occurred.

**See Also** GpiCreateRegion, GpiOffsetClipRegion

## ■ GpiSetColor

**BOOL GpiSetColor(*hps, clr*)**

**HPS *hps*;** /\* presentation-space handle \*/

**LONG *clr*;** /\* color value \*/

The **GpiSetColor** function sets the current foreground color for all primitive types. The foreground color specifies the color used to draw the foreground of the primitive, such as the dashes in a styled line or the diagonal bars in a diagonal shading pattern. The function sets the foreground color to the color specified by *clr*. The *clr* parameter is either an RGB color value or a color index value, depending on the current logical color table. The actual color the *clr* parameter represents also depends on the current logical color table.

If the attribute mode is AM\_PRESERVE, the function saves the previous foreground color on the attribute stack when it sets the new color. The previous foreground color can be retrieved by using the GpiPop function.

**Parameters** *hps* Identifies the presentation space.

*clr* Specifies the color. The values depend on the current logical color table. If the logical color table has been created using the LCOLF\_RGB, the values must be RGB color values. Otherwise, the values must be color-index values. If the default logical color table is used, the parameter can be any one of the following standard color-index values:

| Value          | Meaning                                      |
|----------------|----------------------------------------------|
| CLR_FALSE      | All color planes are zeros.                  |
| CLR_TRUE       | All color planes are ones.                   |
| CLR_DEFAULT    | Default value; same as CLR_NEUTRAL.          |
| CLR_WHITE      | White.                                       |
| CLR_BLACK      | Black.                                       |
| CLR_BACKGROUND | Reset color (used by the GpiErase function). |
| CLR_BLUE       | Blue.                                        |
| CLR_RED        | Red.                                         |
| CLR_PINK       | Pink.                                        |
| CLR_GREEN      | Green.                                       |
| CLR_CYAN       | Cyan.                                        |
| CLR_YELLOW     | Yellow.                                      |
| CLR_NEUTRAL    | Neutral.                                     |
| CLR_DARKGRAY   | Dark gray.                                   |
| CLR_DARKBLUE   | Dark blue.                                   |



| Value         | Meaning     |
|---------------|-------------|
| CLR_DARKRED   | Dark red.   |
| CLR_DARKPINK  | Dark pink.  |
| CLR_DARKGREEN | Dark green. |
| CLR_DARKCYAN  | Dark cyan.  |
| CLR_BROWN     | Brown.      |
| CLR_PALEGRAY  | Light gray. |

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_COLOR_ATTR`

**Comments** The `CLR_BACKGROUND` color for the default logical color table is the natural background color for the device (for example, the paper color for a printer). For the display, the `CLR_BACKGROUND` color is the default window color as set by the `WinSetSysColors` function. The `CLR_NEUTRAL` color for the default logical color table is a device-dependent color that provides a contrasting color to `CLR_BACKGROUND` (for example, it is the ink color for a one-color printer). For the display, it is the default window-text color.

**See Also** `GpiErase`, `GpiPop`, `GpiQueryAttrs`, `GpiQueryColor`, `GpiSetAttrMode`, `GpiSetAttrs`, `GpiSetMix` `WinSetSysColors`

## ■ GpiSetCp

**BOOL GpiSetCp**(*hps*, *idcp*)

**HPS** *hps*; /\* presentation-space handle \*/

**USHORT** *idcp*; /\* code-page identifier \*/

The `GpiSetCp` function selects the code-page identifier to be used for graphics characters for the default character set.

When a presentation space is first created, the code page used is the one defined by the process code page, as set by the `DosSetProcCp` function.

**Parameters** *hps* Identifies the presentation space.

*idcp* Specifies the code-page identifier. The `WinQueryCpList` function can be used to find which code pages are available.

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_CODEPAGE`

**See Also** `DosSetProcCp`, `GpiQueryCp`, `WinQueryCpList`

## ■ GpiSetCurrentPosition

---

**BOOL GpiSetCurrentPosition**(*hps*, *pptl*)

**HPS** *hps*; /\* presentation-space handle \*/  
**PPOINTL** *pptl*; /\* pointer to structure with new position \*/

The **GpiSetCurrentPosition** function sets the current position to the specified point. When used in an area bracket, the function closes the current open figure (if any) and marks the start of a new figure.

This function is equivalent to the **GpiMove** function, except that, if the current attribute mode is **AM\_PRESERVE** (see the **GpiSetAttrMode** function), the function saves the current position before setting it to the new value. It can be restored by using the **GpiPop** function.

**Parameters**     *hps*    Identifies the presentation space.

*pptl*    Points to the **POINTL** structure that contains the new value of the current position. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
 LONG x;
 LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value**    The return value is **GPL\_OK** if the function is successful or **GPL\_ERROR** if an error occurred.

**See Also**        **GpiMove**, **GpiPop**, **GpiQueryCurrentPosition**, **GpiSetAttrMode**

## ■ GpiSetDefaultViewMatrix

---

**BOOL GpiSetDefaultViewMatrix**(*hps*, *cElements*, *pmatlf*, *flType*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *cElements*; /\* number of elements \*/  
**PMATRIXLF** *pmatlf*; /\* pointer to structure with transform matrix \*/  
**LONG** *flType*; /\* transformation type \*/

The **GpiSetDefaultViewMatrix** function sets the default viewing transformation. The function sets the transformation by adding or replacing the existing transformation matrix with the matrix pointed to by the *pmatlf* parameter. The function adds, preempts, or replaces the existing transformation matrix as specified by the *flType* parameter.

The **GpiSetDefaultViewMatrix** function requires a nine-element matrix to set the default viewing transformation. If the specified matrix does not contain nine elements, the function uses the corresponding elements of the identity matrix for each unspecified element. The *cElements* parameter specifies the number of elements in the matrix. If this parameter equals zero, the identity matrix is used.

**Parameters**     *hps*    Identifies the presentation space.

*cElements*    Specifies the number of elements in the matrix to set. It can be any integer in the range 0 to 9.

*pmatlf*    Points to a **MATRIXLF** structure that contains the transformation matrix. The **MATRIXLF** structure has the following form:

```
typedef struct _MATRIXLF {
 FIXED fxM11;
 FIXED fxM12;
 LONG lM13;
 FIXED fxM21;
 FIXED fxM22;
 LONG lM23;
 LONG lM31;
 LONG lM32;
 LONG lM33;
} MATRIXLF;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

*flType* Specifies how a specified transformation matrix should be used to modify the default viewing transformation. It can be one of the following values:

| Value             | Meaning                                                                                                                                                                                                                                                         |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TRANSFORM_ADD     | Additive. The specified transformation matrix is combined with the existing default viewing transformation, in the order of the existing transformation first, the new transformation second. This option is useful for incremental updates to transformations. |
| TRANSFORM_PREEMPT | Preemptive. The specified transformation matrix is combined with the existing default viewing transformation, in the order of the new transformation first, the existing transformation second.                                                                 |
| TRANSFORM_REPLACE | New/replace. The previous default viewing transformation is discarded and replaced by the specified transformation matrix.                                                                                                                                      |

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_TRANSFORM_TYPE`

**See Also** `GpiQueryDefaultViewMatrix`

## ■ GpiSetDrawControl

**BOOL** `GpiSetDrawControl(hps, lControl, flDraw)`

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *lControl*; /\* draw control to change \*/  
**LONG** *flDraw*; /\* drawing flag \*/

The `GpiSetDrawControl` function sets the current draw controls. The draw controls specify whether the system carries out certain actions, such as accumulating boundary data, when the application draws. The function sets the draw control specified by the *lControl* parameter by turning it on or off as specified by the *flDraw* parameter. By default, all draw controls except `DCTL_DISPLAY` are off.

The function cannot be used in an open segment or in an area, path, or element bracket.

## Parameters

*hps* Identifies the presentation space.

*lControl* Specifies the draw control to set. It can be one of the following values:

| Value          | Meaning                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DCTL_BOUNDARY  | Accumulate boundary data. During any output operations except <b>GpiErase</b> , accumulate the bounding rectangle of the drawing. This control can be used with a micro presentation space.                                                                                                                                                                                                                                                            |
| DCTL_CORRELATE | Correlate the <b>GpiPutData</b> , <b>GpiElement</b> , and primitive functions. This control causes these functions to return <b>GPI_HITS</b> if the drawing intersects with the pick aperture. This control applies only to drawing functions used when the drawing mode is <b>DM_DRAW</b> or <b>DM_DRAWANDRETAIN</b> . Also, it does not effect execution of functions stored in a segment. This control can be used with a micro presentation space. |
| DCTL_DISPLAY   | Allow drawing to occur on the device. If this control is off, no output, other than output for <b>GpiErase</b> , appears on the device associated with the presentation space. This control can be used with a micro presentation space.                                                                                                                                                                                                               |
| DCTL_DYNAMIC   | Draw dynamic segments. This control causes the <b>GpiDrawChain</b> , <b>GpiDrawFrom</b> , and <b>GpiDrawSegment</b> functions to call the <b>GpiRemoveDynamics</b> function before drawing and the <b>GpiDrawDynamics</b> function after drawing. The effect is to update the dynamic segments each time the picture chain or a segment is drawn.                                                                                                      |
| DCTL_ERASE     | Erasing before drawing. This control causes the <b>GpiDrawChain</b> , <b>GpiDrawFrom</b> , and <b>GpiDrawSegment</b> functions to call the <b>GpiErase</b> function before drawing.                                                                                                                                                                                                                                                                    |

*flDraw* Specifies whether to turn a draw control on or off. It can be one of the following values:

| Value    | Meaning          |
|----------|------------------|
| DCTL_OFF | Set control off. |
| DCTL_ON  | Set control on.  |

- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:
- `PMERR_INV_DRAW_CONTROL`
  - `PMERR_INV_DRAW_VALUE`
  - `PMERR_INV_MICROPS_FUNCTION`
- See Also** `GpiDrawChain`, `GpiDrawDynamics`, `GpiDrawFrom`, `GpiDrawSegment`, `GpiElement`, `GpiErase`, `GpiPutData`, `GpiQueryDrawControl`, `GpiRemoveDynamics`

## ■ **GpiSetDrawingMode**

---

**BOOL** `GpiSetDrawingMode`(*hps*, *flMode*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *flMode*; /\* drawing mode \*/

The `GpiSetDrawingMode` function sets the drawing mode. The drawing mode affects all subsequent drawing and attribute functions, including the `GpiPutData` function, that occur in open chained segments. The drawing mode specifies whether the functions are drawn, retained, or drawn and retained.

The drawing mode does not affect the functions in unchained segments or outside of segments. For chained segments, the system stores the functions if the `DM_RETAIN` or `DM_DRAWANDRETAIN` mode is set. If the `DM_DRAWANDRETAIN` mode is set, the system draws as well as stores the functions. If the mode is `DM_DRAW`, the functions draw only. For unchained segments, drawing and attribute functions are always retained regardless of the drawing mode. Outside of segments, the functions draw only.

The function cannot be used in an open segment or in an area, path, or element bracket.

- Parameters**
- hps* Identifies the presentation space.
- flMode* Specifies the mode used for subsequent drawing and attribute functions. It can be one of the following values:

| Value                         | Meaning          |
|-------------------------------|------------------|
| <code>DM_DRAW</code>          | Draw only.       |
| <code>DM_RETAIN</code>        | Retain only.     |
| <code>DM_DRAWANDRETAIN</code> | Draw and retain. |

- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

**Errors** Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

PMERR\_INV\_DRAWING\_MODE  
PMERR\_INV\_MICROPS\_FUNCTION

**See Also** GpiCloseSegment, GpiOpenSegment, GpiPutData, GpiQueryDrawingMode

## ■ GpiSetEditMode

**BOOL** GpiSetEditMode (*hps*, *flEditMode*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *flEditMode*; /\* editing mode \*/

The **GpiSetEditMode** function sets the current editing mode. The editing mode specifies whether new elements added to a retained segment are inserted into the segment or whether they replace existing elements. The default editing mode (set by the **GpiCreatePS** or **GpiResetPS** function) is insert.

Although the editing mode applies to retained segments only, the **GpiSetEditMode** function can be used to change the editing mode at any time, regardless of the drawing mode. However, the function cannot be used in an element bracket.

**Parameters** *hps* Identifies the presentation space.

*flEditMode* Specifies the editing mode. It can be one of the following values:

| Value         | Meaning                                                                                                                                                   |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| SEGEM_INSERT  | Insert. The system inserts a new element after the element pointed to by the element pointer. The element pointer is updated to point to the new element. |
| SEGEM_REPLACE | Replace. The system replaces the element pointed by the element pointer. The element pointer does not change.                                             |

**Return Value** The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**Errors** Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

PMERR\_INV\_EDIT\_MODE  
PMERR\_INV\_MICROPS\_FUNCTION

**See Also** GpiCreatePS, GpiQueryEditMode, GpiResetPS, GpiSetElementPointerAtLabel

## ■ GpiSetElementPointer

---

**BOOL GpiSetElementPointer(*hps*, *idElement*)**

**HPS *hps*;** /\* presentation-space handle \*/  
**LONG *idElement*;** /\* element number \*/

The **GpiSetElementPointer** function moves the element pointer to the element specified by *idElement*. The function uses *idElement* as the number of elements to move from the beginning of the segment to reach the new element.

The function can be used only in an open segment and only with **DM\_RETAIN** as the drawing mode. The function cannot be used in an element bracket.

**Parameters**     *hps*    Identifies the presentation space.

*idElement*    Specifies the element number. If the element number is negative, the element pointer is set to zero. If the value is greater than the number of elements in the segment, the element pointer is set to the last element.

**Return Value**    The return value is **GPL\_OK** if the function is successful or **GPL\_ERROR** if an error occurred.

**Errors**            Use the **WinGetLastError** function to retrieve the error value, which may be one of the following:

**PMERR\_INV\_MICROPS\_FUNCTION**  
                  **PMERR\_NOT\_IN\_RETAIN\_MODE**

**See Also**            **GpiQueryElementPointer**

## ■ GpiSetElementPointerAtLabel

---

**BOOL GpiSetElementPointerAtLabel(*hps*, *idLabel*)**

**HPS *hps*;** /\* presentation-space handle \*/  
**LONG *idLabel*;** /\* label identifier \*/

The **GpiSetElementPointerAtLabel** function moves the element pointer to the element containing the specified label. The function starts the search at the next element after the current element-pointer position. If the function does not find the label before reaching the end of the segment, the function leaves the element pointer unchanged and returns an error.

The function can be used only in an open segment and only with **DM\_RETAIN** drawing mode. The function cannot be used in an element bracket.

**Parameters**     *hps*    Identifies the presentation space.

*idLabel*        Specifies the label.

**Return Value**    The return value is **GPL\_OK** if the function is successful or **GPL\_ERROR** if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_INV_MICROPS_FUNCTION
PMERR_LABEL_NOT_FOUND
PMERR_NOT_IN_RETAIN_MODE
```

**See Also** `GpiLabel`, `GpiQueryElementPointer`

## ■ GpiSetGraphicsField

```
BOOL GpiSetGraphicsField(hps, prclField)
HPS hps; /* presentation-space handle */
PRECTL prclField; /* pointer to structure with field */
```

The `GpiSetGraphicsField` function sets the size and position of the graphics field in presentation-page units. The graphics field defines the rectangle in the presentation page to clip. Any output outside the graphics field is not drawn on the device.

The graphics field includes all points in the rectangle interior and all points on the lower and left edges, but not the points on the upper and right edges. Initially, the graphics field has the same size as the page space. The units for the graphics field are not affected by any transformation except the final device transformation.

**Parameters** *hps* Identifies the presentation space.

*prclField* Points to a `RECTL` structure containing the graphics field. It is an error if the top coordinate is less than the bottom, or the right coordinate less than the left. All values must be presentation-page units. The `RECTL` structure has the following form:

```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

**See Also** `GpiQueryGraphicsField`

## ■ GpiSetInitialSegmentAttrs

```
BOOL GpiSetInitialSegmentAttrs(hps, flAttribute, flAttrFlag)
HPS hps; /* presentation-space handle */
LONG flAttribute; /* attribute type */
LONG flAttrFlag; /* attribute on/off flag */
```

The `GpiSetInitialSegmentAttrs` function sets the initial segment attributes. The system applies these attributes to each new segment as it is created. The function can change these attributes, one at a time, by turning them on or off.



When the presentation space is first created, the initial segment attributes create visible, chained, dynamic, and fast-chaining segments that propagate visibility and detectability, even though they are not detectable. These attribute have no effect on primitives outside of segments.

## Parameters

*hps* Identifies the presentation space.

*flAttribute* Specifies the segment attribute to change. It can be one of the following values:

| Value                | Meaning                                                                                                                                                                                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ATTR_CHAINED         | Adds the segment to the picture chain. If this attribute is off, a new segment is an unchained segment. Although unchained segments can be drawn individually, they cannot be added to the picture chain. (However, they can be called from a segment in the picture chain.)      |
| ATTR_DETECTABLE      | Enables correlation for the segment. If this attribute is on, the <b>GpiCorrelateChain</b> , <b>GpiCorrelateFrom</b> , and <b>GpiCorrelateSegment</b> functions can be used to correlate each new segment.                                                                        |
| ATTR_DYNAMIC         | Creates a dynamic segment. Dynamic segments are designed to be rapidly updated without affecting other segments in the picture chain. The system draws dynamic segments using the exclusive-OR operator. This lets the segments be erased. Only retained segments can be dynamic. |
| ATTR_FASTCHAIN       | Relaxes the automatic reset of primitive attributes for the segment. If this attribute is off, the system resets all primitive attributes just before a segment in the picture chain is drawn.                                                                                    |
| ATTR_PROP_DETECTABLE | Forces all segments beneath the segment to inherit the detectable attribute. If this attribute is on, all segments called by the segment can be correlated.                                                                                                                       |
| ATTR_PROP_VISIBLE    | Forces all segments beneath the segment to be visible. The visibility lasts only as long as the segment is called by the segment with this attribute on.                                                                                                                          |
| ATTR_VISIBLE         | Makes the segment visible. The attribute lets the system draw the segment on the output device.                                                                                                                                                                                   |

*flAttrFlag* Specifies whether to turn the attribute on or off. If it is **ATTR\_ON**, the function turns the attribute on. If it is **ATTR\_OFF**, the function turns the attribute off.

- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:
- `PMERR_INV_MICROPS_FUNCTION`  
`PMERR_INV_SEG_ATTR`
- See Also** `GpiCorrelateChain`, `GpiCorrelateFrom`, `GpiCorrelateSegment`, `GpiOpenSegment`, `GpiSetDrawControl`, `GpiSetDrawingMode`

## ■ GpiSetLineEnd

**BOOL** GpiSetLineEnd(*hps*, *lLineEnd*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *lLineEnd*; /\* line end \*/

The `GpiSetLineEnd` function sets the current line-end attribute. The line-end attribute specifies the shape of the ends of lines drawn by the `GpiStrokePath` function or by the `GpiModifyPath` and `GpiFillPath` function pair.

If the attribute mode is `AM_PRESERVE`, the function saves the previous line-end attribute on the attribute stack when it sets the new line end. The previous line-end attribute can be retrieved by using the `GpiPop` function.

**Parameters** *hps* Identifies a presentation space.

*lLineEnd* Specifies the line end. It may be one of the following values:

| Value                        | Meaning                                                                                                                                    |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <code>LINEEND_DEFAULT</code> | Use default.                                                                                                                               |
| <code>LINEEND_FLAT</code>    | Flat. The line is ended at the end point of the original path.                                                                             |
| <code>LINEEND_ROUND</code>   | Round. The line is ended as if a circle having a diameter equal to the line width is drawn centered on the end point of the original path. |
| <code>LINEEND_SQUARE</code>  | Square. The line is ended as if a square having the same width as the line is drawn centered on the end point of the original path.        |

**Return Value** The return value is `TRUE` if the function is successful. Otherwise, it is `FALSE`, indicating that an error occurred.

Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_LINE_END_ATTR`

**See Also** `GpiFillPath`, `GpiModifyPath`, `GpiPop`, `GpiQueryAttrs`, `GpiQueryLineEnd`, `GpiSetAttrMode`

## ■ GpiSetLineJoin

---

**BOOL GpiSetLineJoin**(*hps*, *flLineJoin*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *flLineJoin*; /\* line-join flags \*/

The **GpiSetLineJoin** function sets the current line-join attribute to the specified value. The line-join attribute specifies how the intersection of lines (at the end points) are drawn by the **GpiStrokePath** function or by the **GpiModifyPath** and **GpiFillPath** function pair.

If the attribute mode is **AM\_PRESERVE**, the function saves the previous line-join attribute on the attribute stack when it sets the new line join. The previous line-join attribute can be retrieved by using the **GpiPop** function.

**Parameters** *hps* Identifies the presentation space.

*flLineJoin* Specifies line-join flags. It can be one of the following values:

| Value            | Meaning |
|------------------|---------|
| LINEJOIN_BEVEL   | Bevel   |
| LINEJOIN_DEFAULT | Default |
| LINEJOIN_MITRE   | Mitre   |
| LINEJOIN_ROUND   | Round   |

**Return Value** The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**Errors** Use the **WinGetLastError** function to retrieve the error value, which may be the following:

PMERR\_INV\_LINE\_JOIN\_ATTR

**See Also** **GpiFillPath**, **GpiModifyPath**, **GpiPop**, **GpiQueryAttrs**, **GpiQueryLineJoin**, **GpiSetAttrMode**, **GpiSetAttrs**, **GpiStrokePath**

## ■ GpiSetLineType

---

**BOOL GpiSetLineType**(*hps*, *flLineType*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *flLineType*; /\* line type \*/

The **GpiSetLineType** function sets the current cosmetic line-type attribute to the specified value.

**Parameters** *hps* Identifies the presentation space.

*flLineType* Specifies the line type. If the specified line type is not valid, the default is used. A valid line type is one of the following:

```

LINETYPE_DOT
LINETYPE_SHORTDASH
LINETYPE_DASHDOT
LINETYPE_DOUBLEDOT
LINETYPE_LONGDASH
LINETYPE_DASHDOUBLEDOT
LINETYPE_SOLID
LINETYPE_INVISIBLE
LINETYPE_ALTERNATE

```

- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:
- ```
PMERR_INV_LINE_TYPE_ATTR
```
- Comments**
- A non-solid line type consists of a sequence of on-and-off runs that gives the appearance of a dotted line.
- This attribute specifies the cosmetic line type, which is used for all line and curve drawing. It does not depend on transforms. For example, dashes do not become longer if a “zoom in” occurs.
- The eight standard line types are implemented on each device to give a good appearance based on the “pel” resolution. Their definitions cannot be changed by applications, nor can applications define additional cosmetic line types.
- The system maintains position within the line-type definition. For example, a curve may be implemented as a polyline. However, the following functions cause the position to be reset:
- ```

GpiCallSegmentMatrix
GpiMove
GpiPop
GpiSetCurrentPosition
GpiSetModelTransformMatrix
GpiSetPageViewport
GpiSetSegmentTransformMatrix

```
- The attribute mode (see `GpiSetAttrMode`) determines whether the current value of the line type attribute is preserved.
- See Also** `GpiQueryAttrs`, `GpiQueryLineType`, `GpiSetAttrMode`, `GpiSetAttrs`

## ■ GpiSetLineWidth

```

BOOL GpiSetLineWidth(hps, fxLineWidth)
HPS hps; /* presentation-space handle */
FIXED fxLineWidth; /* line width */

```

The `GpiSetLineWidth` function sets the current cosmetic line-width attribute to the specified value. The line width specifies the width of cosmetic lines, that is, lines drawn by functions such as `GpiLine`. The function treats the line width as a multiplier for the normal line thickness for the device.

If the attribute mode is `AM_PRESERVE`, the function saves the previous line width on the attribute stack when it sets the new width. The previous line width can be retrieved by using the `GpiPop` function.

**Parameters** *hps* Identifies the presentation space.

*fxLineWidth* Specifies the line-width multiplier. It must be a fixed-point number or one of the following values:

| Value                          | Meaning            |
|--------------------------------|--------------------|
| <code>LINEWIDTH_DEFAULT</code> | Default            |
| <code>LINEWIDTH_NORMAL</code>  | Normal width (1.0) |

Any other positive value is a multiplier on the normal line width. Only normal line widths are currently supported. Any value greater than `LINEWIDTH_NORMAL` will result in a warning.

**Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_LINE_WIDTH_ATTR`

**Comments** The thickness of cosmetic lines is not affected by transformations.

**See Also** `GpiLine`, `GpiPop`, `GpiQueryAttrs`, `GpiQueryLineWidth`, `GpiSetAttrMode`, `GpiSetAttrs`

## ■ **GpiSetLineWidthGeom**

**BOOL** `GpiSetLineWidthGeom`(*hps*, *lLineWidth*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *lLineWidth*; /\* line width \*/

The `GpiSetLineWidthGeom` function sets the current geometric-line-width attribute to the specified value. The geometric line width specifies the width of lines drawn by using the `GpiStrokePath` function or the `GpiModifyPath` and `Gpi-FillPath` pair.

If the attribute mode is `AM_PRESERVE`, the function saves the previous geometric-line width on the attribute stack when it sets the new width. The previous geometric-line width can be retrieved using the `GpiPop` function.

**Parameters** *hps* Identifies the presentation space.

*lLineWidth* Specifies the geometric-line width in world coordinates. This value cannot be negative. If it is zero, the resulting line has zero width.

**Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_GEOM_LINE_WIDTH_ATTR`

**See Also** `GpiFillPath`, `GpiModifyPath`, `GpiPop`, `GpiQueryAttrs`, `GpiQueryLineWidthGeom`, `GpiSetAttrMode`, `GpiSetAttrs`, `GpiStrokePath`

## ■ GpiSetMarker

**BOOL** `GpiSetMarker`(*hps*, *ISymbol*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *ISymbol*; /\* marker symbol \*/

The `GpiSetMarker` function sets the value of the marker attribute. The marker attribute specifies the marker drawn by the `GpiMarker` function.

If the attribute mode is `AM_PRESERVE`, the function saves the previous marker attribute on the attribute stack when it sets the new marker. The previous marker attribute can be retrieved by using the `GpiPop` function.

**Parameters** *hps* Identifies the presentation space.

*ISymbol* Specifies the identity of the required marker symbol. Zero selects the default marker symbol; a value in the range 1 through 255 identifies a symbol in the current marker set. If the default marker set is used, the parameter can be one of the following values:

| Value                               | Meaning               |
|-------------------------------------|-----------------------|
| <code>MARKSYM_CROSS</code>          | Cross                 |
| <code>MARKSYM_PLUS</code>           | Plus sign             |
| <code>MARKSYM_DIAMOND</code>        | Diamond               |
| <code>MARKSYM_SQUARE</code>         | Square                |
| <code>MARKSYM_SIXPOINTSTAR</code>   | Six-pointed star      |
| <code>MARKSYM_EIGHTPOINTSTAR</code> | Eight-pointed star    |
| <code>MARKSYM_SOLIDDIAMOND</code>   | Solid diamond         |
| <code>MARKSYM_SOLIDSQUARE</code>    | Solid square          |
| <code>MARKSYM_DOT</code>            | Dot                   |
| <code>MARKSYM_SMALLCIRCLE</code>    | Small circle          |
| <code>MARKSYM_BLANK</code>          | Blank (nothing drawn) |

**Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

**See Also** `GpiMarker`, `GpiPop`, `GpiQueryMarker`

## ■ GpiSetMarkerBox

---

**BOOL GpiSetMarkerBox**(*hps*, *psizfxBox*)

**HPS** *hps*; /\* presentation-space handle \*/  
**PSIZEF** *psizfxBox*; /\* pointer to structure with marker box size \*/

The **GpiSetMarkerBox** function sets the current marker-box attribute. The marker box specifies the width and height of markers drawn by the **GpiMarker** function.

If the attribute mode is **AM\_PRESERVE**, the function saves the previous marker-box attribute on the attribute stack when it sets the new marker box. The previous marker-box attribute can be retrieved by using the **GpiPop** function.

**Parameters** *hps* Identifies the presentation space.

*psizfxBox* Points to a **SIZEF** structure containing the size of the marker box, in world coordinates. The **SIZEF** structure has the following form:

```
typedef struct _SIZEF {
 FIXED cx;
 FIXED cy;
} SIZEF;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**Errors** Use the **WinGetLastError** function to retrieve the error value, which may be the following:

**PMERR\_INV\_MARKER\_BOX\_ATTR**

**See Also** **GpiQueryAttrs**, **GpiQueryMarkerBox**, **GpiSetAttrs**

## ■ GpiSetMarkerSet

---

**BOOL GpiSetMarkerSet**(*hps*, *lcid*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *lcid*; /\* local identifier \*/

The **GpiSetMarkerSet** function sets the current marker-set attribute. The marker-set attribute specifies the marker or character set from which markers can be chosen. The marker set can be the default marker set or any logical font created by using the **GpiCreateLogFont** function.

If the attribute mode is **AM\_PRESERVE**, the function saves the previous marker-set attribute on the attribute stack when it sets the new marker set. The previous marker-set attribute can be retrieved by using the **GpiPop** function.

**Parameters** *hps* Identifies the presentation space.

*lcid* Specifies the logical identifier for the marker set. It can be any integer in the range 1 to 254 for which a logical font has been created, or it can be **LCID\_DEFAULT** to specify the default marker set.

- Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.
- Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:  
`PMERR_INV_MARKER_SET_ATTR`
- See Also** `GpiCreateLogFont`, `GpiPop`, `GpiQueryAttrs`, `GpiQueryMarkerSet`, `GpiSetAttrs`

## ■ GpiSetMetaFileBits

---

**BOOL** GpiSetMetaFileBits(*hmf*, *off*, *cbBuffer*, *pbBuffer*)

**HMF** *hmf*; /\* message-queue handle \*/

**LONG** *off*; /\* offset into the metafile \*/

**LONG** *cbBuffer*; /\* length of the metafile data \*/

**PBYTE** *pbBuffer*; /\* pointer to the metafile data \*/

The `GpiSetMetaFileBits` function copies data to the metafile specified by *hmf* from the buffer pointed to by the *pbBuffer* parameter. The function inserts the bytes into the metafile, up to the number of bytes specified by *cbBuffer*, at the byte in the metafile where the offset from the beginning of the metafile is specified by the *off* parameter.

The application must ensure that the metafile data has the correct format. The data should not be changed after it is created by the `GpiQueryMetaFileBits` function.

- Parameters**
- hmf* Identifies the metafile memory.
- off* Specifies the offset in bytes from the beginning of the metafile to the first byte that receives copied data.
- cbBuffer* Specifies the number of bytes of metafile data to copy.
- pbBuffer* Points to the buffer to receive the metafile data. It must have the number of bytes specified by the *cbBuffer* parameter.

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

`PMERR_INV_METAFILE_LENGTH`  
`PMERR_INV_METAFILE_OFFSET`

**See Also** `GpiQueryMetaFileBits`



## ■ GpiSetMix

**BOOL** GpiSetMix(*hps*, *fMixMode*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *fMixMode*; /\* color-mixing mode \*/

The **GpiSetMix** function sets the current foreground-mix mode. The foreground-mix mode specifies how the foreground color is combined with the colors in underlying primitives. The available mixing modes depend on the device that is associated with the presentation space, but all devices support the mixing modes **FM\_LEAVEALONE**, **FM\_XOR**, and **FM\_OVERPAINT**. If the mixing mode specified by *fMixMode* is not supported, the function uses **FM\_OVERPAINT** instead. The **DevQueryCaps** function can be used to determine which mixing modes are supported.

If the attribute mode is **AM\_PRESERVE**, the function saves the previous foreground-mix mode on the attribute stack when it sets the new mode. The previous foreground-mix mode can be retrieved using the **GpiPop** function.

### Parameters

*hps* Identifies the presentation space.

*fMixMode* Specifies the color-mixing mode. It can be one of the following values:

| Value                 | Meaning                                                                                                |
|-----------------------|--------------------------------------------------------------------------------------------------------|
| <b>FM_AND</b>         | The individual pel colors are combined using the AND operator.                                         |
| <b>FM_DEFAULT</b>     | Default. This is the same as <b>FM_OVERPAINT</b> .                                                     |
| <b>FM_INVERT</b>      | All existing pel colors are inverted. The foreground color is ignored.                                 |
| <b>FM_LEAVEALONE</b>  | The foreground color is ignored. The existing color remains unchanged.                                 |
| <b>FM_MASKSRCNOT</b>  | The individual pel colors are combined using the AND operator after inverting the existing pel colors. |
| <b>FM_MERGENOTSRC</b> | The individual pel colors are combined using the OR operator after inverting the foreground color.     |
| <b>FM_MERGESRCNOT</b> | The individual pel colors are combined using the OR operator after inverting the existing pel colors.  |
| <b>FM_NOTCOPYSRC</b>  | The inverse of the foreground color replaces the existing color.                                       |
| <b>FM_NOTMASKSRC</b>  | The individual pel colors are inverted after being combined using the AND operator.                    |
| <b>FM_NOTMERGESRC</b> | The individual pel colors are inverted after being combined using the OR operator.                     |
| <b>FM_NOTXORSRC</b>   | The individual pel colors are inverted after being combined using the XOR operator.                    |
| <b>FM_ONE</b>         | All pels are set to one.                                                                               |
| <b>FM_OR</b>          | The individual pel colors are combined using the OR operator.                                          |

| Value        | Meaning                                                                                             |
|--------------|-----------------------------------------------------------------------------------------------------|
| FM_OVERPAINT | The foreground color replaces the existing color.                                                   |
| FM_SUBTRACT  | The individual pel colors are combined using the AND operator after inverting the foreground color. |
| FM_XOR       | The individual pel colors are combined using the XOR operator.                                      |
| FM_ZERO      | All pels are set to zero.                                                                           |

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**See Also** `DevQueryCaps`, `GpiQueryMix`, `GpiSetBackMix`

## ■ GpiSetModelTransformMatrix

**BOOL** `GpiSetModelTransformMatrix`(*hps*, *cElements*, *pmatlf*, *flType*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *cElements*; /\* number of elements \*/

**PMATRIXLF** *pmatlf*; /\* pointer to structure with transformation matrix \*/

**LONG** *flType*; /\* transformation types \*/

The `GpiSetModelTransformMatrix` function sets the model transformation. The model transformation applies to all primitives used inside and outside of segments. The function sets the transformation by adding or replacing the existing transformation matrix with the matrix pointed to by the *pmatlf* parameter. The function adds, preempts, or replaces the existing transformation matrix as specified by the *flType* parameter.

The `GpiSetModelTransformMatrix` function requires a nine-element matrix to set the model transformation. If the specified matrix does not contain nine elements, the function uses the corresponding elements of the identity matrix for each unspecified element. The *cElements* parameter specifies the number of elements in the matrix. If this parameter equals zero, the identity matrix is used. If scaling values greater than one are given, care must be taken that the combined effect of this and any other relevant transformations do not exceed the limit for fixed-point numbers.

If the attribute mode is `AM_PRESERVE`, the function saves the previous model transformation on the attribute stack when it sets the new transformation. The previous model transformation can be retrieved using the `GpiPop` function.

**Parameters** *hps* Identifies the presentation space.

*cElements* Specifies the number of elements in the matrix to set. It can be any integer in the range 0 through 9.

*pmatlf* Points to a `MATRIXLF` structure that contains the transformation matrix. The `MATRIXLF` structure has the following form:

```
typedef struct _MATRIXLF {
 FIXED fxM11;
 FIXED fxM12;
 LONG lM13;
 FIXED fxM21;
 FIXED fxM22;
 LONG lM23;
 LONG lM31;
 LONG lM32;
 LONG lM33;
} MATRIXLF;
```

For a full description, see the “Structures” section.

*fType* Specifies how a specified matrix should be used to modify the segment transformation. It can be one of the following values:

| Value             | Meaning                                                                                                                                                                                                                                               |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TRANSFORM_ADD     | Additive. The specified transformation matrix is combined with the existing model transformation, in the order of the existing transformation first, the new transformation second. This option is useful for incremental updates to transformations. |
| TRANSFORM_PREEMPT | Preemptive. The specified transformation matrix is combined with the existing model transformation, in the order of the new transformation first, the existing transformation second.                                                                 |
| TRANSFORM_REPLACE | New/replace. The previous model transformation is discarded and replaced by the specified transformation matrix.                                                                                                                                      |

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_TRANSFORM_TYPE`

**See Also** `GpiQueryModelTransformMatrix`, `GpiSetAttrMode`

## ■ **GpiSetPageViewport**

**BOOL** `GpiSetPageViewport`(*hps*, *prclViewport*)

**HPS** *hps*; /\* presentation-space handle \*/

**RECTL** *prclViewport*; /\* pointer to structure with page viewport \*/

The `GpiSetPageViewport` function sets the page viewport within device space. The page viewport and the presentation page size (specified by the `GpiCreatePS` function) specify the device transformation.

The function can only be used if the presentation space has an associated device context.

**Parameters** *hps* Identifies the presentation space.

*prclViewport* Points to a `RECTL` structure defining the page viewport in device units. The `RECTL` structure has the following form:

```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

- Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.
- Comments** When a presentation space is associated with a device context, the system sets a default page viewport. The default page viewport depends on the page units selected for the presentation space.
- See Also** `GpiCreatePS`, `GpiQueryPageViewport`

## ■ GpiSetPattern

**BOOL** GpiSetPattern(*hps*, *ISymbol*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *ISymbol*; /\* shading pattern \*/

The `GpiSetPattern` function sets the current value of the pattern attribute. The pattern attribute specifies the shading pattern used to fill areas. The pattern can be any pattern from the default pattern set or any character from a raster font. If the specified pattern is not valid for the device associated with the presentation space, the function sets the default pattern.

If the attribute mode is `AM_PRESERVE`, the function saves the previous pattern attribute on the attribute stack when it sets the new pattern. The previous pattern attribute can be retrieved using the `GpiPop` function.

**Parameters** *hps* Identifies the presentation space.

*ISymbol* Specifies the shading pattern to set. The values depend on the particular pattern set selected by the pattern-set attribute. It can be zero to specify the default pattern, or a number in the range 1 through 255 to specify a particular pattern from the current pattern set. If the default pattern set is used, it can be one of the following values:

| Value                       | Meaning                               |
|-----------------------------|---------------------------------------|
| <code>PATSYM_BLANK</code>   | Blank pattern (background only)       |
| <code>PATSYM_DEFAULT</code> | Default pattern (device-dependent)    |
| <code>PATSYM_DENSE1</code>  | Density-1 pattern (mostly foreground) |
| <code>PATSYM_DENSE2</code>  | Density-2 pattern                     |
| <code>PATSYM_DENSE3</code>  | Density-3 pattern                     |
| <code>PATSYM_DENSE4</code>  | Density-4 pattern                     |
| <code>PATSYM_DENSE5</code>  | Density-5 pattern                     |
| <code>PATSYM_DENSE6</code>  | Density-6 pattern                     |
| <code>PATSYM_DENSE7</code>  | Density-7 pattern                     |
| <code>PATSYM_DENSE8</code>  | Density-8 pattern (mostly background) |

| Value           | Meaning                                       |
|-----------------|-----------------------------------------------|
| PATSYM_DIAG1    | Upward-diagonal pattern (steep)               |
| PATSYM_DIAG2    | Upward-diagonal pattern (gradual)             |
| PATSYM_DIAG3    | Downward-diagonal pattern (steep)             |
| PATSYM_DIAG4    | Downward-diagonal pattern (gradual)           |
| PATSYM_HALFTONE | Alternating foreground and background pattern |
| PATSYM_HORIZ    | Horizontal pattern                            |
| PATSYM_NOSHADE  | Blank pattern (background only)               |
| PATSYM_SOLID    | Solid pattern (foreground only)               |
| PATSYM_VERT     | Vertical pattern                              |

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_INV_PATTERN_SET_ATTR`

**Comments** If the current pattern set specifies a bitmap (see the `GpiSetBitmapId` or `GpiSetPatternSet` function), the pattern attribute is ignored.

**See Also** `GpiQueryPattern`, `GpiSetBitmapId`, `GpiSetPatternRefPoint`, `GpiSetPatternSet`

## ■ **GpiSetPatternRefPoint**

**BOOL** `GpiSetPatternRefPoint` (*hps*, *pptlRef*)

**HPS** *hps*; /\* presentation-space handle \*/

**PPOINTL** *pptlRef*; /\* pointer to structure with reference point \*/

The `GpiSetPatternRefPoint` function sets the current pattern reference point to the specified value. The pattern reference point is the point to which the origin of the fill pattern maps. The pattern reference point does not need to be inside the actual area to be filled. The default pattern reference point is (0,0).

If the attribute mode is `AM_PRESERVE`, the function saves the previous pattern reference point on the attribute stack when it sets the new reference point. The previous pattern reference point can be retrieved using the `GpiPop` function.

**Parameters** *hps* Identifies the presentation space.

*pptlRef* Points to the `POINTL` structure that contains the pattern reference point in world coordinates. The `POINTL` structure has the following form:

```
typedef struct _POINTL {
 LONG x;
 LONG y;
} POINTL;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

|                     |                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Return Value</b> | The return value is <code>GPL_OK</code> if the function is successful or <code>GPL_ERROR</code> if an error occurred.                                                                                                                                                                                                                                                                                         |
| <b>Comments</b>     | The pattern reference point is subject to all transformations. This means that moving an area by using a transformation also moves the fill pattern so that the pattern keeps its position relative to the area boundaries. This allows part of a picture to be moved using the <code>GpiBitBlt</code> function, and the remainder is drawn without discontinuity by changing the appropriate transformation. |
| <b>See Also</b>     | <code>GpiBitBlt</code> , <code>GpiQueryPatternRefPoint</code> , <code>GpiSetAttrMode</code> , <code>GpiSetPattern</code> , <code>GpiSetPatternSet</code>                                                                                                                                                                                                                                                      |

## ■ GpiSetPatternSet

```

BOOL GpiSetPatternSet(hps, lcid)
HPS hps; /* presentation-space handle */
LONG lcid; /* local identifier */

```

The `GpiSetPatternSet` function sets the current pattern-set attribute to the specified value. The pattern set specifies the pattern or character set from which patterns can be chosen. The pattern set can be the default pattern set, any logical font created using the `GpiCreateLogFont` function, or any tagged bitmap.

If a logical font is set as the pattern set, the `GpiSetPattern` function can be used to choose which character in the font as the pattern. Depending on the device associated with the presentation space, not all of the character may be used as the pattern. For example, some devices use only 8-by-8 patterns, and therefore use only the lower-left corner of a character.

If a tagged bitmap is set as the pattern set, the bitmap is used as the pattern. As with characters, not all of the bitmap may be used as the pattern. Also, if the bitmap is color and the device is monochrome, the system converts the bitmap to monochrome.

If the attribute mode is `AM_PRESERVE`, the function saves the previous pattern-set attribute on the attribute stack when it sets the new pattern set. The previous pattern-set attribute can be retrieved using the `GpiPop` function.

|                     |                                                                                                                                                                                                                                                                                                       |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b>   | <i>hps</i> Identifies the presentation space.<br><i>lcid</i> Specifies the local identifier for the pattern set. It can be any integer in the range 1 through 254 for which a logical font or tagged bitmap has been created. It can be <code>LCID_DEFAULT</code> to specify the default pattern set. |
| <b>Return Value</b> | The return value is <code>GPL_OK</code> if the function is successful or <code>GPL_ERROR</code> if an error occurred.                                                                                                                                                                                 |
| <b>Errors</b>       | Use the <code>WinGetLastError</code> function to retrieve the error value, which may be the following:<br><br><code>PMERR_INV_PATTERN_ATTR</code>                                                                                                                                                     |
| <b>Comments</b>     | Certain fonts cannot be used for patterns. The function returns an error if an application attempts to set such a font as the pattern set. Fonts that cannot be used as patterns include device fonts which cannot be used for shading, and any kind of raster font used for a plotter device.        |
| <b>See Also</b>     | <code>GpiQueryPatternSet</code> , <code>GpiSetPattern</code> , <code>GpiSetPatternRefPoint</code>                                                                                                                                                                                                     |

## ■ GpiSetPel

---

**LONG GpiSetPel**(*hps*, *pptl*)

**HPS** *hps*; /\* presentation-space handle \*/  
**PPOINTL** *pptl*; /\* pointer to structure with point position \*/

The **GpiSetPel** function sets the pel at the specified position to the current foreground color. The pel's position is expressed in world coordinates. If the pel is not visible (that is, the point lies outside the clip area), the color remains unchanged.

**Parameters**     *hps*   Identifies the presentation space.

*pptl*   Points to a **POINTL** structure containing the position in world coordinates. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
 LONG x;
 LONG y;
} POINTL;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

**Return Value**   The return value is **GPL\_OK** or **GPL\_HITS** if the function is successful (it is **GPL\_HITS** if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is **GPL\_ERROR** if an error occurs.

**See Also**       **GpiQueryPel**, **GpiSetAttrs**, **GpiSetColor**, **GpiSetMix**

## ■ GpiSetPickAperturePosition

---

**BOOL GpiSetPickAperturePosition**(*hps*, *pptlPick*)

**HPS** *hps*; /\* presentation-space handle \*/  
**PPOINTL** *pptlPick*; /\* pointer to structure with center of pick aperture \*/

The **GpiSetPickAperturePosition** function sets the center of the pick aperture, in presentation-page space, for subsequent non-retained correlation operations.

**Parameters**     *hps*   Identifies the presentation space.

*pptlPick*   Points to the **POINTL** structure that contains the center of the pick aperture. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
 LONG x;
 LONG y;
} POINTL;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

**Return Value**   The return value is **GPL\_OK** if the function is successful or **GPL\_ERROR** if an error occurred.

**See Also**       **GpiQueryPickAperturePosition**, **GpiSetPickApertureSize**

## ■ GpiSetPickApertureSize

```

BOOL GpiSetPickApertureSize (hps, flOption, psizlPick)
HPS hps; /* presentation-space handle */
LONG flOption; /* options */
PSIZEL psizlPick; /* pointer to structure with pick-aperture size */

```

The **GpiSetPickApertureSize** function sets the pick-aperture size. The function sets the pick aperture to either the default value or to the size specified by *psizlPick*. The default size is a rectangle in presentation-page space that produces a square in device space that has a width and height equal to the default-character cell height.

### Parameters

*hps* Identifies the presentation space.

*flOption* Specifies the pick-aperture type. It can be one of the following values:

| Value          | Meaning                                                               |
|----------------|-----------------------------------------------------------------------|
| PICKAP_DEFAULT | Use default pick aperture. The <i>psizlPick</i> parameter is ignored. |
| PICKAP_REC     | Use <i>psizlPick</i> value.                                           |

*psizlPick* Points to the **SIZEL** structure that contains the pick-aperture size. The **SIZEL** structure has the following form:

```

typedef struct _SIZEL {
 LONG cx;
 LONG cy;
} SIZEL;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

### Return Value

The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

### Errors

Use the **WinGetLastError** function to retrieve the error value, which may be the following:

**PMERR\_INV\_PICK\_APERTURE\_OPTION**

### See Also

**GpiQueryPickApertureSize**, **GpiSetPickAperturePosition**

## ■ GpiSetPS

```

BOOL GpiSetPS (hps, psizl, flOptions)
HPS hps; /* presentation-space handle */
PSIZEL psizl; /* pointer to structure for presentation-space size */
ULONG flOptions; /* options */

```

The **GpiSetPS** function sets the page size and units for the presentation space. The function is often used to change the device transformation for the presentation space.

The function does not affect the device context associated with the presentation space. This means the device context that was already associated remains associated. Also, the function does not change the presentation space type, such as micro-presentation space or a normal presentation space.



When this function is called, it resets the presentation space to a state that is equivalent to setting the value `GRES_ALL` in the `GpiResetPS` function.

### Parameters

*hps* Identifies the presentation space.

*psizl* Points to the `SIZEL` structure that contains the size of the presentation page. The `SIZEL` structure has the following form:

```
typedef struct _SIZEL {
 LONG cx;
 LONG cy;
} SIZEL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

*flOptions* Specifies the presentation-space options. The options define the page unit for the presentation space. Although the *flOptions* parameter can include many other options (as specified by the `GpiCreatePS` function), the function ignores all but the page units. The page units can be one of the following values:

| Page unit                 | Meaning                                                                                                                     |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <code>PU_ARBITRARY</code> | Sets units initially to pels but permits the units to be modified later using the <code>GpiSetPageViewport</code> function. |
| <code>PU_HIENGLISH</code> | Sets units to 0.001 inch.                                                                                                   |
| <code>PU_HIMETRIC</code>  | Sets units to 0.01 millimeter.                                                                                              |
| <code>PU_LOENGLISH</code> | Sets units to 0.01 inch.                                                                                                    |
| <code>PU_LOMETRIC</code>  | Sets units to 0.1 millimeter.                                                                                               |
| <code>PU_PELS</code>      | Sets units to pels.                                                                                                         |
| <code>PU_TWIPS</code>     | Sets units to 1/1440 inch (1/20 point).                                                                                     |

### Return Value

The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

### See Also

`GpiCreatePS`, `GpiResetPS`

## ■ GpiSetRegion

**BOOL** `GpiSetRegion`(*hps*, *hrgn*, *crcl*, *arcl*)

**HPS** *hps*; /\* presentation-space handle \*/  
**HRGN** *hrgn*; /\* region handle \*/  
**LONG** *crcl*; /\* number of rectangles \*/  
**PRECTL** *arcl*; /\* pointer to array of rectangle structures \*/

The `GpiSetRegion` function redefines the region specified by the *hrgn* parameter. The function replaces the old region by creating a region that consists of the union of the rectangles pointed to by the *arcl* parameter. This function is similar to the `GpiCreateRegion` function.

The function can be used only if a device context is associated with the presentation space.

### Parameters

*hps* Identifies the presentation space.

*hrgn* Identifies the region handle.

*crcl* Specifies the number of rectangles to use to create the new region. If it is zero, the function creates an empty region and the *arcl* parameter is ignored.

*arcl* Points to the array of **RECTL** structures that contains the rectangles for the replacement region. The array must have the number of elements specified by *crcl*. The **RECTL** structures have the form:

```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is **GPI\_OK** if the function is successful or **GPI\_ERROR** if an error occurred.

**See Also** **GpiCombineRegion**, **GpiCreateRegion**

## ■ GpiSetSegmentAttrs

**BOOL** GpiSetSegmentAttrs (*hps*, *idSegment*, *flAttribute*, *flAttrFlag*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *idSegment*; /\* segment identifier \*/  
**LONG** *flAttribute*; /\* attributes \*/  
**LONG** *flAttrFlag*; /\* attribute on/off flag \*/

The **GpiSetSegmentAttrs** function sets a segment attribute for a retained segment. The segment attributes specify whether a segment is chained, visible, detectable, dynamic, and so on. The function can turn these attributes on or off, one attribute at a time.

**Parameters** *hps* Identifies the presentation space.

*idSegment* Specifies the segment to change. It must be greater than zero.

*flAttribute* Specifies the segment attribute to set or clear. It can be one of the following values:

| Value                       | Meaning                 |
|-----------------------------|-------------------------|
| <b>ATTR_CHAINED</b>         | Chained                 |
| <b>ATTR_DETECTABLE</b>      | Detectability           |
| <b>ATTR_DYNAMIC</b>         | Dynamic                 |
| <b>ATTR_FASTCHAIN</b>       | Fast chaining           |
| <b>ATTR_PROP_DETECTABLE</b> | Propagate detectability |
| <b>ATTR_PROP_VISIBLE</b>    | Propagate visibility    |
| <b>ATTR_VISIBLE</b>         | Visibility              |

*flAttrFlag* Specifies whether to turn the attribute on or off. If it is **ATTR\_ON**, the function turns on the attribute; if **ATTR\_OFF**, the function turns off the attribute.

- Return Value**      The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.
- Errors**            Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:
  - `PMERR_INV_MICROPS_FUNCTION`
  - `PMERR_INV_SEG_ATTR`
  - `PMERR_INV_SEG_NAME`
- Comments**        When a segment is modified from non-chained to chained, it is added to the end of the drawing chain.
- See Also**         `GpiQuerySegmentAttrs`

## ■ GpiSetSegmentPriority

**BOOL** `GpiSetSegmentPriority` (*hps, idSegment, idRefSegment, cmdOrder*)

**HPS** *hps*;                /\* presentation-space handle \*/  
**LONG** *idSegment*;        /\* segment identifier \*/  
**LONG** *idRefSegment*;     /\* reference-segment identifier \*/  
**LONG** *cmdOrder*;        /\* command options \*/

The `GpiSetSegmentPriority` function sets the priority for the segment specified by *idSegment*. Segment priority applies only to chained segments. The segment priority of a segment specifies the position of that segment in the picture chain. The priority affects how the segment appears when drawn, since segments with higher priorities (later positions in the chain) may draw over the segment.

The function changes a segment's priority by moving its position in the picture chain relative to a given segment. The function places the segment either before or after the segment specified by the *idRefSegment* parameter. The *cmdOrder* parameter specifies the priority that the segment should have relative to the *idRefSegment* segment, and therefore determines whether it goes before or after. The function places the segment at either the beginning or end of the picture chain if the *idRefSegment* parameter is zero.

- Parameters**      *hps*    Identifies the presentation space.
- idSegment*    Specifies the identifier of the segment whose priority is to change; it must be greater than zero.
- idRefSegment*    Specifies the reference-segment identifier. It must be the identifier of a segment in the picture chain, or it must be zero. If it is zero, the function uses the beginning or end of the picture chain.
- cmdOrder*    Specifies whether to give the segment higher or lower priority than the segment specified by *idRefSegment*. It can be one of the following values:

| Value                  | Meaning                                                                                                                                                                                                       |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>LOWER_PRI</code> | Gives the segment lower priority. The function places the segment before the reference-segment in the chain. If the <i>idRefSegment</i> is zero, the function makes the segment the highest-priority segment. |

|                     | Value                                                                                                                                                                                                                                         | Meaning                                                                                                                                                                                                      |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                     | HIGHER_PRI                                                                                                                                                                                                                                    | Gives the segment higher priority. The function places the segment after the reference-segment in the chain. If the <i>idRefSegment</i> is zero, the function makes the segment the lowest-priority segment. |
| <b>Return Value</b> | The return value is <code>GPL_OK</code> if the function is successful or <code>GPL_ERROR</code> if an error occurred.                                                                                                                         |                                                                                                                                                                                                              |
| <b>Errors</b>       | Use the <code>WinGetLastError</code> function to retrieve the error value, which may be one of the following:<br><br><code>PMERR_INV_MICROPS_FUNCTION</code><br><code>PMERR_INV_SEG_NAME</code><br><code>PMERR_SEG_AND_REFSEG_ARE_SAME</code> |                                                                                                                                                                                                              |
| <b>See Also</b>     | <code>GpiDrawChain</code> , <code>GpiDrawFrom</code> , <code>GpiQuerySegmentPriority</code>                                                                                                                                                   |                                                                                                                                                                                                              |

## ■ GpiSetSegmentTransformMatrix

**BOOL** GpiSetSegmentTransformMatrix(*hps*, *idSegment*, *cElements*, *pmatlf*, *flType*)

**HPS** *hps*; /\* presentation-space handle \*/  
**LONG** *idSegment*; /\* segment identifier \*/  
**LONG** *cElements*; /\* number of elements \*/  
**PMATRIXLF** *pmatlf*; /\* pointer to structure with transformation matrix \*/  
**LONG** *flType*; /\* type of transformation \*/

The `GpiSetSegmentTransformMatrix` function sets the segment transformation for the specified segment. The segment transformation applies to all primitives in the segment. The function sets the transformation by adding or replacing the existing transformation matrix with the matrix pointed to by the *pmatlf* parameter. The function adds, preempts, or replaces the existing transformation matrix as specified by the *flType* parameter.

The `GpiSetSegmentTransformMatrix` function requires a nine-element matrix to set the segment transformation. If the specified matrix does not contain nine elements, the function uses the corresponding elements of the identity matrix for each unspecified element. The *cElements* parameter specifies the number of elements in the matrix. If this parameter equals zero, the identity matrix is used. If scaling values greater than one are given, care must be taken that the combined effect of this and any other relevant transformations do not exceed the limit for fixed-point numbers.

### Parameters

*hps* Identifies the presentation space.

*idSegment* Specifies the segment identifier; it must be greater than zero. The segment transformation does not affect primitives outside the specified segment.

*cElements* Specifies the number of elements in the matrix to set. It can be any integer in the range 0 through 9.

*pmatlf* Points to a `MATRIXLF` structure that contains the transformation matrix. The `MATRIXLF` structure has the following form:

```
typedef struct _MATRIXLF {
 FIXED fxM11;
 FIXED fxM12;
 LONG lM13;
 FIXED fxM21;
 FIXED fxM22;
 LONG lM23;
 LONG lM31;
 LONG lM32;
 LONG lM33;
} MATRIXLF;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

*fType* Specifies how a specified matrix should be used to modify the segment transformation. It can be one of the following values:

| Value             | Meaning                                                                                                                                                                                                                                                 |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TRANSFORM_ADD     | Additive. The specified transformation matrix is combined with the existing segment transformation, in the order of the existing transformation first, the new transformation second. This option is useful for incremental updates to transformations. |
| TRANSFORM_PREEMPT | Preemptive. The specified transformation matrix is combined with the existing segment transformation, in the order of the new transformation first, the existing transformation second.                                                                 |
| TRANSFORM_REPLACE | New/replace. The previous segment transformation is discarded and replaced by the specified transformation matrix.                                                                                                                                      |

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```
PMERR_INV_MICROPS_FUNCTION
PMERR_INV_SEG_NAME
PMERR_INV_TRANSFORM_TYPE
```

**Comments** The system applies a segment transformation to the primitives in the given segment after applying the model transformation and any instance and segment transformations from calling segments.

**See Also** `GpiQueryDefaultViewMatrix`, `GpiQuerySegmentTransformMatrix`

## ■ GpiSetStopDraw

```
BOOL GpiSetStopDraw(hps, fStopDraw)
```

```
HPS hps; /* presentation-space handle */
```

```
LONG fStopDraw; /* stop-draw condition flag */
```

The `GpiSetStopDraw` function sets or clears the stop-draw condition. The stop-draw condition terminates specific functions that may be executing in another thread of the process. If the stop-draw condition is set, the system stops the following functions and forces each to return an error:

GpiDrawChain  
 GpiDrawDynamics  
 GpiDrawFrom  
 GpiDrawSegment  
 GpiPlayMetaFile  
 GpiPutData

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b>   | <i>hps</i> Identifies the presentation space.<br><i>fStopDraw</i> Specifies the stop-draw condition. If it is FALSE, the function clears the stop-draw condition. If it is TRUE, the function sets the condition.                                                                                                                                                                                                                       |
| <b>Return Value</b> | The return value is <code>GPI_OK</code> if the function is successful or <code>GPI_ERROR</code> if an error occurred.                                                                                                                                                                                                                                                                                                                   |
| <b>Errors</b>       | Use the <code>WinGetLastError</code> function to retrieve the error value, which may be one of the following:<br><code>PMERR_INV_MICROPS_FUNCTION</code><br><code>PMERR_INV_STOP_DRAW_VALUE</code>                                                                                                                                                                                                                                      |
| <b>Comments</b>     | The <code>GpiSetStopDraw</code> function lets an application set up and control an asynchronous thread that carries out long drawing operations. If the controlling thread needs to stop drawing, it sets the condition. If it wants to allow drawing, it clears the condition. The stop-draw condition only affects the listed functions.<br>Using the stop-draw condition to stop drawing to a metafile creates an unusable metafile. |
| <b>See Also</b>     | <code>GpiDrawChain</code> , <code>GpiDrawDynamics</code> , <code>GpiDrawFrom</code> , <code>GpiDrawSegment</code> , <code>GpiPlayMetaFile</code> , <code>GpiPutData</code>                                                                                                                                                                                                                                                              |

## ■ GpiSetTag

```
BOOL GpiSetTag(hps, lTag)
```

```
HPS hps; /* presentation-space handle */
```

```
LONG lTag; /* tag */
```

The `GpiSetTag` function sets the current primitive tag. A primitive tag is a way of identifying a primitive stored in a segment. The function sets the primitive tag and the system applies this tag to all subsequent primitives. The default tag is zero.

Primitive tags are typically used when correlating segments. The `GpiCorrelateChain`, `GpiCorrelateFrom`, and `GpiCorrelateSegment` functions return the segment identifier and the primitive tag of objects that lie in the pick aperture.

The `GpiSetTag` function cannot be used in an area bracket, but can be used before an area bracket to give all primitives in the area the same tag. If the attribute mode is `AM_PRESERVE`, the function saves the previous tag on the attribute stack when it sets the new tag. The previous tag can be retrieved using the `GpiPop` function.

|                   |                                                                                                            |
|-------------------|------------------------------------------------------------------------------------------------------------|
| <b>Parameters</b> | <i>hps</i> Identifies the presentation space.<br><i>lTag</i> Specifies a tag. It must be an integer value. |
|-------------------|------------------------------------------------------------------------------------------------------------|

|                     |                                                                                                                                                                                       |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Return Value</b> | The return value is <code>GPL_OK</code> if the function is successful or <code>GPL_ERROR</code> if an error occurred.                                                                 |
| <b>Errors</b>       | Use the <code>WinGetLastError</code> function to retrieve the error value, which may be the following:<br><code>PMERR_INV_MICROPS_FUNCTION</code>                                     |
| <b>Comment</b>      | Although primitives in an unnamed segment can be tagged, the correlation functions cannot pick or correlate unnamed segments or any tags applied to them are ignored.                 |
| <b>See Also</b>     | <code>GpiBeginArea</code> , <code>GpiCorrelateChain</code> , <code>GpiCorrelateFrom</code> , <code>GpiCorrelateSegment</code> , <code>GpiEndArea</code> , <code>GpiSetAttrMode</code> |

## ■ GpiSetViewingLimits

---

```

BOOL GpiSetViewingLimits(hps, pgrcLimits)
HPS hps; /* presentation-space handle */
PRECTL prcLimits; /* pointer to structure with viewing limits */

```

The `GpiSetViewingLimits` function sets the viewing limits. The viewing limits specify a rectangle in model space that the system uses to clip output. The viewing limits include all points inside the rectangle and all points on the left and bottom edges. It does not include points on the right and top edges. Points on these edges are clipped.

The `GpiSetViewingLimits` function can be used in a segment to set the viewing limits for subsequent primitives in the segment. The viewing limits also apply to any called segments, unless the called segment itself sets the viewing limits.

**Parameters** *hps* Identifies the presentation space.  
*prcLimits* Points to the `RECTL` structure that contains the coordinates of the viewing limits. The `RECTL` structure has the following form:

```

typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Comments** Unless the segments in the picture chain have the fast-chaining attribute, the system resets the default viewing limits when each segment in the chain is drawn. The default viewing limits include all model space, meaning nothing is clipped. The segment and model transformations do not affect the viewing limits, but the current viewing and default viewing transformations do.

**See Also** `GpiQueryViewingLimits`, `GpiSetAttrMode`

## ■ GpiSetViewingTransformMatrix

```

BOOL GpiSetViewingTransformMatrix(hps, cElements, pmatlf, flType)
HPS hps; /* presentation-space handle */
LONG cElements; /* number of elements */
PMATRIXLF pmatlf; /* pointer to structure with transformation matrix */
LONG flType; /* transformation type */

```

The `GpiSetViewingTransformMatrix` function sets the viewing transformation. The viewing transformation applies to all primitives inside subsequently opened (new) segments (it has no effect on primitives outside segments). All graphics primitives in a segment have the same viewing transformation, since the function cannot be used in an open segment. Also, once set for a segment, the viewing transformation cannot be altered.

The `GpiSetViewingTransformMatrix` function sets the transformation by replacing the existing transformation matrix with the matrix pointed to by the `pmatlf` parameter. The function replaces the existing transformation matrix as specified by the `flType` parameter. The function requires a nine-element matrix to set the viewing transformation. If the specified matrix does not contain nine elements, the function uses the corresponding elements of the identity matrix for each unspecified element. The `cElements` parameter specifies the number of elements in the matrix. If this parameter equals zero, the identity matrix is used. If scaling values greater than one are given, care must be taken that the combined effect of this and any other relevant transformations do not exceed the limit for fixed-point numbers.

### Parameters

*hps* Identifies the presentation space.

*cElements* Specifies the number of elements in the matrix to set. It can be any integer in the range 0 through 9.

*pmatlf* Points to the `MATRIXLF` structure that contains the transformation matrix. The `MATRIXLF` structure has the following form:

```

typedef struct _MATRIXLF {
 FIXED fxM11;
 FIXED fxM12;
 LONG lM13;
 FIXED fxM21;
 FIXED fxM22;
 LONG lM23;
 LONG lM31;
 LONG lM32;
 LONG lM33;
} MATRIXLF;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

*flType* Specifies the transform type. It can be `TRANSFORM_REPLACE`. The previous viewing transformation is discarded and replaced by a specified transformation.

### Return Value

The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.



**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

PMERR\_INV\_MICROPS\_FUNCTION  
PMERR\_INV\_TRANSFORM\_TYPE

**See Also** `GpiQueryDefaultViewMatrix`, `GpiQueryViewingTransformMatrix`

## ■ **GpiStrokePath**

---

**LONG GpiStrokePath**(*hps*, *lPath*, *flOptions*)

**HPS** *hps*; /\* presentation-space handle \*/

**LONG** *lPath*; /\* stroke path \*/

**ULONG** *flOptions*; /\* reserved \*/

The `GpiStrokePath` function strokes a path and then draws it using the area attributes. This function first converts the path to one describing the envelope of a wide line stroked using the current geometric wide-line attribute (see the `GpiSetLineWidthGeom` function).

**Parameters** *hps* Identifies a presentation space.

*lPath* Specifies the path to stroke.

*flOptions* Specifies a reserved value; it must be zero.

**Return Value** The return value is `GPL_OK` or `GPL_HITS` if the function is successful (it is `GPL_HITS` if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is `GPL_ERROR` if an error occurs.

**See Also** `GpiBeginPath`, `GpiEndPath`, `GpiModifyPath`, `GpiSetLineWidthGeom`

## ■ **GpiUnloadFonts**

---

**BOOL GpiUnloadFonts**(*hab*, *pszModName*)

**HAB** *hab*; /\* anchor-block handle \*/

**PSZ** *pszModName*; /\* pointer to the module name \*/

The `GpiUnloadFonts` function unloads font definitions that were previously loaded from the resource file specified by the *pszModName* parameter. Before unloading fonts, the application should delete any local identifiers previously assigned to the fonts. The function unloads the fonts for the application only. If any other applications have loaded the fonts, they remain available for those applications.

**Parameters** *hab* Identifies the anchor block.

*pszModName* Points to a null-terminated string. This string must be a valid MS OS/2 filename. If it does not specify a path and the filename extension, the function appends the default extension (*.dll*) and searches for the font resource file in the directories specified by the `libpath` command in the `config.sys` file.

**Return Value** The return value is `GPL_OK` if the function is successful or `GPL_ERROR` if an error occurred.

**Errors** Use the `WinGetLastError` function to retrieve the error value, which may be the following:

`PMERR_FONT_NOT_LOADED`

**See Also** `GpiCreateLogFont`, `GpiDeleteSetId`, `GpiLoadFonts`, `GpiSetCharSet`

## ■ GpiUnrealizeColorTable

**BOOL** `GpiUnrealizeColorTable` (*hps*)

**HPS** *hps*; /\* presentation-space handle \*/

The `GpiUnrealizeColorTable` function “unrealizes” the logical color table. The function has the opposite effect of the `GpiRealizeColorTable` function. It restores the default device colors for the physical palette for the device associated with the given presentation space.

The function does not change the logical color table.

**Parameters** *hps* Identifies the presentation space.

**Return Value** The return value is `GPI_OK` if the function is successful or `GPI_ERROR` if an error occurred.

**See Also** `GpiRealizeColorTable`

## ■ GpiWCBitBit

**LONG** `GpiWCBitBit` (*hps*, *hbm*, *cPoints*, *aptl*, *lRop*, *flOptions*)

**HPS** *hps*; /\* presentation-space handle \*/

**HBITMAP** *hbm*; /\* bitmap handle \*/

**LONG** *cPoints*; /\* number of points \*/

**PPOINTL** *aptl*; /\* pointer to structure with points \*/

**LONG** *lRop*; /\* mixing function \*/

**ULONG** *flOptions*; /\* options \*/

The `GpiWCBitBit` function copies a bitmap to a presentation space. It can also modify the bitmap within a rectangle in a presentation space. The exact operation carried out by `GpiWCBitBit` depends on the raster operation specified by the `lRop` parameter.

If `lRop` directs `GpiWCBitBit` to copy a bitmap, the function copies the bitmap specified by *hbm* to the presentation space. The presentation space must be associated with a device context for the display, for memory, or for some other suitable raster device. The *aptl* parameter points to an array of points that specify the corners of a rectangle in the bitmap as well as the corners of the rectangle in the presentation space to receive the bitmap. The bitmap rectangle is specified in device coordinates; the presentation-space rectangle in world coordinates. If the bitmap and presentation-space rectangles are not the same (after converting the presentation space to device coordinates), `GpiWCBitBit` stretches or compresses the bitmap to fit the presentation-space rectangle.

If *IRop* directs **GpiWCBitBit** to modify a bitmap, the function uses the raster operation to determine how to alter the bits in a rectangle in the presentation space. Raster operations include changes such as inverting existing bits, replacing bits with pattern bits, and mixing existing and pattern bits to create new colors. For some raster operations, the function mixes the bits of the bitmap with the presentation space and/or pattern bits.

## Parameters

*hps* Identifies the presentation space.

*hbm* Identifies the bitmap.

*cPoints* Specifies the number of points pointed to by the *aptl* parameter. It must be 4.

*aptl* Points to an array of **POINTL** structures that contains the number of points specified in the *cPoints* parameter. The points must be given in the following order:

| Element index | Coordinate                                                                      |
|---------------|---------------------------------------------------------------------------------|
| 0             | Specifies the lower-left corner of the target rectangle in world coordinates.   |
| 1             | Specifies the upper-right corner of the target rectangle in world coordinates.  |
| 3             | Specifies the lower-left corner of the source rectangle in device coordinates.  |
| 4             | Specifies the upper-right corner of the source rectangle in device coordinates. |

The **POINTL** structure has the following form:

```
typedef struct _POINTL {
 LONG x;
 LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

*IRop* Specifies the raster operation for the function. It can be any value in the range 0 through 255 or one of the following values, which represent common raster operations:

| Value           | Meaning                                                                                                  |
|-----------------|----------------------------------------------------------------------------------------------------------|
| ROP_DSTINVERT   | Inverts the target.                                                                                      |
| ROP_MERGECOPY   | Combines the source and the pattern using the bitwise AND operator.                                      |
| ROP_MERGEPAINT  | Combines the inverse of the source and the target using the bitwise OR operator.                         |
| ROP_NOTSRCCOPY  | Copies the inverse of the source to the target.                                                          |
| ROP_NOTSRCERASE | Combines the inverse of the source and the inverse of the target bitmaps using the bitwise AND operator. |
| ROP_ONE         | Sets all target pels to 1.                                                                               |
| ROP_PATCOPY     | Copies the pattern to the target.                                                                        |
| ROP_PATINVERT   | Combines the target and the pattern using the bitwise exclusive XOR operator.                            |

| Value         | Meaning                                                                                    |
|---------------|--------------------------------------------------------------------------------------------|
| ROP_PATPAINT  | Combines the inverse of the source, the pattern, and target using the bitwise OR operator. |
| ROP_SRCAND    | Combines the source and target bitmaps using the bitwise AND operator.                     |
| ROP_SRCCOPY   | Copies the source bitmap to the target.                                                    |
| ROP_SRCERASE  | Combines the source and the inverse of the target bitmaps using the bitwise AND operator.  |
| ROP_SRCINVERT | Combines the source and target bitmaps using the bitwise exclusive OR operator.            |
| ROP_SRCPAINT  | Combines the source and target bitmaps using the bitwise OR operator.                      |
| ROP_ZERO      | Sets all target pels to 0.                                                                 |

*fOptions* Specifies how to compress a bitmap if the target rectangle is smaller than the source. It can be one of the following values:

| Value      | Meaning                                                                                                                                                                                               |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BBO_AND    | Compresses two rows or columns into one by combining them with the bitwise AND operator. This value is useful for compressing bitmaps that have black images on a white background.                   |
| BBO_OR     | Compresses two rows or columns into one by combining them with the bitwise OR operator. This value is the default and is useful for compressing bitmaps that have white images on a black background. |
| BBO_IGNORE | Compresses two rows or columns by throwing one out. This value is useful for compressing color bitmaps.                                                                                               |

All values in the range 0x0100 to 0xFF00 are reserved for privately supported modes for particular devices.

### Return Value

The return value is `GPI_OK` or `GPI_HITS` if the function is successful (it is `GPI_HITS` if the detectable attribute is set for the presentation space and a correlation hit occurs). The return value is `GPI_ERROR` if an error occurs.

### Errors

Use the `WinGetLastError` function to retrieve the error value, which may be one of the following:

```

PMERR_BASE_ERROR
PMERR_BITMAP_NOT_SELECTED
PMERR_INCOMPATIBLE_BITMAP
PMERR_INV_BITBLT_MIX
PMERR_INV_BITBLT_STYLE
PMERR_INV_COORDINATE
PMERR_INV_DC_TYPE
PMERR_INV_HBITMAP
PMERR_INV_HDC
PMERR_INV_IN_AREA
PMERR_INV_IN_PATH
PMERR_INV_LENGTH_OR_COUNT

```

**Comments**

The `GpiWCBitBlt` function can be used in an open segment. If the drawing mode is `DM_DRAWANDRETAIN` or `DM_RETAIN`, the function builds a graphics order in the current open segment. The order identifies the bitmap handle and uses long or short coordinates, as determined by the presentation-space format.

`GpiWCBitBlt` does not affect the pels in the upper and right boundaries of the presentation-space rectangle. This means the function draws up to but does not include those pels. Also, the function ignores any rotation transformations.

If the `IRop` parameter includes a pattern, `GpiWCBitBlt` uses the current area color, area background color, pattern set, and pattern symbol of the presentation space. Although the function may stretch or compress the bitmap, it never stretches or compresses the pattern.

If the presentation-space and the bitmap have different color formats, `GpiWCBitBlt` converts the bitmap color format as it copies the bitmap. This applies to bitmaps copied to a device context having a monochrome format. To convert a monochrome bitmap to a color bitmap, `GpiWCBitBlt` converts 1 pels to the presentation foreground color, and 0 pels to the current-area background color.

**Example**

This example uses `GpiWCBitBlt` to copy and compress a bitmap in a presentation space. The function copies the bitmap that is 100 pels wide and 100 pels high into a 50-by-50-pel rectangle at the location (300,400). Since the raster operation is `ROP_SRCCOPY`, `GpiWCBitBlt` replaces the image previously in the presentation-space rectangle. The function compresses the bitmap to fit the new rectangle by discarding extra rows and columns as specified by the `BBO_IGNORE` option.

```
HPS hps;
HBITMAP hbm;
POINTL aptl[4] = {
 300, 400, /* lower-left corner of target */
 350, 450, /* upper-right corner of target */
 0, 0, /* lower-left corner of source */
 100, 100 }; /* upper-right corner of source */

GpiWCBitBlt(hps, /* presentation space */
 hbm, /* bitmap handle */
 4L, /* four points needed to compress */
 aptl, /* points for source and target rectangles */
 ROP_SRCCOPY, /* copy source replacing target */
 BBO_IGNORE); /* discard extra rows and columns */
```

**See Also**

`DevOpenDC`, `GpiBitBlt`, `GpiCreateBitmap`, `GpiLoadBitmap`, `GpiSetBitmap`, `GpiSetBitmapDimension`, `GpiSetBitmapId`

## ■ WinAddAtom

---

**ATOM** WinAddAtom(*hAtomTbl*, *pszAtomName*)  
**HATOMTBL** *hAtomTbl*; /\* handle to the atom table \*/  
**PSZ** *pszAtomName*; /\* address of the buffer for the atom name \*/

The **WinAddAtom** function adds an atom to an atom table and sets its use count to 1. If the atom name already exists in the table, this function adds 1 to its use count.

**Parameters**     *hAtomTbl*    Identifies the atom table. This handle must have been created by a previous call to the **WinCreateAtomTable** or **WinQuerySystemAtomTable** function.

*pszAtomName*    Points to the null-terminated character string that contains an atom name to be added to the table. If the string begins with a “#” character, the ASCII digits that follow are converted into a 16-bit integer. If this integer is a valid integer atom, this function returns that atom without actually modifying the atom table. If the string begins with an “!” character, the next two bytes are interpreted as an atom.

If the high word of *pszAtomName* is 0xFFFF, the low word is treated as an atom. If it is an integer atom, that atom is returned. Otherwise the reference count associated with that atom is increased by one and the atom is returned.

**Return Value**    The return value is the atom associated with the passed string or NULL. The return value is zero if an invalid atom table or atom name was specified.

**See Also**        **WinCreateAtomTable**, **WinDeleteAtom**, **WinFindAtom**, **WinQueryAtomLength**, **WinQueryAtomName**, **WinQueryAtomUsage**

## ■ WinAddProgram

---

**HPROGRAM** WinAddProgram(*hab*, *ppib*, *hGroupHandle*)  
**HAB** *hab*; /\* handle to the anchor block \*/  
**PPIBSTRUCT** *ppib*; /\* address of structure with program information \*/  
**HPROGRAM** *hGroupHandle*; /\* handle of the program group \*/

The **WinAddProgram** function adds a program title to the program list of a group. Program titles need not be unique, although duplicate titles within the same group are not allowed.

**Parameters**     *hab*        Identifies the anchor block.  
*ppib*        Points to a **PIBSTRUCT** structure that contains program information for the program being added to the program list.

The **PIBSTRUCT** structure has the following form:

```

typedef struct _PIBSTRUCT {
 PROGTYPE prog;
 CHAR szTitle[MAXNAMEL+1];
 CHAR szIconFileName[MAXPATHL+1];
 CHAR szExecutable[MAXPATHL+1];
 CHAR szStartupDir[MAXPATHL+1];
 XYWINSIZE xywinInitial;
 USHORT res1;
 LHANDLE res2;
 USHORT cchEnvironmentVars;
 PCH pchEnvironmentVars;
 USHORT cchProgramParameter;
 PCH pchProgramParameter;
} PIBSTRUCT;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

*hGroupHandle* Identifies the program group to which the program is added. The special value `SGH_ROOT` may be used, indicating the root group.

**Return Value** The return value is the program handle for the program added to the program list.

**See Also** `WinCreateGroup`, `WinQueryDefinition`, `WinQueryProgramTitles`

## ■ WinAddSwitchEntry

`HSWITCH WinAddSwitchEntry(pswctl)`

`PSWCNTRL pswctl`; /\* address of structure with new entry information \*/

The `WinAddSwitchEntry` function adds an entry to the switch list (the list of running programs that is displayed by the Task Manager).

**Parameters** *pswctl* Points to the `SWCNTRL` structure that contains information about the new switch-list entry. If the `szSwtitle` field in the `SWCNTRL` structure is `NULL`, the system uses the name under which the application was started. This applies only for the first call to this function for that program (since the program was started). Otherwise, a `NULL` entry name is invalid.

The `SWCNTRL` structure has the following form:

```

typedef struct _SWCNTRL {
 HWND hwnd;
 HWND hwndIcon;
 HPROGRAM hprog;
 USHORT idProcess;
 USHORT idSession;
 UCHAR uchVisibility;
 UCHAR fbJump;
 CHAR szSwtitle[MAXNAMEL+1];
 BYTE fReserved;
} SWCNTRL;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

**Return Value** The return value is a handle to the new switch-list entry, or `NULL` if an error occurs.

**Example** This example calls `WinQueryWindowProcess` to get the current processor identifier (needed for the `SWCNTRL` structure). It then sets up the *swctl* structure and calls `WinAddSwitchEntry` to add the program’s name to the task list. The returned handle can be used in subsequent calls to `WinChangeSwitchEntry` if the title needs to be changed. The variables *swctl*, *hswitch*, and *pid* should be

global if your application will be calling the `WinChangeSwitchEntry` function to avoid having to set up the structure again.

```
SWCCTRL swctl;
HSWITCH hswitch;
PID pid;

WinQueryWindowProcess(hwndFrame, &pid, NULL);

swctl.hwnd = hwndFrame; /* window handle */
swctl.hwndIcon = NULL; /* icon handle */
swctl.hprog = NULL; /* program handle */
swctl.idProcess = pid; /* process identifier */
swctl.idSession = NULL; /* session identifier */
swctl.uchVisibility = SWL_VISIBLE; /* visibility */
swctl.fbJump = SWL_JUMPABLE; /* jump indicator */
swctl.szSwtitle[0] = NULL; /* program name */

hswitch = WinAddSwitchEntry(&swctl);
```

**Comment** Leading and trailing blanks are removed from the title. The title is truncated to 60 characters. There is a system limit to the number of switch-list entries (several hundred) but it is unlikely to be reached because other system limits, such as memory size, will impinge first.

**See Also** `WinChangeSwitchEntry`, `WinQueryWindowProcess`, `WinRemoveSwitchEntry`

## ■ WinAlarm

```
BOOL WinAlarm(hwndDesktop, fsType)
HWND hwndDesktop; /* handle of the desktop */
USHORT fsType; /* alarm style */
```

The `WinAlarm` function generates an audible alarm that can be used to alert the user about special conditions.

**Parameters** *hwndDesktop* Identifies the desktop window. This parameter can be `HWND_DESKTOP` or the desktop window handle.  
*fsType* Specifies the alarm style. It can be one of the following values:

```
WA_WARNING
WA_NOTE
WA_ERROR
```

**Return Value** The return value is `TRUE` if the function is successful or `FALSE` if an error occurs.

**Comments** The duration and frequency of the alarm can be changed by the `WinSetSysValue` function. The alarm is not generated if the system value `SV_ALARM` is set to `FALSE`.

The following system values control the alarm:

| Value                         | Meaning                                                                                                                   |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <code>SV_ALARM</code>         | Specifies whether calls to <code>WinAlarm</code> generate a sound. A value of <code>TRUE</code> means sound is generated. |
| <code>SV_ERRORDURATION</code> | Specifies the duration in milliseconds of a <code>WA_ERROR</code> sound.                                                  |



| Value              | Meaning                                                             |
|--------------------|---------------------------------------------------------------------|
| SV_ERRORFREQ       | Specifies the frequency in cycles per second of a WA_ERROR sound.   |
| SV_NOTEDURATION    | Specifies the duration in milliseconds of a WA_NOTE sound.          |
| SV_NOTEFREQ        | Specifies the frequency in cycles per second of a WA_NOTE sound.    |
| SV_WARNINGDURATION | Specifies the duration in milliseconds of a WA_WARNING sound.       |
| SV_WARNINGFREQ     | Specifies the frequency in cycles per second of a WA_WARNING sound. |

**Example**

This example calls an application-defined initialization function, and if the function fails it calls `WinAlarm` to generate an audible alarm notifying the user the initialization failed.

```
if (!GenericInit()) { /* general initialization */
 WinAlarm(HWND_DESKTOP, WA_ERROR);
}
```

**See Also**

`WinFlashWindow`, `WinSetSysValue`

## ■ WinAllocMem

---

`NPBYTE WinAllocMem(hHeap, cb)`

`HHEAP hHeap;` /\* handle of the heap \*/

`USHORT cb;` /\* number of bytes to allocate \*/

The `WinAllocMem` function allocates memory from a heap and returns the 16-bit offset from the start of the segment that contains the heap.

**Parameters**

*hHeap* Identifies the heap from which to allocate memory. This handle is returned by a previous call to the `WinCreateHeap` function.

*cb* Specifies the number of bytes to allocate.

**Return Value**

The return value points to the allocated memory block if the function is successful or is `NULL` if an error occurs.

**Comments**

The low two bits of the returned pointer are always zero. This function returns `NULL` when it cannot allocate the memory object, either because an invalid heap handle is specified or because 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.

If the specified heap is created with the `HM_MOVEABLE` option, the value of the *cb* parameter is remembered in the second reserved word of the allocated block.

If no free block is found, `WinAllocMem` seeks space by calling the `WinAvailMem` function. If this does not generate a free block large enough, `WinAllocMem` attempts to grow the segment by the maximum of the size of the request and the minimum growth parameter specified on the call to the `WinCreateHeap` function. If that fails, this function returns `NULL`.

No synchronization is done for this call. Multi-threaded applications should use semaphores if more than one thread will be making this call to prevent two or more threads from calling this function at the same time.

If the heap was created in the default data segment, the returned value may be used directly as a near pointer. Otherwise, it must be combined with the selector of the heap segment to create a far pointer. The MAKEP macro can be used for this purpose as in the following code fragment:

```
NPBYTE npb;
PBYTE pb;

npb = WinAllocMem(...);
pb = MAKEP(sel, /* heap segment selector */
 (USHORT) npb); /* value returned from WinAllocMem */
```

**See Also** WinAvailMem, WinCreateHeap, WinFreeMem, WinReallocMem

## ■ WinAvailMem

**USHORT WinAvailMem**(*hHeap*, *fCompact*, *cbMinFree*)

**HHEAP** *hHeap*; /\* handle of the heap \*/

**BOOL** *fCompact*; /\* memory-compaction flag \*/

**USHORT** *cbMinFree*; /\* amount of storage requested \*/

The WinAvailMem function returns the size of the largest free block on the heap.

**Parameters** *hHeap* Identifies the heap. This handle must have been created by a previous call to the WinCreateHeap function.

*fCompact* Specifies the memory-compaction flag. If TRUE, the heap is reorganized. If FALSE, the heap is not reorganized.

*cbMinFree* This parameter is currently not used, but should be set to the amount of storage the caller is requesting. A future release may use this value to limit the scope of the compaction.

**Return Value** The return value is the largest memory block available if the function is successful, or 0xFFFF if an error occurs.

**Comments** If the passed heap was created with the HM\_MOVEABLE option, the reorganization entails moving all movable blocks toward the beginning of the heap. The presence of fixed objects may inhibit the amount of movement that can occur. While the compaction is occurring, the dedicated and nondedicated free lists are reconstructed from any free blocks that cannot be filled by the compactor.

If the passed heap was not created with the HM\_MOVEABLE option, the reorganization entails sorting all free lists into a single list in address order, scanning the list for adjacent blocks to coalesce, and then reconstructing the dedicated and nondedicated free lists.

No synchronization is done for this call. Multi-threaded applications should use semaphores if more than one thread will be making this call to prevent two or more threads from calling this function at the same time.

**See Also** WinAllocMem, WinCreateHeap, WinFreeMem, WinReallocMem

## ■ WinBeginEnumWindows

---

HENUM WinBeginEnumWindows(*hwnd*)

HWND *hwnd*; /\* handle of the parent window \*/

The **WinBeginEnumWindows** function begins the enumeration process for all immediate child windows of a specified window. This function takes a snapshot of the window hierarchy at the time the function is called and enumerates the windows in the order they were at the time the snapshot is taken. The topmost child window is enumerated first, guaranteeing that all child windows are enumerated.

**Parameters** *hwnd* Identifies the parent window whose child windows are enumerated. If this parameter is **HWND\_DESKTOP**, all main windows are enumerated.

**Return Value** The return value is the handle to an enumeration list and is used in subsequent calls to **WinGetNextWindow** to retrieve the window handles in succession. When the application has finished the enumeration, this handle must be destroyed by the **WinEndEnumWindows** function.

**Comments** The enumerated windows are not locked and can be destroyed between the time this function is called and the time that the **WinGetNextWindow** function is used to obtain the window handle. However, enumerated window handles referenced by the enumeration handle will be removed from the enumeration list if they are destroyed. Thus they will not be returned by **WinGetNextWindow**.

**See Also** **WinEndEnumWindows**, **WinGetNextWindow**

## ■ WinBeginPaint

---

HPS WinBeginPaint(*hwnd*, *hps*, *prclPaint*)

HWND *hwnd*; /\* handle of the window handle \*/

HPS *hps*; /\* handle of the presentation space \*/

PRECTL *prclPaint*; /\* address of structure for bounding rectangle \*/

The **WinBeginPaint** function obtains a presentation space whose visible region is the window's update region. This sets up the presentation space so that any drawing will only occur within the update region. The presentation space can be an existing one supplied to this function, in which case its visible region will be set to the update region of *hwnd*. Otherwise, a cached presentation space is obtained specifically for the window.

The **WinEndPaint** function must be called when drawing is complete.

**Parameters** *hwnd* Identifies the window where drawing will occur.

*hps* Identifies the presentation space to use. If this parameter is **NULL**, a cache presentation space is created.

*prclPaint* Points to a **RECTL** structure that will be set to the smallest rectangle bounding the update region. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
 LONG xLeft;
 LONG yBottom;
 LONG xRight;
 LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

- Return Value** The return value is a handle to a presentation space, or NULL if an error occurred.
- Comments** This function hides the cursor, if it is in the window, until the `WinEndPaint` function is called.
- Example** This example calls `WinBeginPaint` when it receives a `WM_PAINT` message to get a presentation-space handle to the update region, and to get the coordinates of the update rectangle. It then fills the update rectangle and calls `WinEndPaint` to release the presentation space.
- ```
case WM_PAINT:
    hps = WinBeginPaint(hwnd, /* handle of the window */
                        NULL, /* get a cache ps */
                        &rcl); /* receives update rectangle */
    WinFillRect(hps, &rcl, CLR_WHITE);
    WinEndPaint(hps);
```
- See Also** `WinEndPaint`

■ WinBroadcastMsg

```
BOOL WinBroadcastMsg(hwnd, msg, mp1, mp2, fs)
HWND hwnd; /* handle of the parent window */
USHORT msg; /* message */
MPARAM mp1; /* message parameter */
MPARAM mp2; /* message parameter */
USHORT fs; /* windows to send message to */
```

The `WinBroadcastMsg` function broadcasts a message to multiple windows. This function sends or posts a message to all immediate child windows of the specified window.

- Parameters**
- hwnd* Identifies the window whose immediate child windows will receive the message. If this parameter is NULL, the function sends the message to all main windows on the screen.
- msg* Specifies the message.
- mp1* Specifies the first message parameter.
- mp2* Specifies the second message parameter.
- fs* Specifies which windows to send the message to, and whether the message should be sent or posted. The value consists of a flag from each of the following lists combined using the OR operator.

The following list contains the values specifying which windows to broadcast the message to:

Destination	Meaning
BMSG_DESCENDANTS	Post or send the message to <i>hwnd</i> and all of its descendants.
BMSG_FRAMEONLY	Post or send the message to frame windows only.

The following list contains the values specifying how to broadcast the message (send or post):

Value	Meaning
BMSG_POST	Post a message to all child windows of the window specified by the <i>hwnd</i> parameter.
BMSG_POSTQUEUE	Post a message to all threads that have a message queue. The message's <i>hwnd</i> parameter will be NULL.
BMSG_SEND	Send a message to all children of the window specified by the <i>hwnd</i> parameter.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinPostMsg, WinSendMsg

■ WinCalcFrameRect

```

BOOL WinCalcFrameRect(hwndFrame, prcl, fClient)
HWND hwndFrame; /* handle of the frame window */
PRECTL prcl; /* address of structure with window rectangle */
BOOL fClient; /* client-indicator flag */

```

The **WinCalcFrameRect** function calculates a client rectangle from a frame rectangle or calculates a frame rectangle from a client rectangle. This function provides the size and position of the client area within 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.

Parameters *hwndFrame* Identifies the frame window.

prcl Points to the **RECTL** structure that contains the coordinates of the window. If the *fClient* parameter is TRUE, this structure contains the coordinates of the frame window, and on return, it will contain the coordinates of a client window. If the *fClient* parameter is FALSE, this structure contains the coordinates of the client window, and on return, it will contain the coordinates of a frame window. The **RECTL** structure has the following form:

```

typedef struct _RECTL {
    LONG xLeft;
    LONG yBottom;
    LONG xRight;
    LONG yTop;
} RECTL;

```

For a full description, see Chapter 4, "Types, Macros, Structures."

fClient Specifies whether the window to calculate is a client window or a frame window. If TRUE, a client window is calculated. If FALSE, a frame window is calculated.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

■ WinCallMsgFilter

```

BOOL WinCallMsgFilter(hab, pqmsg, msgf)
HAB hab;           /* handle of the anchor block      */
PQMSG pqmsg;      /* address of structure with message */
USHORT msgf;      /* message-filter code              */

```

The **WinCallMsgFilter** function calls a message-filter hook. This function allows an application to pass a message to the message-filter hook procedure(s).

Parameters *hab* Identifies the anchor block.

pqmsg Points to the **QMSG** structure that contains the message to be passed to the message-filter hook procedure. The **QMSG** structure has the following form:

```

typedef struct _QMSG {
    HWND   hwnd;
    USHORT msg;
    MPARAM mp1;
    MPARAM mp2;
    ULONG  time;
    POINTL pt1;
} QMSG;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

msgf Specifies the message-filter code passed to the message-filter hook. This can be one of the following values:

Message	Meaning
MSGF_DIALOGBOX	Message originated while processing a modal dialog.
MSGF_MESSAGEBOX	Message originated while processing a message box.
MSGF_TRACK	Message originated while tracking a control, such as a scroll bar.

Return Value The return value is TRUE if a message-filter hook returned TRUE. Otherwise, it is FALSE, indicating that all message-filter hooks returned FALSE or that no message-filter hooks are defined.

See Also WinDispatchMsg, WinGetMsg

■ WinCancelShutdown

BOOL WinCancelShutdown(*hmq*, *fCancelAlways*)

HMQ *hmq*; /* handle of the message queue */

BOOL *fCancelAlways*; /* status of WM_QUIT message */

The **WinCancelShutdown** function allows a thread to function after receiving a **WM_QUIT** message. The thread must call the **WinCancelShutdown** function, passing its message-queue handle for identification. If the thread owns the active window, that window is pushed to the bottom and deactivated. **WinCancelShutdown** maintains a list of queues associated with the threads that called it to avoid sending another **WM_QUIT** message to the same thread later.

Parameters *hmq* Identifies the message queue for the current thread.

fCancelAlways Specifies whether the thread receives **WM_QUIT** messages during system shutdown. If this parameter is **TRUE**, the thread does not receive **WM_QUIT** messages during system shutdown. Note that this does not prevent a **WM_QUIT** message from being put into the queue for this thread by some other mechanism. If this parameter is **FALSE**, the thread ignores the **WM_QUIT** message. Note that a subsequent system shutdown causes a new **WM_QUIT** message to be sent to this thread.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

■ WinCatch

SHORT WinCatch(*pcatchbuf*)

PCATCHBUF *pcatchbuf*; /* address of structure for execution environment */

The **WinCatch** function captures the current execution environment and copies it to a buffer. The buffer can later be used by the **WinThrow** function to restore the execution environment. The execution environment includes the state of all system registers and the instruction counter.

Parameters *pcatchbuf* Points to the **CATCHBUF** structure that receives the execution environment. The **CATCHBUF** structure has the following form:

```
typedef struct _CATCHBUF {
    ULONG reserved[4];
} CATCHBUF;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

Return Value The **WinCatch** function returns immediately with a return value of zero. It returns again when the **WinThrow** function is called, this time with the return value specified in the *sErrorReturn* parameter of the **WinThrow** function.

Comments The routine that calls **WinCatch** is responsible for freeing any resources allocated between the time **WinCatch** was called and the time **WinThrow** was called.

Example This example calls **WinCatch** to save the current execution environment before calling a recursive sort function. The first return from **WinCatch** is zero. If the *doSort* function calls **WinThrow**, execution will again return to the **WinCatch** function. This time, **WinCatch** will return the **STACKOVERFLOW** error

passed by the *doSort* function. The *doSort* function is recursive, that is, it calls itself. It maintains a variable, *usStackCheck*, that is used to check to see how much stack space has been used. If more than 3K of the stack has been used, it calls *WinThrow* to drop out of all the nested function calls back into the function that called *WinCatch*.

```
USHORT usStackCheck
CATCHBUF ctchbf;

main() {
    SHORT sErrorReturn;

    sErrorReturn = WinCatch(&ctchbf); /* save execution environment */
    if (sErrorReturn) {
        . /* error processing */
    }
    usStackCheck = 0; /* initialize stack usage count */
    doSort(1, 1000); /* call sort function */
}

VOID doSort(sLeft, sRight)
SHORT sLeft, sRight;
{
    SHORT i, sLast;

    /*
     * check to see if more than 3K of the stack has been used, and if
     * so, call WinThrow to drop back into the original calling program
     */

    usStackCheck += 10;
    if (usStackCheck > (3 * 1024))
        WinThrow(&ctchbf, STACKOVERFLOW);

    . /* sorting algorithm */
    doSort(sLeft, sLast - 1); /* note recursive call */
    usStackCheck -= 10; /* update stack check variable */
}
```

See Also WinThrow

■ WinChangeSwitchEntry

```
USHORT WinChangeSwitchEntry(hSwitch, pswctl)
HSWITCH hSwitch; /* handle to task-switch list */
PSWCNTRL pswctl; /* address of structure with change information */
```

The *WinChangeSwitchEntry* function changes information in the switch list (the list of running programs displayed by the Task Manager).

Parameters *hSwitch* Identifies the switch-list entry to change. This handle is returned by the *WinAddSwitchEntry* function.

pswctl Points to the *SWCNTRL* structure that contains information about the changed switch-list entry. The *SWCNTRL* structure has the following form:


```
typedef struct _SWCNTRL {
    HWND      hwnd;
    HWND      hwndIcon;
    HPROGRAM  hprog;
    USHORT    idProcess;
    USHORT    idSession;
    UCHAR     uchVisibility;
    UCHAR     fbJump;
    CHAR      szSwtitle[MAXNAMEL+1];
    BYTE      fReserved;
} SWCNTRL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is zero if the function is successful or nonzero if an error occurs.

Example This example changes a *swctl* structure (SWCNTRL) and then calls **WinChangeSwitchEntry** to change the title of the application in the task list. For this example to work, the *swctl* structure must have been a global that was initialized prior to this call. See the example for the **WinAddSwitchEntry** function to see how to set up a SWCNTRL structure.

```
strcpy(swctl.szSwtitle, "Generic: "); /* application name */
strcat(swctl.szSwtitle, pszFileName); /* current filename */
WinChangeSwitchEntry(hswitch, &swctl); /* make the change */
```

See Also WinAddSwitchEntry, WinRemoveSwitchEntry

■ WinCloseClipbrd

BOOL WinCloseClipbrd(*hab*)

HAB *hab*; /* handle of the anchor block */

The **WinCloseClipbrd** function closes the clipboard, allowing other applications to open it. This function sends a WM_DRAWCLIPBOARD message, causing the clipboard contents to be drawn in the clipboard-viewer window. The clipboard must be open prior to this function being called.

Parameters *hab* Identifies the anchor block.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinEmptyClipbrd, WinOpenClipbrd

■ WinCompareStrings

USHORT WinCompareStrings(*hab, idcp, idcc, psz1, psz2, reserved*)

HAB *hab*; /* handle of the anchor block */
USHORT *idcp*; /* code-page identifier */
USHORT *idcc*; /* country-code identifier */
PSZ *psz1*; /* address of first string */
PSZ *psz2*; /* address of second string */
USHORT *reserved*; /* must be zero */

The **WinCompareStrings** function compares two strings.

Parameters

hab Identifies the anchor block.
idcp Identifies the code page.
idcc Identifies the country code.
psz1 Points to the first string.
psz2 Points to the second string.
reserved Specifies a reserved value; must be zero.

Return Value The return value is the comparison result. It can be one of the following values:

Value	Meaning
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	Invalid country-code or code-page identifier.

■ WinCopyAccelTable

USHORT WinCopyAccelTable (*haccel*, *pacct*, *cbCopyMax*)

HACCEL *haccel*; /* handle of the accelerator table */
PACCELTABLE *pacct*; /* address of structure receiving information */
USHORT *cbCopyMax*; /* maximum size of data area */

The **WinCopyAccelTable** function copies an accelerator table. This function is used to obtain the accelerator-table data that corresponds to an accelerator-table handle or to determine the size of the accelerator-table data.

Parameters

haccel Identifies the accelerator table.
pacct Points to the area of memory where the accelerator-table information will be copied (in the form of an **ACCELTABLE** structure). If this parameter is **NULL**, the function will return with the number of bytes needed to copy the table. The **ACCELTABLE** structure has the following form:

```
typedef struct _ACCELTABLE {
    USHORT cAccel;
    USHORT codepage;
    ACCEL aaccel[1];
} ACCELTABLE;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

cbCopyMax Specifies the maximum size of the data area pointed to by the *pacct* parameter. This parameter is ignored if *pacct* is **NULL**.

Return Value The return value is the amount of data copied or the length required for the complete accelerator table if the function was successful, or it is zero if an error occurs.

■ WinCopyRect

BOOL WinCopyRect(*hab*, *prclDst*, *prclSrc*)

HAB *hab*; /* handle of the anchor block */
PRECTL *prclDst*; /* address of structure with destination rectangle */
PRECTL *prclSrc*; /* address of structure with source rectangle */

The **WinCopyRect** function copies the coordinates of one rectangle to another.

Parameters *hab* Identifies the anchor block.

prclDst Points to the **RECTL** structure that receives a copy of the rectangle specified by the *prclSrc* parameter. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

prclSrc Points to the **RECTL** structure that specifies the source rectangle.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also **WinSetRectEmpty**

■ WinCpTranslateChar

UCHAR WinCpTranslateChar(*hab*, *cpSrc*, *chSrc*, *cpDst*)

HAB *hab*; /* handle of the anchor block */
USHORT *cpSrc*; /* code page of source string */
UCHAR *chSrc*; /* character to be translated. */
USHORT *cpDst*; /* code page of resultant string */

The **WinCpTranslateChar** function translates a character from one code page to another.

Parameters *hab* Identifies the anchor block.
cpSrc Specifies the code page of the source string.
chSrc Specifies the character to be translated.
cpDst Specifies the code page of the resultant string.

Return Value The return value is the translated character. If a match for this character is not found, 0xFF is returned.

See Also **WinCpTranslateString**

■ WinCpTranslateString

BOOL WinCpTranslateString (*hab, cpSrc, pszSrc, cpDst, cchDestMax, pszDest*)

HAB *hab*; /* handle of the anchor block */
USHORT *cpSrc*; /* code page of source string */
PSZ *pszSrc*; /* address of string to be translated */
USHORT *cpDst*; /* code page of resultant string */
USHORT *cchDestMax*; /* maximum length of output string */
PSZ *pszDest*; /* address of buffer for translated string */

The **WinCpTranslateString** function translates a string from one code page to another. Both source and destination strings are null terminated.

Parameters

hab Identifies the anchor block.
cpSrc Specifies the code page of the source string.
pszSrc Points to the string to be translated.
cpDst Specifies the code page of the resultant string.
cchDestMax Specifies the maximum length of the output string.
pszDest Points to the buffer to receive the translated string.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs. A return value of **TRUE** indicates that most, if not all, characters were translated successfully. All untranslated characters are converted to 0xFF.

See Also **WinCpTranslateChar**

■ WinCreateAccelTable

HACCEL WinCreateAccelTable (*hab, pacct*)

HAB *hab*; /* handle of the anchor block */
PACCELTABLE *pacct*; /* address of structure for accelerator table */

The **WinCreateAccelTable** function allocates an accelerator table with its contents initialized to that of the specified **ACCELTABLE** structure.

Parameters

hab Identifies the anchor block.
pacct Points to the **ACCELTABLE** structure that contains an accelerator table. The **ACCELTABLE** structure has the following form:

```
typedef struct _ACCELTABLE {
    USHORT cAccel;
    USHORT codepage;
    ACCEL aaccel[1];
} ACCELTABLE;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value identifies an accelerator table.

Comments When an application terminates it must call **WinDestroyAccelTable** to destroy every accelerator table created by the call to the **WinCreateAccelTable** function.

See Also **WinCopyAccelTable**, **WinDestroyAccelTable**, **WinQueryAccelTable**, **WinSetAccelTable**

■ WinCreateAtomTable

HATOMTBL WinCreateAtomTable (*cbInitial*, *cBuckets*)

USHORT *cbInitial*; /* number of bytes for atom table */
USHORT *cBuckets*; /* size of hash table */

The **WinCreateAtomTable** function creates an empty atom table. This function must be called before any other atom-manager function.

Parameters *cbInitial* Specifies the number of initial bytes 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 on the actual number of atoms stored in the table. If this parameter is zero, the size of the atom table is the minimum size needed to store the atom hash table.

cBuckets Specifies the size of the hash table used to access atoms. If this parameter is zero, the default value 37 is used. For best results, this parameter should be a prime number.

Return Value The return value is a handle to an atom table, or NULL if an error occurs.

See Also WinAddAtom, WinDeleteAtom, WinDestroyAtomTable, WinQuerySystemAtomTable

■ WinCreateCursor

BOOL WinCreateCursor (*hwnd*, *x*, *y*, *cx*, *cy*, *fs*, *prcClip*)

HWND *hwnd*; /* handle of the window with the cursor */
SHORT *x*; /* horizontal position of the cursor */
SHORT *y*; /* vertical position of the cursor */
SHORT *cx*; /* cursor width */
SHORT *cy*; /* cursor height */
USHORT *fs*; /* cursor appearance */
PRECTL *prcClip*; /* address of structure with cursor area */

The **WinCreateCursor** function creates a cursor for a specified window.

Parameters *hwnd* Identifies the window in which the cursor is displayed. This parameter can be the desktop window handle or **HWND_DESKTOP**.

x Specifies the horizontal position of the cursor.

y Specifies the vertical position of the cursor.

cx Specifies the width of the cursor. If this parameter is zero, the system border width (**SV_CXBORDER**) is used.

cy Specifies the height size of cursor. If this parameter is zero, the system border height (**SV_CYBORDER**) is used.

fs Specifies the appearance of the cursor. This parameter can be one of the following values:

Value	Meaning
CURSOR_FLASH	Cursor flashes.
CURSOR_FRAME	Cursor is a rectangular frame.
CURSOR_HALFTONE	Cursor is halftone.

Value	Meaning
CURSOR_SOLID	Cursor is solid.
CURSOR_SETPOS	Set a new cursor position. The <i>cx</i> and <i>cy</i> parameters are ignored. Used when a cursor has already been created. All other appearance flags are ignored.

prclClip Points to the **RECTL** structure that contains the coordinates of a rectangle within which the cursor is visible. If the cursor goes outside this rectangle, it becomes invisible (it is clipped). The rectangle is specified in window coordinates. If *prclClip* is **NULL**, the cursor is clipped to the window identified by the *hwnd* parameter. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

Comments The cursor is used to indicate the position of text input. It is initially hidden and must be made visible by using the **WinShowCursor** function.

Only one cursor is available at a time on a given screen. If another cursor exists when this function is called, it is destroyed. An application should create and display a cursor when it has the input focus or is the active window. Creating a cursor at any other time may stop the cursor from flashing in another window. Similarly, when the application loses the input focus or becomes inactive, it should destroy its cursor.

The cursor width is typically specified as zero (default width is used). This is preferable to a value of 1 because such a fine width will be almost invisible on a high-resolution device.

See Also **WinDestroyCursor**, **WinQueryCursorInfo**, **WinShowCursor**

■ WinCreateDlg

HWND WinCreateDlg(*hwndParent*, *hwndOwner*, *pfnDlgProc*, *pdlg*, *pCreateParams*)

HWND *hwndParent*; /* handle of the parent window */
HWND *hwndOwner*; /* handle of the owner window */
PFNWP *pfnDlgProc*; /* address of dialog procedure */
PDLGTEMPLATE *pdlg*; /* address of structure with dialog template */
VOID *pCreateParams*;

The **WinCreateDlg** function creates a dialog window from a dialog template in memory. This function works like the **WinLoadDlg** function, which creates a dialog window from a dialog template in a resource.

Parameters

hwndParent Identifies the parent window.

hwndOwner Identifies the owner window.

pfnDlgProc Points to the dialog procedure.

pdlgt Points to the **DLGTEMPLATE** structure that contains the dialog template. The **DLGTEMPLATE** structure has the following form:

```
typedef struct _DLGTEMPLATE {
    USHORT   cbTemplate;
    USHORT   type;
    USHORT   codepage;
    USHORT   offadlgti;
    USHORT   fsTemplateStatus;
    USHORT   iItemFocus;
    USHORT   coffPresParams;
    DLGITEM  adlgti[1];
} DLGTEMPLATE;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

pCreateParams Contains application-specific data that is passed to the dialog procedure as part of the **WM_INITDLG** message.

Return Value The return value is the handle of the dialog window that was created, or **NULL** if an error occurred.

Example This example loads a dialog template from the application’s resources and uses the template with the **WinCreateDlg** function to create a dialog window. This example is identical to calling the **WinLoadDlg** function, but gives the application the advantage of reviewing and modifying the dialog template before creating the dialog window.

```
SEL sel;
PDLGTEMPLATE pdlgt;

DosGetResource(NULL, RT_DIALOG, ID_DIALOG, &sel);
pdlgt = MAKEP(sel, 0); /* convert resource to structure pointer */

/* make any changes to dialog template here */

WinCreateDlg(HWND_DESKTOP,
    NULL, /* owner window */
    MyDlgProc, /* address of dialog procedure */
    pdlgt, /* address of dialog structure */
    NULL); /* application-specific data */
DosFreeSeg(sel); /* free the memory */
```

See Also **DosGetResource**, **WinDlgBox**, **WinLoadDlg**, **WinProcessDlg**

■ WinCreateFrameControls

```
BOOL WinCreateFrameControls(hwndFrame, pfcdata, pszTitle, hmod)
HWND hwndFrame; /* handle of the frame window */
PFRAMECDATA pfcdata; /* address of structure */
PSZ pszTitle; /* address of title-bar string */
HMODULE hmod; /* handle of module with the frame controls */
```

The **WinCreateFrameControls** function creates standard frame controls for a specified window. This function is used when the standard frame controls are

needed for a nonstandard window; for example, with a window with a class other than WC_FRAME.

Parameters *hwndFrame* Identifies the frame window that becomes the parent and owner window of all the frame controls created.

pfcdata Points to the FRAMECDATA structure that contains information about the frame controls that are to be created. The FRAMECDATA structure has the following form:

```
typedef struct _FRAMECDATA {
    USHORT      cb;
    ULONG       flCreateFlags;
    HMODULE     hmodResources;
    USHORT     idResources;
} FRAMECDATA;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

pszTitle Points to a null-terminated string displayed in a title-bar control.

hmod Identifies the module that contains the frame controls.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinCreateWindow

■ WinCreateGroup

HPROGRAM WinCreateGroup(*hab, pszTitle, fVisible, hprogDest, pszHelp*)

HAB <i>hab</i> ;	/* handle of the anchor block	*/
PSZ <i>pszTitle</i> ;	/* address of the group title	*/
BYTE <i>fVisible</i> ;	/* visibility flag	*/
HPROGRAM <i>hprogDest</i> ;	/* handle of the destination group	*/
PSZ <i>pszHelp</i> ;	/* address of help text	*/

The WinCreateGroup function creates a new program-group entry in the installed program list. The new group is created empty. The WinAddProgram function must be used to add program entries to the group. If the group already exists, the existing group handle is returned.

Parameters *hab* Identifies the anchor block.

pszTitle Points to the title of the new group. 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 a backslash (\).

fVisible Specifies the visibility of the new group. If it is SHE_VISIBLE, the group is visible (can be viewed by end-user). If it is SHE_INVISIBLE, the group is invisible.

hprogDest Identifies the program group into which the new group is placed. If this parameter is NULL, the new group is placed in the root group.

pszHelp Points to a null-terminated text string that is used as a short piece of help information relating to the new program group. This parameter is optional and can be NULL. If used, the string must contain at least one nonblank character and be less than 60 characters in length.

Return Value The return value is the group handle for the newly created group if the function is successful. Otherwise, the return value is NULL, indicating that an error occurred.

See Also WinAddProgram

■ WinCreateHeap

HHEAP WinCreateHeap (*selHeapBase, cbHeap, cbGrow, cbMinDed, cbMaxDed, fsOptions*)

```
USHORT selHeapBase; /* selector of the heap */
USHORT cbHeap; /* initial heap size */
USHORT cbGrow; /* number of bytes heap must grow */
USHORT cbMinDed; /* minimum number of dedicated free lists */
USHORT cbMaxDed; /* maximum number of dedicated free lists */
USHORT fsOptions; /* heap options */
```

The **WinCreateHeap** function creates a heap that can be used for memory management.

Parameters *selHeapBase* Specifies the selector of the segment that will contain the local heap.

cbHeap Specifies the initial heap size (in bytes).

cbGrow Specifies the minimum number of bytes by which the heap must be increased if it is too small to satisfy a memory allocation request (see the **WinAllocMem** and **WinReallocMem** functions). If this parameter is zero, the default value of 512 bytes is used.

cbMinDed Specifies the minimum number of dedicated free lists.

cbMaxDed Specifies the maximum number of dedicated free lists.

fsOptions Specifies the optional characteristics for the heap. It may be one or more of the following values:

Value	Meaning
HM_MOVEABLE	Specifies that the created heap supports movable objects. WinAllocMem then reserves an additional two words at the beginning of each allocated object.
HM_VALIDSIZE	Specifies that the heap manager should check the size arguments to WinReallocMem and WinFreeMem function calls against the two additional words stored at the beginning of each allocated object. This option is only valid if HM_MOVEABLE is also specified.

Return Value The return value is a handle to the heap, or zero if an error occurred.

Comments There are three types of segments that can contain a heap:

- Automatic data segment of an application
- Automatic data segment of a dynamic-link package
- Segment allocated by the **DosAllocSeg** function (public or shared)

To accommodate these various targets for heaps, all four possible combinations of the parameters are used to discriminate between the various options.

These combinations are shown in the following list:

selHeapBase	cbHeap	Meaning
Zero	Zero	Caller is an application that places the heap at the end of its automatic data segment. The size of the heap was specified with the HEAPSIZE keyword in the application's <i>.def</i> file to the linker. This function extracts the heap-size parameter from the local information segment 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 space at the end of the data segment, after the static data was loaded from the <i>.exe</i> file.
Selector	Nonzero	Caller is a dynamic-link package that places a heap at the end of its automatic data segment. The <i>cbHeap</i> parameter must be less than or equal to the HEAPSIZE value from the <i>.def</i> file that was passed to the dynamic-link package's initialization entry point in the <i>cx</i> register. Otherwise, this function may produce incorrect results.
Selector	Zero	Caller is either an application or a dynamic-link package that has explicitly allocated a segment using the DosAllocSeg function and places a heap in that segment. The heap is placed at the beginning of the segment and the size of the segment (determined by using DosSizeSeg) is the size of the heap.
Zero	Nonzero	Caller is either an application or a dynamic-link package that places a heap of a specific size in a separate segment but does not call DosAllocSeg . For information about accessing the segment implicitly allocated by WinCreateHeap when called with this combination of parameters, see the WinLockHeap function.

See Also

DosAllocSeg, **DosSizeSeg**, **WinAllocMem**, **WinDestroyHeap**, **WinFreeMem**, **WinLockHeap**, **WinReallocMem**

■ WinCreateMenu

HWND WinCreateMenu(*hwndParent*, *pvmt*)

HWND *hwndParent*; /* handle of the parent window */

VOID *pvmt*; /* address of the menu template */

The **WinCreateMenu** function creates a menu window from a menu template. The menu window is created with its parent and owner set to the *hwndParent* parameter.

Parameters *hwndParent* Identifies the owner and parent window which should be a frame window.

pvmt Points to the menu template. The format of the *pvmt* parameter is the binary menu-template-resource format.

Return Value The return value is the menu-window handle, or zero if an error occurred.

See Also WinCreateWindow, WinLoadMenu

■ WinCreateMsgQueue

HMQ WinCreateMsgQueue(*hab*, *cmsg*)

HAB *hab*; /* handle of the anchor block */

SHORT *cmsg*; /* size of the message queue */

The **WinCreateMsgQueue** function creates a message queue for the current thread. This function must be called after the **WinInitialize** function but before any other Presentation Manager functions are called. It can be called only once per thread.

Parameters *hab* Identifies the anchor block.

cmsg Specifies the maximum queue size. This parameter can use a value of **DEFAULT_QUEUE_SIZE** to get the system default queue size.

Return Value The return value identifies a message queue, or it is **NULL** if the queue cannot be created.

Comments The default queue size is 10 messages which is sufficient for most applications. However, if an application processes a high volume of messages, and the processing of some of these messages is slow, the application should create a larger queue.

Example This example shows the typical startup code for a thread that will be making Presentation Manager function calls; in this case the startup function of the application. It calls **WinInitialize** to initialize the thread for making Presentation Manager function calls, and **WinCreateMsgQueue** to create a message queue for the thread. Before the thread terminates, it calls **WinDestroyMsgQueue** to destroy the message queue.

```

HAB hab;      /* handle to the anchor block */
HMQ hmq;     /* handle to the message queue */

VOID cdecl main() {
    hab = WinInitialize(NULL);
    hmq = WinCreateMsgQueue(hab, DEFAULT_QUEUE_SIZE);
        /* initialization and message loop */
        :
    WinDestroyMsgQueue(hmq);
    WinTerminate(hab);
    DosExit(EXIT_PROCESS, 0);
}

```

See Also WinDestroyMsgQueue, WinInitialize

■ WinCreatePointer

HPOINTER WinCreatePointer(*hwndDesktop, hbmPointer, fPointer, xHotspot, yHotspot*)

HWND *hwndDesktop*; /* handle of the desktop -/
HBITMAP *hbmPointer*; /* handle of the bitmap -/
BOOL *fPointer*; /* full-size or icon-size pointer -/
SHORT *xHotspot*; /* horizontal hot spot -/
SHORT *yHotspot*; /* vertical hot spot -/

The **WinCreatePointer** function creates a pointer from a bitmap.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be **HWND_DESKTOP** or the desktop window handle.

hbmPointer Identifies the bitmap from which the pointer image is created. The bitmap must be of an even vertical size, logically divided into two sections in the vertical direction, each half representing one of the two images used as the successive drawing masks for the pointer. The top half is the XOR image. The bottom half is the AND image. This allows a pointer to have four colors: black, white, background, and inverted background.

fPointer Indicates if the pointer is pointer- or icon-sized. If this parameter is **TRUE**, the bitmap will be stretched (if necessary) to the system-pointer dimensions. If this parameter is **FALSE**, the bitmap will be stretched (if necessary) to the system-icon dimensions.

xHotspot Specifies the horizontal offset of the pointer hot spot from the lower-left corner (in pels).

yHotspot Specifies the vertical offset of the pointer hot spot from the lower-left corner (in pels).

Return Value The return value is the handle of the new pointer, or it is **NULL** if an error occurs.

Comments Since the bitmap needs to include two images, one on top of the other, the height of the bitmap must be an even number. This function will fail if the height is not an even number.

See Also WinDestroyPointer, WinDrawPointer, WinLoadPointer, WinQueryPointer, WinSetPointer

■ WinCreateStdWindow

HWND WinCreateStdWindow(*hwndParent*, *flStyle*, *pflCreateFlags*, *pszClientClass*, *pszTitle*, *flClientStyle*, *hmod*, *idResources*, *phwndClient*)

HWND *hwndParent*; /* handle of the parent window */
ULONG *flStyle*; /* frame-window style */
PULONG *pflCreateFlags*; /* creation flags */
PSZ *pszClientClass*; /* client-window class name */
PSZ *pszTitle*; /* address of title-bar text */
ULONG *flClientStyle*; /* client-window style */
HMODULE *hmod*; /* handle of the resource module */
USHORT *idResources*; /* frame-window identifier */
PHWND *phwndClient*; /* address of client-window handle */

The **WinCreateStdWindow** function creates a standard frame window.

Parameters

hwndParent Identifies the parent window. A main window is created if this parameter is **NULL**, **HWND_DESKTOP**, or a window handle returned by the **WinQueryDesktopWindow** function. An object window is created if this parameter is **HWND_DESKTOP** or a window handle returned by the **WinQueryObjectWindow** function.

flStyle Specifies the frame-window style. It can be any combination of the **WS_** styles (see the description for **flClientStyle**) and the following values:

Value	Meaning
FS_ACCELTABLE	Creates an accelerator table. The table is loaded from the application's resource file. It should have the same identifier as the menu and the icon (if any).
FS_BORDER	Creates a window that has a border with an inner border drawn with the active title-bar color. It is most often used by dialog boxes.
FS_DLGBORDER	Creates a window with a single line border around it. The width of the border line is SV_CXBORDER and the height is SV_CYBORDER . For a description of these system values, see the comments section of WinSetSysValue .
FS_ICON	The created window has an icon with the same identifier as the menu and accelerator table (if any).
FS_MOUSEALIGN	Creates a window whose position is relative to the current position of the mouse. Normally, this is only used by dialog boxes.

Value	Meaning
FS_NOBYTEALIGN	Creates a window whereby the frame window is not byte aligned. Setting this flag will decrease the performance of drawing operations to the client window.
FS_NOMOVEWITHOWNER	Creates a frame window that will not move if the owner window moves.
FS_SCREENALIGN	Creates a window that is aligned with respect to the screen. Normally, this is only used by dialog boxes.
FS_SHELLPOSITION	The Presentation Manager (shell) determines the position of the window, typically in a cascaded position from the last application that started.
FS_SIZEBORDER	Creates a sizing border.
FS_SYSMODAL	Creates a system modal window. For a description of a system modal window, see the comments section of the <code>WinSetSysModalWindow</code> function.
FS_TASKLIST	The window title is added to the Task Manager's list.
FS_STANDARD	Specifies a combination of the following flags: FS_ICON FS_ACCELTABLE FS_SHELLPOSITION FS_TASKLIST

pfCreateFlags Specifies options that control how the frame window is created. If no options are specified, `FCF_STANDARD` is used. The flags may be any combination of the following values:

Value	Meaning
FCF_ACCELTABLE	Creates an accelerator table. The table is loaded from the application's resource file. It should have the same identifier as the menu and the icon (if any).
FCF_BORDER	Creates a window with a single line border around it. The width of the border line is <code>SV_CXBORDER</code> and the height is <code>SV_CYBORDER</code> . For a description of these system values, see the <code>WinSetSysValue</code> function.
FCF_DLGBORDER	Creates a window that has a border with an inner border drawn with the active title-bar color. It is most often used by dialog boxes.
FCF_HORZSCROLL	Creates a horizontal scroll bar.
FCF_ICON	The created window has an icon with the same identifier as the menu and accelerator table (if any).

Value	Meaning
FCF_MAXBUTTON	Creates a maximize button.
FCF_MENU	Creates a menu bar.
FCF_MINBUTTON	Creates a minimize button.
FCF_MINMAX	Creates a minimize and a maximize button (FCF_MINBUTTON FCF_MAXBUTTON).
FCF_MOUSEALIGN	Creates a window whose position is relative to the current position of the mouse. Normally, this is only used by dialog boxes.
FCF_NOBYTEALIGN	Creates a window whereby the client window is not byte aligned. Setting this flag will decrease the performance of drawing operations to the client window.
FCF_NOMOVEWITHOWNER	Creates a window that will not move if the owner window moves.
FCF_SCREENALIGN	Creates a window that is aligned with respect to the screen. Normally, this is only used by dialog boxes.
FCF_SHELLPOSITION	The Presentation Manager (shell) determines the position of the window, typically in a cascaded position from the last application that started.
FCF_SIZEBORDER	Creates a sizing border.
FCF_SYSMENU	Creates a system menu.
FCF_SYSMODAL	Creates a system modal window. For a description of a system modal window, see the WinSetSysModalWindow function.
FCF_TASKLIST	Adds the window to the switch list of the Task Manager.
FCF_TITLEBAR	Creates a title bar.
FCF_VERTSCROLL	Creates a vertical scroll bar.
FCF_STANDARD	Specifies a combination of the following flags: FCF_TITLEBAR FCF_SYSMENU FCF_MENU FCF_SIZEBORDER FCF_MINMAX FCF_ICON FCF_ACCELTABLE FCF_SHELLPOSITION FCF_TASKLIST

pszClientClass Points to the client-window class name. If the *pszClientClass* parameter is not a zero-length string, a client window of style *flClientStyle* and class *pszClientClass* is created. This parameter is an application-specified name (defined by the `WinRegisterClass` function), the name of a preregistered WC class, or a WC constant. If this parameter is NULL, no client area is created.

pszTitle Points to the title-bar text. This parameter is ignored if `FCF_TITLEBAR` is not specified in the *pflCreateFlags* parameter.

flClientStyle Specifies the client-window style. It can be a combination of one or more of the following values:

Value	Meaning
<code>WS_CLIPCHILDREN</code>	Prevents a window from painting over its child windows.
<code>WS_CLIPSIBLINGS</code>	Prevents a window from painting over its sibling windows.
<code>WS_DISABLED</code>	Disables mouse and keyboard input to the window. It is used to temporarily prevent the user from using the window.
<code>WS_MAXIMIZED</code>	Enlarges the window to the maximum size.
<code>WS_MINIMIZED</code>	Reduces the window to the minimum size.
<code>WS_PARENTCLIP</code>	Prevents a window from painting over its parent window.
<code>WS_SAVEBITS</code>	Saves the image under the window as a bit-map. When the window is moved or hidden, the system restores the image by copying the bits.
<code>WS_SYNCPAINT</code>	Causes the window to immediately receive <code>WM_PAINT</code> messages after a part of the window becomes invalid. Without this style, the window receives <code>WM_PAINT</code> messages only if no other message is waiting to be processed.
<code>WS_VISIBLE</code>	Makes the window visible. This window will be drawn on the screen unless overlapping windows completely obscure it. Windows without this style are hidden.

This parameter is ignored if the *pszClientClass* parameter is a zero-length string.

hmod Identifies the module that contains the resource definitions. This parameter can be either the module handle returned by the `DosLoadModule` function or NULL for the application's module. This parameter is ignored unless `FS_ICON`, `FS_ACCELTABLE`, or `FS_MENU` is specified in the *flStyle* parameter.

idResources Identifies the frame-window identifier and the identifier within the resource definition of the required resource. The application must ensure that all resources related to one frame window have the same identifier value.

phwndClient Points to the variable that receives the client-window handle. It will be NULL if the function fails.

Return Value

The return value is the handle of the frame window, or it is NULL if an error occurs.

Example

This example shows a typical initialization function for a window. The function first registers the window class, then calls `WinCreateStdWindow` to create a standard window and returns immediately if the function fails. Otherwise, it continues on to do other initialization processing.

Note: The `FCF_STANDARD` constant can only be used if you have all the resources in defines. For example, if you use this constant, and you don't have an accelerator table, the function will fail.

```
CHAR szClassName[] = "Generic";    /* window class name */
HWND hwndClient;                  /* handle to the client */
HWND hwndClient;                  /* handle to the client */

BOOL GenericInit()
{
    ULONG flStyle;

    flStyle = FCF_STANDARD;
    if (!WinRegisterClass(hab, szClassName, GenericWndProc, OL, 0))
        return (FALSE);

    hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
        OL, /* frame-window style */
        &flStyle, /* window style */
        szClassName, /* class name */
        "Generic Application", /* window title */
        OL, /* default client style */
        NULL, /* resource in executable file */
        IDM_RESOURCE, /* resource id */
        &hwndClient); /* receives client window handle */

    if (!hwndFrame)
        return (FALSE);
    else {
        . /* other initialization code */
    }
}
```

See Also

`DosLoadModule`, `WinCreateWindow`, `WinQueryDesktopWindow`, `WinQueryObjectWindow`, `WinSetSysModalWindow`, `WinSetSysValue`, `WinSetWindowPos`, `WinSetWindowUShort`

WinCreateWindow

`HWND WinCreateWindow` (*hwndParent*, *pszClass*, *pszName*, *flStyle*, *x*, *y*, *cx*, *cy*, *hwndOwner*, *hwndInsertBehind*, *id*, *pCtlData*, *pPresParams*)

```
HWND hwndParent;    /* desktop-window handle */
PSZ pszClass;      /* address of registered class name */
PSZ pszName;       /* address of window text */
ULONG flStyle;     /* window style */
SHORT x;           /* horizontal position of window */
SHORT y;           /* vertical position of window */
SHORT cx;          /* window width */
SHORT cy;          /* window depth */
HWND hwndOwner;    /* owner-window handle */
HWND hwndInsertBehind; /* handle to sibling window */
USHORT id;         /* window identifier */
PVOID pCtlData;    /* address of buffer */
PVOID pPresParams; /* reserved */
```

The `WinCreateWindow` function creates a new window.

Parameters

hwndParent Specifies the parent window for the new window. Any valid window handle can be used.

pszClass Points to the registered class name. This parameter is either an application-specified name (defined by the `WinRegisterClass` function), the name of a preregistered WC class, or a WC constant.

pszName Points to window text or other class-specific data. The actual structure of the data is class-specific. This data is usually a zero-terminated string, which is often displayed in the window.

flStyle Specifies the window style. It can be a combination of one or more of the following values:

Value	Meaning
WS_CLIPCHILDREN	Prevents a window from painting over its child windows.
WS_CLIPSIBLINGS	Prevents a window from painting over its sibling windows.
WS_DISABLED	Disables mouse and keyboard input to the window. It is used to temporarily prevent the user from using the window.
WS_MAXIMIZED	Enlarges the window to the maximum size.
WS_MINIMIZED	Reduces the window to the minimum size.
WS_PARENTCLIP	Prevents a window from painting over its parent window.
WS_SAVEBITS	Saves the image under the window as a bitmap. When the window is moved or hidden, the system restores the image by copying the bits.
WS_SYNCPAINT	Causes the window to immediately receive WM_PAINT messages after a part of the window becomes invalid. Without this style, the window receives WM_PAINT messages only if no other message is waiting to be processed.
WS_VISIBLE	Makes the window visible. This window will be drawn on the screen unless overlapping windows completely obscure it. Windows without this style are hidden.

x Specifies the horizontal position of the window relative to the origin of the parent window.

y Specifies the vertical position of the window relative to the origin of the parent window.

cx Specifies the window width in pels.

cy Specifies the window depth in pels.

hwndOwner Identifies the owner window.

hwndInsertBehind Identifies the sibling window behind which the specified window is placed. If this parameter is `HWND_TOP`, the specified window is placed on top of all its sibling windows. If this parameter is `HWND_BOTTOM`,

the specified window is placed behind all its sibling windows. If this parameter is not `HWND_TOP` or `HWND_BOTTOM`, or it is a child window of the desktop window identified by *hwndParent*, then `NULL` is returned.

id Specifies the window identifier, a value given by the application that allows a specific child window to be identified. For example, the controls of a dialog box have unique identifiers so that an owner window can distinguish which control has notified it. Window identifiers are also used for frame windows.

pCilData Points to the buffer that contains class-specific information. This data is passed to the window procedure by the `WM_CREATE` message.

pPresParams Points to the presentation parameters. This is a reserved field and must be zero.

Return Value The return value is the handle of the window, or `NULL` if an error occurs.

Comments `WinCreateWindow` sends a `WM_CREATE` message to the window procedure of the window being created, and then sends the `WM_ADJUSTWINDOWPOS` message before the window is displayed. The values passed are those given to the `WinCreateWindow` function.

The `WM_SIZE` message is not sent by `WinCreateWindow` while the window is being created. Any required size processing is performed during the processing of the `WM_CREATE` message.

See Also `WinCreateStdWindow`, `WinQueryObjectWindow`, `WinRegisterClass`

■ WinDdeInitiate

BOOL `WinDdeInitiate`(*hwndClient*, *pszAppName*, *pszTopicName*)

HWND *hwndClient*; /* handle of the client window */

PSZ *pszAppName*; /* address of application name */

PSZ *pszTopicName*; /* address of topic name */

The `WinDdeInitiate` function initiates a DDE conversation by sending a `WM_DDE_INITIATE` message. All server applications matching the supplied application name will reply with a `WM_DDE_INITIATEACK` message if they support the specified topic.

Parameters *hwndClient* Identifies the client window. Since the window handle serves as the recipient of DDE messages for its conversation, it need not be visible. All applications must rely only on their own window handles to identify conversations.

pszAppName Points to a `NULL` terminated string containing the server's application name. If the string has zero length, any server application may respond.

pszTopicName Points to a `NULL` terminated string containing the topic name. If the string has zero length, the server will respond once for each topic that it supports.

Return Value The return value is `TRUE` if the function is successful or `FALSE` if an error occurs.

Comments The WM_DDE_INITIATE is sent to all top-level frame windows in the system. Any prospective server must subclass a top-level frame window in order to retrieve this message.

See Also WinDdePostMsg, WinDdeRespond

■ WinDdePostMsg

```

BOOL WinDdePostMsg(hwndTo, hwndFrom, wm, pddes, fRetry)
HWND hwndTo;           /* handle of the window to post to */
HWND hwndFrom;        /* handle of the window that is posting */
USHORT wm;            /* message number */
PDDESTRUCT pddes;    /* address of the structure to pass */
BOOL fRetry;          /* retry flag */

```

The WinDdePostMsg function posts a DDE message to the specified window's message queue.

Parameters

- hwndTo* Identifies the window the message is to be posted to.
- hwndFrom* Identifies the window that is posting the message.
- wm* Specifies the message being sent. The following messages may be sent:

```

WM_DDE_ACK
WM_DDE_ADVISE
WM_DDE_DATA
WM_DDE_EXECUTE
WM_DDE_INITIATE
WM_DDE_INITIATEACK
WM_DDE_POKE
WM_DDE_REQUEST
WM_DDE_TERMINATE
WM_DDE_UNADVISE

```

pddes Points to a DDESTRUCT structure. The structure has the following form:

```

typedef struct _DDESTRUCT {
    ULONG      cbData;
    USHORT     fsStatus;
    USHORT     usFormat;
    USHORT     offszItemName;
    USHORT     offabData;
} DDESTRUCT;

```

For more information, see Chapter 4, "Types, Macros, Structures."

fRetry Specifies whether or not to try to send the message again if it fails the first time because the destination queue was full. If TRUE, the message is sent repeatedly at one second intervals until it could be successfully posted.

Return Value The return value is TRUE if the function is successful, or FALSE if an error occurred. If the *fRetry* parameter is TRUE, this function will still return FALSE if the message cannot be sent for any reason other than the destination queue being full.

See Also WinDdeInitiate, WinDdeRespond

■ WinDdeRespond

MRESULT WinDdeRespond(*hwndClient*, *hwndServer*, *pszAppName*, *pszTopicName*)

HWND *hwndClient*; /* handle of the client window */

HWND *hwndServer*; /* handle of the server window */

PSZ *pszAppName*; /* address of name of application */

PSZ *pszTopicName*; /* address of name of topic */

The **WinDdeRespond** function sends an acknowledgement message (**WM_DDE_INITIATEACK**) back to the application that sent a **WM_DDE_INITIATE** message.

Parameters

- hwndClient* Identifies the client window.
- hwndServer* Identifies the server window.
- pszAppName* Points to the name of the application that is acknowledging the **WM_DDE_INITIATE** message.
- pszTopicName* Points to the topic name that the acknowledging application will accept.

Return Value The return value is the result of sending the **WM_DDE_INITIATEACK** message to the client window.

See Also [WinDdeInitiate](#)

■ WinDefAVioWindowProc

MRESULT WinDefAVioWindowProc(*hwnd*, *msg*, *mp1*, *mp2*)

HWND *hwnd*; /* handle of the window */

USHORT *msg*; /* message */

MPARAM *mp1*; /* message parameter */

MPARAM *mp2*; /* message parameter */

The **WinDefAVioWindowProc** function calls the default window procedure for an advanced video-input-and-output (AVIO) window. An AVIO application must use this function instead of the **WinDefWindowProc** function that is used with Presentation Manager applications.

The default window procedure provides default processing for any window messages that an application does not process. This function is used to ensure that every message is processed. It should be called with the same parameters as those received by the window procedure.

Parameters

- hwnd* Identifies the window that received the message.
- msg* Specifies the message.
- mp1* Specifies message parameter 1.
- mp2* Specifies message parameter 2.

Return Value The return value is dependent on the message that was passed to this function.

See Also [WinDefWindowProc](#)

■ WinDefDlgProc

MRESULT WinDefDlgProc(*hwndDlg*, *msg*, *mp1*, *mp2*)

HWND *hwndDlg*; /* handle of the dialog */
USHORT *msg*; /* message */
MPARAM *mp1*; /* message parameter */
MPARAM *mp2*; /* message parameter */

The **WinDefDlgProc** function calls the default dialog procedure. The default dialog procedure provides default processing for any dialog window messages that an application does not process. This function is used to ensure that every message is processed. It should be called with the same parameters as those received by the dialog procedure.

Parameters

- hwndDlg* Identifies the dialog window that received the message.
- msg* Specifies the message.
- mp1* Specifies message parameter 1.
- mp2* Specifies message parameter 2.

Return Value The return value is dependent on the message that was passed to this function.

Example This example shows a typical dialog box procedure. A **switch** statement is used to process individual messages. All messages not processed are passed on to the **WinDefDlgProc** function.

```
MRESULT CALLBACK AboutDlg(hwnd, usMessage, mp1, mp2)
HWND hwnd;
USHORT usMessage;
MPARAM mp1;
MPARAM mp2;
{
    switch (usMessage) {
        /*
         * process whatever messages you want here and send the rest
         * to WinDefWindowProc
         */
        default:
            return (WinDefDlgProc(hwnd, usMessage, mp1, mp2));
    }
}
```

See Also [WinDefWindowProc](#)

■ WinDefWindowProc

MRESULT WinDefWindowProc(*hwnd*, *msg*, *mp1*, *mp2*)

HWND *hwnd*; /* handle of the window */
USHORT *msg*; /* message */
MPARAM *mp1*; /* message parameter */
MPARAM *mp2*; /* message parameter */

The **WinDefWindowProc** function calls the default window procedure. The default window procedure provides default processing for any window messages that an application does not process. This function is used to ensure that every message is processed. It should be called with the same parameters as those received by the window procedure.

Parameters

hwnd Identifies the window that received the message.

msg Specifies the message.

mp1 Specifies message parameter 1.

mp2 Specifies message parameter 2.

Return Value The return value is dependent on the message that was passed to this function.

Example This example shows a typical window procedure. A **switch** statement is used to process individual messages. All messages not processed are passed on to the **WinDefWindowProc** function.

```
MRESULT CALLBACK GenericWndProc(hwnd, usMessage, mp1, mp2)
HWND hwnd;
USHORT usMessage;
MPARAM mp1;
MPARAM mp2;
{
    switch (usMessage) {
        /*
         * process whatever messages you want here and send the rest
         * to WinDefWindowProc
         */
        default:
            return (WinDefWindowProc(hwnd, usMessage, mp1, mp2));
    }
}
```

See Also WinDefAViewWindowProc, WinDefDlgProc

■ WinDeleteAtom

ATOM WinDeleteAtom(*hAtomTbl*, *atom*)

HATOMTBL *hAtomTbl*; /* handle of the atom table */

ATOM *atom*; /* atom */

The **WinDeleteAtom** function deletes an atom from the atom table.

Parameters

hAtomTbl Identifies the atom table. This handle must have been created by a previous call to the **WinCreateAtomTable** function.

atom Specifies the atom to be deleted.

Return Value The return value is **NULL** if the function is successful. Otherwise, it is equal to the value of the *atom* parameter if the function failed and the atom has not been deleted.

If the passed atom is an integer atom, **NULL** is returned. If it is not an integer atom and it is a valid atom for the given atom table (it has an atom name and use count associated with it), the use count is decreased by one and **NULL** is returned. If the use count has been decreased to zero, the atom name and use count are removed from the atom table.

See Also WinAddAtom, WinCreateAtomTable

■ WinDestroyAccelTable

BOOL WinDestroyAccelTable(*haccel*)

HACCEL *haccel*; /* handle of the accelerator table */

The **WinDestroyAccelTable** function destroys an accelerator table. Before an application terminates, it should call this function for each accelerator table created with a call to the **WinCreateAccelTable** function.

Parameters *haccel* Identifies the accelerator table to be destroyed. This handle must have been created by a previous call to the **WinCreateAccelTable** function.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinCreateAccelTable

■ WinDestroyAtomTable

HATOMTBL WinDestroyAtomTable(*hAtomTbl*)

HATOMTBL *hAtomTbl*; /* handle of the atom table */

The **WinDestroyAtomTable** function destroys an atom table created by the **WinCreateAtomTable** function.

Parameters *hAtomTbl* Identifies the atom table to be destroyed. This handle must have been created by a previous call to the **WinCreateAtomTable** function.

Return Value The return value is NULL if the function is successful. Otherwise, the return value is the *hAtomTbl* parameter.

This method of returning failure allows for updating the status of an atom table and destroying it at the same time with a call similar to the following:

```
hAtomTbl = WinDestroyAtomTable(hAtomTbl);
```

See Also WinCreateAtomTable

■ WinDestroyCursor

BOOL WinDestroyCursor(*hwnd*)

HWND *hwnd*; /* handle of the window */

The **WinDestroyCursor** function erases and destroys the current cursor, if it belongs to the specified window.

This function has no effect if the current cursor does not belong to the specified window.

Parameters *hwnd* Identifies the window to which the cursor belongs.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments It is not necessary to call the **WinDestroyCursor** function before calling the **WinCreateCursor** function, as the **WinCreateCursor** function will automatically destroy any existing cursor.

See Also **WinCreateCursor**, **WinQueryCursorInfo**, **WinShowCursor**

■ WinDestroyHeap

HHEAP WinDestroyHeap(*hHeap*)

HHEAP *hHeap*; /* handle of the heap */

The **WinDestroyHeap** function destroys a heap created by the **WinCreateHeap** function. If the **WinCreateHeap** function allocated a segment, the **WinDestroyHeap** will free that segment.

Parameters *hHeap* Identifies the heap to be destroyed. This handle must have been created by a previous call to the **WinCreateHeap** function.

Return Value The return value is zero if the function is successful. Otherwise, it is nonzero, indicating that an error occurred.

See Also **WinCreateHeap**

■ WinDestroyMsgQueue

BOOL WinDestroyMsgQueue(*hmq*)

HMQ *hmq*; /* handle of the message queue */

The **WinDestroyMsgQueue** function destroys a message queue. Any thread or application that creates a message queue should call **WinDestroyMsgQueue** to destroy that message queue before terminating.

Parameters *hmq* Identifies the message queue to be destroyed. This handle must have been created by a previous call to the **WinCreateMsgQueue** function in the same thread that is calling the **WinDestroyMsgQueue** function.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

Example This example calls **WinInitialize** to initialize the thread for making Presentation Manager function calls, and calls **WinCreateMsgQueue** to create a message queue for the thread. Before the thread terminates, it calls **WinDestroyMsgQueue** to destroy the message queue.

```

hab = WinInitialize(NULL);
hmq = WinCreateMsgQueue(hab, DEFAULT_QUEUE_SIZE);
    /* initialization and message loop */
WinDestroyMsgQueue(hmq);

```

See Also **WinCreateMsgQueue**, **WinInitialize**

■ WinDestroyPointer

BOOL WinDestroyPointer(*hptr*)

HPOINTER *hptr*; /* handle of the pointer or icon */

The **WinDestroyPointer** function destroys a pointer or an icon. A pointer can be destroyed only by the process that created it. The system pointers and icons cannot be destroyed.

Parameters *hptr* Identifies the pointer or icon to destroy.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinCreatePointer

■ WinDestroyWindow

BOOL WinDestroyWindow(*hwnd*)

HWND *hwnd*; /* handle of the window to destroy */

The **WinDestroyWindow** function destroys a window and any child windows of that window. If the window is locked, this function will not return until the window has been unlocked (and destroyed).

Parameters *hwnd* Identifies the window to be destroyed.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments The window to be destroyed must have been created by the thread that issues this call. Before the window identified by the *hwnd* parameter is itself destroyed, all child windows it owns are also destroyed.

Note that any presentation spaces created with the **WinGetPS** function must be released by calling the **WinReleasePS** function. This must be done before calling **WinDestroyWindow**.

The following messages are sent by this function:

Value	Meaning
WM_ACTIVATE	Sent with the first message parameter equal to FALSE if the window being destroyed is the active window.
WM_DESTROY	Always sent to the window being destroyed after the window has been hidden on the device, but before its child windows have been destroyed.
WM_OTHERWINDOWDESTROYED	Sent to all main windows if the window being destroyed, or any of its descendants, has been registered with the WinRegisterWindowDestroy function.
WM_RENDERALLFMTS	Sent if the clipboard owner is being destroyed and there are unrendered formats in the clipboard.

If the window being destroyed is the active window, then both the active window and the input-focus window are transferred to another window when the window is destroyed.

See Also WinCreateStdWindow, WinCreateWindow, WinGetPS, WinLockWindow, WinRegisterWindowDestroy, WinReleasePS

■ WinDismissDlg

```
BOOL WinDismissDlg(hwndDlg, usResult)
HWND hwndDlg;      /* handle of the dialog */
USHORT usResult;   /* result code to return */
```

The WinDismissDlg function hides the dialog window and causes the WinProcessDlg or WinDlgBox function to return.

Parameters *hwndDlg* Identifies the dialog window to be hidden.
usResult Specifies the value that is returned to the caller of WinProcessDlg or WinDlgBox.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments This function is required to complete the processing of a modal dialog window and is called from its dialog procedure.

The WinDismissDlg function is automatically called by the WinDefDlgProc function upon receiving a WM_COMMAND message. The WinDefDlgProc function will set *usResult* to the identifier of the control that generated the WM_COMMAND message.

Note that this function can be called from a modeless dialog box, although this is not necessary since there is no internal message processing loop. If the function is called, the dialog box window is hidden. The application must destroy the dialog box window, if required.

Example This example shows a typical dialog procedure that has both an OK and a Cancel button. If the user selects the OK button, WinDismissDlg is called with a result value of TRUE. If the user selects the Cancel button, WinDismissDlg is called with a result value of FALSE.

```
case WM_COMMAND:
    switch (SHORT1FROMMP(mpl)) {
        case ID_ENTER: /* OK button selected */
            WinDismissDlg(hwnd, TRUE);
            return (OL);

        case ID_CANCEL: /* Cancel button selected */
            WinDismissDlg(hwnd, FALSE);
            return (OL);
```

See Also WinDlgBox, WinProcessDlg

■ WinDispatchMsg

ULONG WinDispatchMsg(*hab*, *pqmsg*)

HAB *hab*; /* handle of the anchor block */
PQMSG *pqmsg*; /* address of structure for message queue */

The **WinDispatchMsg** dispatches a message to a window. It is typically used to dispatch a message retrieved with the **WinGetMsg** function. Unlike the **WinSendMsg** function, the **WinDispatchMsg** function does not call any hooks installed with the **WinSetHook** function.

Parameters *hab* Identifies an anchor block.

pqmsg Points to a **QMSG** structure that contains the message. The **QMSG** structure has the following form:

```
typedef struct _QMSG {
    HWND hwnd;
    USHORT msg;
    MPARAM mp1;
    MPARAM mp2;
    ULONG time;
    POINTL pt1;
} QMSG;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is the value returned by the window procedure that was called.

Example This example calls **WinGetMsg** to retrieve a message and **WinDispatchMsg** to send the message.

```
HAB hab;
QMSG qmsg;

while (WinGetMsg(hab, /* handle to the anchor block */
                &qmsg, /* address of the message queue structure */
                NULL, /* accept messages for any window */
                0, /* first message to accept */
                0) /* accept all messages */
    WinDispatchMsg(hab, &qmsg);
```

See Also **WinGetMsg**, **WinProcessDlg**, **WinSendMsg**, **WinSetHook**

■ WinDlgBox

USHORT WinDlgBox(*hwndParent*, *hwndOwner*, *pfnDlgProc*, *hmod*, *idDlg*, *pCreateParams*)

HWND *hwndParent*; /* handle of the parent window */
HWND *hwndOwner*; /* handle of the owner window */
PFNWP *pfnDlgProc*; /* dialog procedure address */
HMODULE *hmod*; /* handle of resource module */
USHORT *idDlg*; /* identifies the dialog */
PVOID *pCreateParams*; /* address of application-specific data */

The **WinDlgBox** function loads and processes a modal dialog box and returns the *usResult* value passed to the **WinDismissDlg** function. It is equivalent to calling **WinLoadDlg**, **WinProcessDlg**, and **WinDestroyWindow**.

This function does not return until the dialog procedure calls the **WinDismissDlg** function.

Parameters

hwndParent Identifies the parent window.

hwndOwner Identifies the owner window.

pfnDlgProc Points to the dialog procedure.

hmod Identifies the module that contains the dialog template. This parameter can be either the module handle returned by the `DosLoadModule` function or NULL for the application's module.

idDlg Identifies the dialog window.

pCreateParams Points to application-specific data that is passed to the dialog procedure as part of the WM_INITDLG message.

Return Value The return value is the value specified in the *usResult* parameter of the `WinDismissDlg` function, or `DID_ERROR` if an error occurred while trying to load the dialog box.

Example This example processes an application-defined message (`IDM_OPEN`) and calls `WinDlgBox` to load a dialog box.

```
case IDM_OPEN:
    if (WinDlgBox(HWND_DESKTOP,
        hwndFrame, /* handle of the owner */
        OpenDlg, /* dialog procedure address */
        NULL, /* location of dialog resource */
        IDD_OPEN, /* resource identifier */
        NULL) { /* application-specific data */
        . /* code executed if dialog box returns TRUE */
    }
}
```

See Also `DosLoadModule`, `WinDismissDlg`, `WinLoadDlg`, `WinProcessDlg`

■ WinDrawBitmap

BOOL WinDrawBitmap(*hpsDst*, *hbm*, *prclSrc*, *pptlDst*, *clrFore*, *clrBack*, *fs*)

HPS *hpsDst*; /* handle of the destination presentation space */

HBITMAP *hbm*; /* handle of the bitmap */

PRECTL *prclSrc*; /* address of structure with rectangle coordinates */

PPOINTL *pptlDst*; /* address of structure with bitmap position */

LONG *clrFore*; /* color of the foreground */

LONG *clrBack*; /* color of the background */

USHORT *fs*; /* bitmap-drawing flags */

The `WinDrawBitmap` function draws a bitmap using the current image colors and mixes.

Parameters *hpsDst* Identifies the presentation space in which the bitmap is drawn.

hbm Identifies the bitmap.

prclSrc Points to the `RECTL` data structure that contains the coordinates of the rectangle to be drawn. If this parameter is NULL, the entire bitmap is drawn. The `RECTL` structure has the following form:

```
typedef struct _RECTL {
    LONG xLeft;
    LONG yBottom;
    LONG xRight;
    LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

pptlDst Points to the position of the lower left of the bitmap in the presentation space (in presentation space coordinates).

clrFore Specifies the color of the foreground.

clrBack Specifies the color of the background.

fs Specifies the flags that determine how the bitmap is drawn. It can be one of the following values:

Value	Meaning
DBM_HALFTONE	Use the OR operator to combine the bitmap with an alternating pattern of ones and zeros before drawing it. This flag can be used in conjunction with either DBM_NORMAL or DBM_INVERT.
DBM_IMAGEATTRS	The <i>clrFore</i> and <i>clrBack</i> parameters are ignored and the image attribute colors already selected in <i>hpsDst</i> are used instead.
DBM_INVERT	Draw the bitmap inverted, using ROP_NOTSRCCOPY.
DBM_NORMAL	Draw the bitmap normally, using ROP_SRCCOPY.
DBM_STRETCH	The <i>pptlDst</i> parameter points to a RECTL data structure, representing a rectangle in the destination presentation space to which the bitmap should be stretched.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also GpiCreateBitmap, GpiLoadBitmap, WinGetSysBitmap

■ WinDrawBorder

```

BOOL WinDrawBorder(hps, prcl, cx, cy, clrFore, clrBack, fsCmd)
HPS hps;           /* handle of the presentation space */
PRECTL prcl;       /* address of structure with bounding rectangle */
SHORT cx;          /* width of the border */
SHORT cy;          /* height of the border */
LONG clrFore;      /* color of the foreground */
LONG clrBack;      /* color of the background */
USHORT fsCmd;     /* border-drawing flags */

```

The **WinDrawBorder** function draws a border (a rectangular frame, normally used around the edge of a window).

Parameters *hps* Identifies the presentation space in which the border is drawn.

prcl Points to a **RECTL** structure that contains the bounding rectangle for the border. The border is drawn within this rectangle. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

cx Specifies the width of the left and right sides of the border rectangle.

cy Specifies the height of the top and bottom sides of the border rectangle.

clrFore Specifies the color of the foreground.

clrBack Specifies the color of the background.

fsCmd Specifies the flags that determine how the border is drawn. It can be one of the following values:

Value	Meaning
DB_AREAMIXMODE	Draws the border with the current area attributes, using the current-area foreground mix mode mapped to a GpiBitBlt raster operation. Note that the background mix mode is ignored by WinDrawBorder in this release because GpiBitBlt ignores it.
DB_DESTINVERT	Inverts the border.
DB_DLGBORDER	A standard dialog border is drawn. If DB_PATCOPY is specified, then an active dialog border is drawn. If DB_PATINVERT is specified, then an inactive dialog border is drawn. If this option is specified, the <i>cx</i> and <i>cy</i> parameters are ignored.
DB_INTERIOR	The interior of the border is drawn with the current pattern background color. The border is drawn in the current pattern foreground color.
DB_PATCOPY	Draws the border with the current area attributes, forcing a mix mode of ROP_SRCOPY.
DB_PATINVERT	Draws the border with the current area attributes, forcing a mix mode of ROP_PATINVERT.
DB_ROP	Used as a mask to isolate the raster operation related flags of the <i>fsCmd</i> parameter.
DB_STANDARD	The <i>cx</i> and <i>cy</i> parameters are multiplied by the system SV_CXBORDER and SV_CYBORDER constants to produce the width and height of the border sides.

The current area attributes pattern is used. For instance, if the caller selects a diagonal-crosshatch symbol, the borders will be drawn with diagonal cross-hatches, no matter what colors are selected. The only raster operation which does not use the pattern is DB_DESTINVERT.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also GpiBitBlt

■ WinDrawPointer

```

BOOL WinDrawPointer(hps, x, y, hptr, fs)
HPS hps;           /* handle of the presentation space */
SHORT x;           /* horizontal position */
SHORT y;           /* vertical position */
HPOINTER hptr;   /* handle to the mouse pointer */
USHORT fs;        /* pointer-drawing flags */

```

The **WinDrawPointer** function draws a pointer.

Parameters

hps Identifies the presentation space in which the pointer is drawn.

x Specifies the *x*-coordinate at which to draw the pointer.

y Specifies the *y*-coordinate at which to draw the pointer.

hptr Identifies the pointer.

fs Specifies the flags that determine how the pointer is drawn. It can be one of the following values:

Value	Meaning
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.
DP_NORMAL	Draw the pointer as it normally appears.

Return Value

The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also

WinCreatePointer, **WinLoadPointer**

■ WinDrawText

```

SHORT WinDrawText(hps, cchText, pszText, prcl, clrFore, clrBack, fsCmd)
HPS hps;           /* handle of the presentation space */
SHORT cchText;     /* number of characters in the string */
PSZ pszText;      /* address of the text */
PRECTL prcl;      /* address of structure with formatting dimensions */
LONG clrFore;     /* color of the foreground */
LONG clrBack;     /* color of the background */
USHORT fsCmd;     /* text-drawing flags */

```

The **WinDrawText** function draws a single line of formatted text into a specified rectangle.

Parameters

hps Identifies the presentation space in which to draw the text.

cchText Specifies the number of characters in the string to draw. If this is set to 0xFFFF, the string is assumed to be null-terminated and its length is calculated by the function itself.

pszText Points to the character string to draw. A carriage-return or linefeed character terminates the line, even if the line is shorter than specified by the *cchText* parameter.

prcl Points to a **RECTL** data structure that contains the rectangle in which the text is formatted. If **DT_QUERYEXTENT** is specified in the *fsCmd* parameter, the **RECTL** structure is set to the string's bounding rectangle upon return from **WinDrawText**. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

clrFore Specifies the color of the foreground. This parameter is ignored if **DT_TEXTATTRS** is set in the *fsCmd* parameter.

clrBack Specifies the color of the background. This parameter is ignored if **DT_TEXTATTRS** is set in the *fsCmd* parameter.

fsCmd Specifies an array of flags that determine how the text is drawn. It can be any of the following values:

Value	Meaning
DT_LEFT	Left justify the text.
DT_RIGHT	Right justify the text.
DT_TOP	Top justify the text.
DT_BOTTOM	Bottom justify the text.
DT_CENTER	Center the text.
DT_VCENTER	Center the text vertically.
DT_ERASERECT	This flag causes the RECTL structure specified in <i>prcl</i> to be filled with the window color before printing the text, but only if the DT_QUERYEXTENT flag is <i>not</i> specified.
DT_EXTERNALLEADING	This flag causes the external leading value from the passed font to be added to the bottom of the bounding rectangle before returning. It only has an effect when both DT_TOP and DT_QUERYEXTENT are also specified.
DT_HALFTONE	Make the text display halftone.
DT_MNEMONIC	If a mnemonic prefix character is encountered, draw the next character with mnemonic emphasis.
DT_QUERYEXTENT	No drawing is performed. The <i>prcl</i> parameter is changed to a rectangle that bounds the string if it was drawn with the WinDrawText function.
DT_TEXTATTRS	This flag causes the colors specified in <i>clrFore</i> and <i>clrBack</i> to be ignored. The colors already selected in the presentation space are used instead.

Value	Meaning
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 linefeed character. When calculating whether a particular word will fit within the given rectangle, WinDrawText does not consider the trailing blanks. It tests only the length of the visible part of the word against the right edge of the rectangle. Note that WinDrawText will always try 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.

Some of the DT flags are mutually exclusive. Only one flag from each of the following groups is significant:

- DT_LEFT, DT_CENTER, DT_RIGHT
- DT_TOP, DT_VCENTER, DT_BOTTOM

Return Value

The return value is the number of characters, drawn by the function, that fit completely within the structure pointed to by *prcl*.

Example

This example shows how the **WinDrawText** function can be used to wrap text within a window by using the **DT_WORDBREAK** flag. The *cchDrawn* variable receives the number of characters actually drawn by the **WinDrawText** function. If this value is zero, no text was drawn and the **for** loop is exited. This can occur if the vertical height of the window is too short for the entire text. Otherwise, *cchDrawn* is added to the *cchTotalDrawn* variable to provide an offset into the string for the next call to **WinDrawText**.

```

hps = WinGetPS(hwnd);           /* get a ps for the entire window */
WinQueryWindowRect(hwnd, &rcl); /* get window dimensions */
WinFillRect(hps, &rcl, CLR_WHITE); /* clear entire window */
cchText = strlen(pszText);      /* get length of string */
cyCharHeight = 15L;           /* set character height */
for (cchTotalDrawn = 0;
     cchTotalDrawn != cchText; /* until all chars drawn */
     rcl.yTop -= cyCharHeight) {

    /* draw the text */

    cchDrawn = WinDrawText(hps, /* presentation-space handle */
                          cch - cchTotalDrawn, /* length of text to draw */
                          pszText + cchTotalDrawn, /* address of the text */
                          &rcl, /* rectangle to draw in */
                          OL, /* foreground color */
                          OL, /* background color */
                          DT_WORDBREAK | DT_TOP | DT_LEFT | DT_TEXTATTRS);
    if (cchDrawn)
        cchTotalDrawn += cchDrawn;
    else
        break; /* text could not be drawn */
}
WinReleasePS(hps); /* release the ps */

```

See Also

WinSetDlgItemText

■ WinEmptyClipbrd

BOOL WinEmptyClipbrd (*hab*)

HAB *hab*; /* handle of the anchor block */

The **WinEmptyClipbrd** function empties the clipboard, removing and freeing all handles to data that were on the clipboard.

Parameters *hab* Identifies an anchor block.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comment This function will send a WM_DESTROYCLIPBOARD message to the clipboard owner.

See Also **WinCloseClipbrd**, **WinOpenClipbrd**

■ WinEnablePhysInput

BOOL WinEnablePhysInput (*hwndDesktop*, *fEnable*)

HWND *hwndDesktop*; /* handle of the desktop */

BOOL *fEnable*; /* input status */

The **WinEnablePhysInput** function enables or disables queuing of keyboard and mouse input.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be HWND_DESKTOP or the desktop window handle.

fEnable Specifies whether the input is queued or disabled. If TRUE, keyboard and mouse input are queued. If FALSE, keyboard and mouse input are disabled.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also **WinIsPhysInputEnabled**

■ WinEnableWindow

BOOL WinEnableWindow (*hwnd*, *fEnable*)

HWND *hwnd*; /* handle of the window */

BOOL *fEnable*; /* enable-state flag */

The **WinEnableWindow** function sets the window's enabled state.

Parameters *hwnd* Identifies the window whose enabled state is to be set.

fEnable Specifies the new enabled state. If TRUE, the window state is set to enabled. If FALSE, the window state is set to disabled.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

- Comments** If the enable state of the window is changing, a WM_ENABLE message is sent before this function returns.
- If a window is disabled, all its child windows are implicitly disabled, although they are not sent a WM_ENABLE message.
- See Also** WinIsWindowEnabled

■ WinEnableWindowUpdate

BOOL WinEnableWindowUpdate(*hwnd*, *fEnable*)

HWND *hwnd*; /* handle of the window to be enabled or disabled */

BOOL *fEnable*; /* enabled-state flag */

The WinEnableWindowUpdate function enables or disables the window visibility state for subsequent drawing to a window. This function can be used to prevent unnecessary redrawing when making changes to a window. The window can then be redrawn by calling this function with the *fEnable* parameter set to TRUE.

- Parameters** *hwnd* Identifies the window whose enable state will be changed.
- fEnable* Specifies whether drawing to the window is enabled. If TRUE, any subsequent drawing into the window will be visible. If FALSE, any subsequent drawing into the window will be invisible.
- Return Value** The return value is TRUE if the function is successful or FALSE if an error occurs.
- Comments** When the WinEnableWindowUpdate function is called with the *fEnable* parameter TRUE, the entire window is invalidated, however WS_SYNCPAINT windows are not redrawn. If you need to redraw a WS_SYNCPAINT window, you must call the WinShowWindow function.
- See Also** WinShowWindow

■ WinEndEnumWindows

BOOL WinEndEnumWindows(*henum*)

HENUM *henum*; /* handle of the enumeration list */

The WinEndEnumWindows function ends an enumeration process. This function destroys the snapshot of the window hierarchy captured by the WinBeginEnumWindows function.

- Parameters** *henum* Identifies the enumeration list. This handle must have been created by a previous call to the WinBeginEnumWindows function.
- Return Value** The return value is TRUE if the function is successful or FALSE if an error occurs.
- See Also** WinBeginEnumWindows

■ WinEndPoint

BOOL WinEndPoint (*hps*)

HPS *hps*; /* handle of the presentation space */

The **WinEndPoint** function restores the presentation space to its state prior to the **WinBeginPaint** call. If a cache presentation space was created by the **WinBeginPaint** function, it is released. If the text cursor was hidden by the **WinBeginPaint** function, it is displayed.

Parameters *hps* Identifies the presentation space. This must be the handle of the presentation space returned by a previous call to the **WinBeginPaint** function.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

Example This example calls **WinBeginPaint** when it receives a **WM_PAINT** message to get a presentation-space handle to the update region and to get the coordinates of the update rectangle. It then fills the update rectangle and calls **WinEndPoint** to release the presentation space.

```
case WM_PAINT:
    hps = WinBeginPaint(hwnd, /* handle of the window */
                        NULL, /* get a cache ps */
                        &rc1); /* receives update rectangle */
    WinFillRect(hps, &rc1, CLR_WHITE);
    WinEndPoint(hps);
```

See Also **WinBeginPaint**

■ WinEnumClipbrdFmts

USHORT WinEnumClipbrdFmts (*hab, fmt*)

HAB *hab*; /* handle of the anchor block */

USHORT *fmt*; /* index of last format enumerated */

The **WinEnumClipbrdFmts** function enumerates the list of clipboard data formats available in the clipboard. The clipboard must have been previously opened by calling the **WinOpenClipbrd** function.

Parameters *hab* Identifies the anchor block.

fmt Specifies the index of the last clipboard data format enumerated using this function. This should start at zero, in which case the first available format is obtained. Subsequently, it should be set to the last format index value returned by this function.

Return Value The return value is the index of the next available clipboard data format in the clipboard. When zero is returned, enumeration is complete (there are no further formats available).

See Also **WinOpenClipbrd**

■ WinEnumDlgItem

HWND WinEnumDlgItem(*hwndDlg*, *hwnd*, *code*, *fLock*)

HWND *hwndDlg*; /* handle of the dialog window */

HWND *hwnd*; /* handle of the child window */

USHORT *code*; /* returned dialog item */

BOOL *fLock*; /* lock/unlock flag */

The **WinEnumDlgItem** function returns the handle of a dialog item within a dialog window.

Parameters

hwndDlg Identifies the dialog window that contains the dialog item.

hwnd Identifies the child window of the dialog window. This may be an immediate child window or a window lower in the hierarchy, such as a child of a child window.

code Specifies which dialog item to return. This parameter is one of the following values:

Value	Meaning
EDL_FIRSTGROUPITEM	First item in same group.
EDL_FIRSTTABITEM	First item in dialog window with style WS_TABSTOP. The <i>hwnd</i> window is ignored.
EDL_LASTGROUPITEM	Last item in same group.
EDL_LASTTABITEM	Last item in dialog box with style WS_TABSTOP. The <i>hwnd</i> window is ignored.
EDL_NEXTGROUPITEM	Next item in same group. Wraps to beginning of group when end of group is reached.
EDL_NEXTTABITEM	Next item with style WS_TABSTOP. Wraps around to beginning of dialog-item list when end is reached.
EDL_PREVGROUPITEM	Previous item in same group. Wraps to end of group when start of group is reached.
EDL_PREVTABITEM	Previous item with style WS_TABSTOP. Wraps to end of dialog-item list when beginning is reached.

fLock Specifies whether the dialog item is to be locked or unlocked. If TRUE, the item is locked. If FALSE, it is not.

Return Value

The return value is the item handle obtained by this function, specified by the *code* parameter. The window is always an immediate child window of the window identified by the *hwndDlg* parameter.

Comments

If the dialog item is locked by this function, then you must at some point call the **WinLockWindow** function to unlock the dialog item. The reason for locking the dialog item is so that it cannot be destroyed until you are done using it.

■ WinEqualRect

BOOL WinEqualRect(*hab*, *prcl1*, *prcl2*)

HAB *hab*; /* handle of the anchor block */
PRECTL *prcl1*; /* address of structure for first rectangle */
PRECTL *prcl2*; /* address of structure for second rectangle */

The **WinEqualRect** function compares two rectangles for equality. Equal rectangles have identical coordinates (all sides are the same).

Parameters *hab* Identifies the anchor block.

prcl1 Points to the **RECTL** structure that contains the first rectangle. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

prcl2 Points to the **RECTL** structure that contains the second rectangle.

Return Value The return value is **TRUE** if the rectangles are equal, or **FALSE** if they are not.

■ WinExcludeUpdateRegion

SHORT WinExcludeUpdateRegion(*hps*, *hwnd*)

HPS *hps*; /* handle of the presentation space */
HWND *hwnd*; /* handle of the window */

The **WinExcludeUpdateRegion** function subtracts the update region of a window from the clipping region of a presentation space. If the presentation space does not have a clipping region, one is created. The result of this function is that drawing into the update region of a window will be clipped (will not be drawn). This function is typically used to prevent drawing into parts of a window known to be invalid.

Parameters *hps* Identifies the presentation space whose clipping region is updated.

hwnd Identifies the window whose update region is subtracted from the clipping region of the presentation space.

Return Value The return value is **EXRGN_NULL**, **EXRGN_RECT**, or **EXREGN_COMPLEX** if the function is successful. The return value is **EXRGN_ERROR** if an error occurs.

See Also **GpiCombineRegion**, **WinValidateRect**

WinFillRect

BOOL WinFillRect(*hps*, *prcl*, *clr*)

HPS *hps*; /* handle of the presentation space */
PRECTL *prcl*; /* address of the structure with the rectangle */
COLOR *clr*; /* color of the rectangle */

The **WinFillRect** function fills a rectangular area. It does not change any presentation-space state.

Parameters

hps Identifies the presentation space where the rectangle is drawn.

prcl Points to the **RECTL** structure that contains the coordinates of the rectangle to fill. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

clr Specifies the rectangle color. This parameter is either a color index or an RGB color value, depending on how a logical color table has been loaded. (For more information on color tables, see **GpiCreateLogColorTable**.)

Typically, this parameter will be a color index taken from one of the following values:

CLR_WHITE	CLR_DARKGRAY
CLR_BLACK	CLR_DARKBLUE
CLR_BACKGROUND	CLR_DARKRED
CLR_BLUE	CLR_DARKPINK
CLR_RED	CLR_DARKGREEN
CLR_PINK	CLR_DARKCYAN
CLR_GREEN	CLR_BROWN
CLR_CYAN	CLR_PALEGRAY
CLR_YELLOW	CLR_NEUTRAL
SYSCLR_WINDOWSTATICTEXT	SYSCLR_WINDOWTEXT
SYSCLR_SCROLLBAR	SYSCLR_TITLETEXT
SYSCLR_BACKGROUND	SYSCLR_ACTIVEBORDER
SYSCLR_ACTIVETITLE	SYSCLR_INACTIVEBORDER
SYSCLR_INACTIVETITLE	SYSCLR_APPWORKSPACE
SYSCLR_MENU	SYSCLR_HELPBACKGROUND
SYSCLR_WINDOW	SYSCLR_HELPTEXT
SYSCLR_WINDOWFRAME	SYSCLR_HELPHILITE
SYSCLR_MENUTEXT	SYSCLR_CSYS_COLORS

Return Value

The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

Example

This example calls **WinBeginPaint** to get a presentation-space handle and the coordinates of the update rectangle. It then calls **WinFillRect** to fill the update rectangle.

```
case WM_PAINT:
    hps = WinBeginPaint(hwnd, NULL, &rcl);
    WinFillRect(hps, /* presentation-space handle */
               &rcl, /* coordinates of the rectangle */
               CLR_WHITE); /* color to use */
    WinEndPaint(hps);
```

See Also

GpiCreateLogColorTable

■ WinFindAtom

ATOM WinFindAtom(*hAtomTbl*, *pszAtomName*)

HATOMTBL *hAtomTbl*; /* handle of the atom table */

PSZ *pszAtomName*; /* address of the atom name */

The WinFindAtom function finds an atom in the atom table. This function is identical to the WinAddAtom function, with two exceptions: If the atom name is not found in the table, it is not added to the table and NULL is returned as the value of this function; if the atom name is found in the table, the use count is not increased.

Because integer atoms do not have a use count and do not actually occupy memory in the atom table, this function is identical to WinAddAtom with respect to integer atoms.

Parameters *hAtomTbl* Identifies the atom table. This handle must have been created by a previous call to the WinCreateAtomTable function.

pszAtomName Points to the null-terminated character string containing the atom name. If the string begins with a “#” character, the ASCII digits that follow are converted into an integer atom. If the string begins with an “!” character, the next two bytes are interpreted as an atom. If the high word of this parameter is -1, the low word is an atom and that atom is returned.

Return Value The return value is the atom associated with the passed string, or it is NULL if the string is not in the atom table.

See Also WinAddAtom, WinCreateAtomTable

■ WinFlashWindow

BOOL WinFlashWindow(*hwndFrame*, *fFlash*)

HWND *hwndFrame*; /* handle of the window to flash */

BOOL *fFlash*; /* start/stop flashing flag */

The WinFlashWindow function starts or stops flashing a window. Flashing is achieved by inverting the title bar continuously. A beep is emitted for the first five flashes. If the window has been minimized, the icon text will flash when this function is called.

Parameters *hwndFrame* Identifies the window to flash.

fFlash Specifies whether the window flashes. If TRUE, the window starts flashing. If FALSE, the window stops flashing.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments If an application needs to display a message box, but it does not have the focus, then it should call the WinFlashWindow function to flash its frame window and emit a beep to get the user’s attention. Once it receives the focus, it can call WinFlashWindow again to stop the flashing, and then display its message box.

By using this method, a message box from your application will not be displayed while the user is working with another application.

See Also WinAlarm

■ WinFocusChange

BOOL WinFocusChange (*hwndDesktop*, *hwndSetFocus*, *fsFocusChange*)
HWND *hwndDesktop*; /* handle of the desktop */
HWND *hwndSetFocus*; /* handle of the focus window */
USHORT *fsFocusChange*; /* focus-changing flags */

The **WinFocusChange** function sets the focus to the specified window.

A window may temporarily set the focus to itself without changing the active window or the selection. Examples of this are menu and scroll bar windows. When complete, the window sets the focus back to the previous focus window.

Parameters

hwndDesktop Identifies the desktop window. This parameter can be **HWND_DESKTOP** or the desktop window handle.

hwndSetFocus Identifies the new focus window.

fsFocusChange Specifies the flags that control the focus-changing process. This parameter can be a combination of the following values:

Value	Meaning
FC_NOLOSEACTIVE	Do not send the WM_ACTIVATE message to the window being deactivated.
FC_NOLOSEFOCUS	Do not send the WM_SETFOCUS message to the window losing the focus.
FC_NOLOSESELECTION	Do not send the WM_SETSELECTION message to the window losing the selection.
FC_NOSETACTIVE	Do not send the WM_ACTIVATE message to the window becoming active.
FC_NOSETFOCUS	Do not send the WM_SETFOCUS message to the window receiving the focus.
FC_NOSETSELECTION	Do not send the WM_SETSELECTION message to the window receiving the selection.

By using various combinations of these flags, an application can control activation, selection, focus changes, and other default activities, such as bringing frame windows to the top of their sibling windows. If *fsFocusChange* is zero, the system takes the default action (this is the same as calling the **WinSetFocus** function).

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also WinQueryFocus, WinSetFocus

■ WinFormatFrame

SHORT WinFormatFrame (*hwndFrame*, *prclFrame*, *pswp*, *cswpMax*, *prclClient*)

HWND *hwndFrame*; /* handle of window with frame controls to be formatted */
PRECTL *prclFrame*; /* address of structure with rectangle */
PSWP *pswp*; /* address of array of structures */
USHORT *cswpMax*; /* number of SWP structures */
PRECTL *prclClient*; /* address of client window rectangle */

The **WinFormatFrame** function calculates the size and position of all standard frame controls within a frame window. The data is calculated and returned in an array with one entry for each control window. This function allows an application which has subclassed its frame window, to more easily modify the appearance of a frame window and its controls.

Parameters

hwndFrame Identifies the window whose frame controls are to be formatted.

prclFrame Points to the **RECTL** structure that contains the rectangle where the frame controls are formatted. This typically is the window rectangle identified by the *hwndFrame* parameter, but where the window has a wide border, (for example, as specified by **FS_DLGBORDER**), the rectangle is inset by the size of the border. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

pswp Points to an array of **SWP** structures. There must be one **SWP** structure for each frame control. Typically, the count of frame windows is calculated as follows:

$$(\text{FID_CLIENT} - \text{FID_SYSMENU} + 1)$$

This can change, however, if the frame window is subclassed. The **SWP** structure has the following form:

```
typedef struct _SWP {
    USHORT fs;
    SHORT  cy;
    SHORT  cx;
    SHORT  y;
    SHORT  x;
    HWND   hwndInsertBehind;
    HWND   hwnd;
} SWP;
```

For a full description, See Chapter 4, "Types, Macros, Structures."

cswpMax Specifies the number of **SWP** structures.

prclClient Points to the window rectangle of the **FID_CLIENT** window after formatting. If this parameter is **NULL**, no client window rectangle is returned.

Return Value

The return value is the number of **SWP** structures returned in the array pointed to by *pswp*.

Comments An SWP structure will be filled in for the following frame controls:

Value	Meaning
FID_CLIENT	Identifies the client window.
FID_HORZSCROLL	Identifies the horizontal scroll bar.
FID_MENU	Identifies the system menu.
FID_MINMAX	Identifies the minimum/maximum box.
FID_SYSMENU	Identifies the system menu.
FID_TITLEBAR	Identifies the title bar.
FID_VERTSCROLL	Identifies the vertical scroll bar.

The returned array of SWP structures can be used in the `WinSetMultWindowPos` function to set the position and size of the frame windows.

The `WinFormatFrame` function typically is used by applications that require a nonstandard frame-window layout. This function is called while the `WM_UPDATEFRAME` message is being processed. The application should alter the calculated positions and sizes as required, after returning from this function. Any additional windows added to the standard set can be handled by adding SWP structures to the array, with positions and sizes set as necessary.

See Also `WinSetMultWindowPos`

■ WinFreeErrorInfo

`BOOL WinFreeErrorInfo(perrinfo)`

`PERRINFO perrinfo`; /* address of structure with error-info block */

The `WinFreeErrorInfo` function frees memory allocated for an error information block.

Parameters *perrinfo* Points to the `ERRINFO` structure that contains the error-information block whose memory is to be freed. The `ERRINFO` structure has the following form:

```
typedef struct _ERRINFO {
    USHORT    cbFixedErrInfo;
    ERRORID   idError;
    USHORT    cDetailLevel;
    USHORT    offaoffszMsg;
    USHORT    offBinaryData;
} ERRINFO;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

Return Value The return value is `TRUE` if the function is successful or `FALSE` if an error occurs.

See Also `WinGetErrorInfo`, `WinGetLastError`

■ WinFreeMem

NPBYTE WinFreeMem(*hHeap*, *npMem*, *cbMem*)

HHEAP *hHeap*; /* handle of the heap */
NPBYTE *npMem*; /* address of memory block to free */
USHORT *cbMem*; /* size of the memory to free */

The **WinFreeMem** function frees memory allocated by the **WinAllocMem** function.

Parameters *hHeap* Identifies the heap. This handle must have been created by a previous call to the **WinCreateHeap** function.

npMem Points to the memory block to free. This parameter must have been returned by a previous call to the **WinAllocMem** or **WinReallocMem** function.

cbMem Specifies the size of the memory to free; it must match the allocated size of the block.

Return Value The return value is **NULL** if the function is successful. Otherwise, it returns the *npMem* parameter. This method of returning failure allows for updating the memory pointer and freeing the memory at the same time with a call similar to the following:

```
npMem = WinFreeMem(hHeap, npMem, cbMem);
```

Comments This function does not attempt to coalesce the block being freed with other free blocks. Use the **WinAvailMem** function to coalesce free blocks.

If the heap was created with the **HM_MOVEABLE** option, the value of the *cbMem* parameter is ignored and the value of the size word in the allocated memory block is used instead.

If the heap was created with the **HM_MOVEABLE** and **HM_VALIDSIZE** options, the value of the *cbMem* parameter is checked against the value of the size word and an error is generated if the two values are not the same.

See Also **WinAllocMem**, **WinAvailMem**, **WinCreateHeap**, **WinReallocMem**

■ WinGetClipPS

HPS WinGetClipPS(*hwnd*, *hwndClip*, *fs*)

HWND *hwnd*; /* address of the parent window */
HWND *hwndClip*; /* handle of clipping type */
USHORT *fs*; /* clipping flags */

The **WinGetClipPS** function returns a specially clipped presentation space for a specified window.

Parameters *hwnd* Identifies the parent window.

hwndClip Identifies the type of clipping to perform. This parameter can be one of the following values:

Value	Meaning
HWND_BOTTOM	Clip the last window in the sibling chain and continue clipping until the next window is <i>hwnd</i> or NULL .

Value	Meaning
HWND_TOP	Clip the first window in the sibling chain and continue clipping until the next window is <i>hwnd</i> or NULL.
NULL	Clip all siblings to the window identified by the <i>hwnd</i> parameter.

fs Specifies one or more clipping flags. This parameter can be one of the following values:

Value	Meaning
PSF_CLIPCHILDREN	Clip out all child windows of <i>hwnd</i> . Same as PSF_CLIPWINDOWLIST and the <i>hwndClip</i> parameter equal to the first child window of <i>hwnd</i> .
PSF_CLIPDOWNWARDS	Clip out all windows from <i>hwndClip</i> to the bottom-most sibling window of <i>hwndClip</i> .
PSF_CLIPSIBLINGS	Clip out all sibling windows of <i>hwnd</i> . Same as PSF_CLIPWINDOWLIST and <i>hwndClip</i> equal to NULL.
PSF_CLIPUPWARDS	Clip out all windows from the sibling windows directly in front of <i>hwndClip</i> to the front-most sibling window of <i>hwndClip</i> .
PSF_LOCKWINDOWUPDATE	The presentation space returned is not locked from updating <i>hwnd</i> because of calls to WinLockWindowUpdate.
PSF_PARENTCLIP	Obtain presentation space with a visible region for the parent window, but with window origin set to the origin of <i>hwnd</i> .

Return Value The return value identifies a presentation space if the function is successful, or it is NULL if an error occurs.

See Also WinGetPS, WinLockWindowUpdate

■ WinGetCurrentTime

ULONG WinGetCurrentTime(*hab*)

HAB *hab*; /* handle of the anchor block */

The WinGetCurrentTime function returns the current time.

Parameters *hab* Identifies the anchor block.

Return Value The return value is the system timer count (in milliseconds) from the time the system is restarted.

See Also WinQueryMsgTime

■ WinGetErrorInfo

PERRINFO WinGetErrorInfo(*hab*)

HAB *hab*; /* handle of the anchor block */

The **WinGetErrorInfo** function returns detailed error information.

Parameters *hab* Identifies the anchor block.

Return Value The return value points to an **ERRINFO** structure that contains information about the previous error code for the current thread, or it is **NULL** if no error information is available.

Comments This function allocates a single private segment to contain the **ERRINFO** structure. All pointers to string fields within **ERRINFO** are offsets to memory within that segment. The **ERRINFO** structure has the following form:

```
typedef struct _ERRINFO { /* erri */
    USHORT   cbFixedErrInfo;
    ERRORID  idError;
    USHORT   cDetailLevel;
    USHORT   offaoffszMsg;
    USHORT   offBinaryData;
} ERRINFO;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

The memory allocated by this function is not freed until the returned pointer is passed to the **WinFreeErrorInfo** function.

Like the **WinGetLastError** function, the **WinGetErrorInfo** function releases any saved error information after formatting the error message. If two calls are made to **WinGetErrorInfo** without any intervening calls, the second call will return **NULL** because the saved error information was released by the first call.

See Also **WinFreeErrorInfo**, **WinGetLastError**

■ WinGetKeyState

SHORT WinGetKeyState(*hwndDesktop*, *vkey*)

HWND *hwndDesktop*; /* handle of the desktop */

SHORT *vkey*; /* virtual key */

The **WinGetKeyState** function returns the key state at the time the last message from the queue was posted. This function is used to determine whether a virtual key is up, down, or toggled.

This function can be used to obtain the state of the mouse buttons with the **VK_BUTTON1**, **VK_BUTTON2**, and **VK_BUTTON3** virtual-key codes.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be **HWND_DESKTOP** or the desktop window handle.

vkey Specifies the virtual-key value in the low byte and zero in the high byte.

Return Value If the key is down, the 0x8000 bit is set (less than 0); if the key is up, this bit is not set. If the key is toggled, the 0x0001 bit is set (a key is toggled if it has been pressed an odd number of times since the system was started).

See Also **WinGetPhysKeyState**

■ WinGetLastError

ERRORID WinGetLastError(*hab*)

HAB *hab*; /* handle of the anchor block */

The **WinGetLastError** function returns the error state set by the failure of a Presentation Manager function. This function returns the last nonzero error code and sets the error code to zero.

Parameters *hab* Identifies the anchor block.

Return Value The return value is the last error code.

Comments In multiple-thread applications where there are multiple anchor blocks, errors are stored in the anchor block created by the **WinInitialize** function of the thread. The application must specify the correct anchor-block value for the thread calling **WinGetLastError**.

See Also **WinGetErrorInfo**, **WinInitialize**

■ WinGetMaxPosition

BOOL WinGetMaxPosition(*hwnd*, *pswp*)

HWND *hwnd*; /* handle of the window */

PSWP *pswp*; /* address of structure for maximum window size and position */

The **WinGetMaxPosition** function fills an **SWP** structure with the maximized-window size and position. On return, the **SWP_SIZE** and **SWP_MOVE** flags will have been combined using the **OR** operator into the **fs** field of the **SWP** structure.

Parameters *hwnd* Identifies the window whose maximum size will be retrieved.

pswp Points to the **SWP** structure that retrieves the size and position of a maximized window. The **SWP** structure has the following form:

```
typedef struct _SWP {
    USHORT fs;
    SHORT cy;
    SHORT cx;
    SHORT y;
    SHORT x;
    HWND hwndInsertBehind;
    HWND hwnd;
} SWP;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also **WinGetMinPosition**

■ WinGetMinPosition

BOOL WinGetMinPosition(*hwnd*, *pswp*, *pptl*)

HWND *hwnd*; /* handle of the window */
PSWP *pswp*; /* address of structure with icon information */
PPOINTL *pptl*; /* address of structure with minimum window position */

The **WinGetMinPosition** function fills an **SWP** structure with the minimized-window size and position. On return, the **SWP_SIZE** and **SWP_MOVE** flags will have been combined using the **OR** operator into the **fs** field of the **SWP** structure.

Parameters

hwnd Identifies the window whose minimum size will be retrieved.

pswp Points to the **SWP** structure that will receive the size and position of a minimized-window icon. The **SWP** structure has the following form:

```
typedef struct _SWP {
    USHORT fs;
    SHORT cy;
    SHORT cx;
    SHORT y;
    SHORT x;
    HWND hwndInsertBehind;
    HWND hwnd;
} SWP;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

pptl Points to the **POINTL** structure that specifies the position (in screen coordinates) to place the minimized window. If **NULL**, the system will determine the position. Otherwise, an icon location as near as possible to the specified position is chosen.

The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also

WinGetMaxPosition

■ WinGetMsg

BOOL WinGetMsg(*hab*, *pqmsg*, *hwndFilter*, *msgFilterFirst*, *msgFilterLast*)

HAB *hab*; /* handle of the anchor block */
PQMSG *pqmsg*; /* address of structure with message */
HWND *hwndFilter*; /* window-filter handle */
USHORT *msgFilterFirst*; /* first message */
USHORT *msgFilterLast*; /* last message */

The **WinGetMsg** function retrieves a message from the thread’s message queue, waits if necessary, and returns when a message conforming to the filtering criteria is available.

- Parameters**
- hab* Identifies the anchor block.
 - pqmsg* Points to the QMSG structure that contains a message. The QMSG structure has the following form:


```
typedef struct _QMSG {
    HWND hwnd;
    USHORT msg;
    MPARAM mp1;
    MPARAM mp2;
    ULONG time;
    POINTL ptl;
} QMSG;
```
 - For a full description, see Chapter 4, “Types, Macros, Structures.”
 - hwndFilter* Identifies the window filter.
 - msgFilterFirst* Specifies the first message.
 - msgFilterLast* Specifies the last message.
- Return Value**
- The return value is TRUE if the returned message is not WM_QUIT. The return value is FALSE if the returned message is WM_QUIT.
- Comments**
- Filtering allows an application to process messages in a different order than the one in the queue. Filtering is used so the application can receive messages of a particular type only, rather than receiving other types of messages at an inconvenient point in the logic of the application. For example, when a “mouse button down” message is received, the application can use filtering to wait for the “mouse button up” message without having to process other messages.
- When using filtering, you must ensure that a message satisfying the specification of the filtering parameters can occur; otherwise, the WinGetMsg function cannot completely execute. For example, calling the WinGetMsg function with the *msgFilterFirst* and *msgFilterLast* parameters equal to WM_CHAR and with the *hwndFilter* parameter set to a window handle that does not have the input focus prevents WinGetMsg from returning.
- Keystrokes are passed to the WinTranslateAccel function. This means that accelerator keys are translated into WM_COMMAND or WM_SYSCOMMAND messages and are not seen as WM_CHAR messages by the application.
- The *hwndFilter* parameter limits the returned message to a specific window or its child windows. When *hwndFilter* is NULL, the returned message can be for any window. The message identity is restricted to the range specified by the *msgFilterFirst* and *msgFilterLast* parameters. When *msgFilterFirst* and *msgFilterLast* are both zero, any message satisfies the range constraint. When *msgFilterFirst* is greater than *msgFilterLast*, messages except those whose identities lie between *msgFilterFirst* and *msgFilterLast* can be returned. Messages that do not conform to the filtering criteria remain in the queue.
- When *msgFilterFirst* and *msgFilterLast* are both zero, all messages are returned.
- The constants WM_MOUSEFIRST and WM_MOUSELAST can be used for *msgFilterFirst* and *msgFilterLast* to filter all but mouse messages.
- The constants WM_BUTTONCLICKFIRST and WM_BUTTONCLICKLAST can be used for *msgFilterFirst* and *msgFilterLast* to filter all but mouse button messages.
- The constants WM_DDE_FIRST and WM_DDE_LAST can be used for *msgFilterFirst* and *msgFilterLast* to filter all but dynamic data exchange messages.

Example

This example calls `WinGetMsg` to retrieve a message and `WinDispatchMsg` to send the message.

```
HAB hab;
QMSG qmsg;

while (WinGetMsg(hab,      /* handle to the anchor block      */
               &qmsg,    /* address of the message queue structure */
               NULL,     /* accept messages for any window      */
               0,        /* first message to accept             */
               0))      /* accept all messages                 */
    WinDispatchMsg(hab, &qmsg);
```

See Also

`WinDispatchMsg`, `WinPeekMsg`, `WinPostMsg`, `WinTranslateAccel`, `WinWaitMsg`

■ WinGetNextWindow

HWND `WinGetNextWindow(henum)`

HENUM *henum*; /* handle of the enumeration list */

The `WinGetNextWindow` function obtains the handle of the next window in a specified enumeration list.

The enumeration list details the window hierarchy at the time `WinBeginEnumWindows` was called. Enumeration starts with the top-most child window (listed first) and proceeds through the list each time the function is called, until all windows have been enumerated. Once all windows have been enumerated, the function returns `NULL`. The enumeration then returns to the top of the list and the handle of the top-most child window is returned on the next call.

When a valid window handle is returned, the window is locked by this function. The window must then be unlocked by using the `WinLockWindow` function before `WinGetNextWindow` is called again.

Parameters

henum Identifies the enumeration list. This parameter is created by the `WinBeginEnumWindows` function.

Return Value

The return value is the handle of the next window in the enumeration list, or it is `NULL` if an error occurs.

See Also

`WinBeginEnumWindows`, `WinLockWindow`

■ WinGetPhysKeyState

SHORT `WinGetPhysKeyState(hwndDesktop, sc)`

HWND *hwndDesktop*; /* handle of the desktop */

SHORT *sc*; /* scan code of the key */

The `WinGetPhysKeyState` function returns the physical-key state of the key represented by the scan-code parameter. This function is not synchronized to the processing of input.

Parameters

hwndDesktop Identifies the desktop window. This parameter can be `HWND_DESKTOP` or the desktop window handle.

sc Specifies the scan code of the key.

Return Value The return value is a flag indicating if the key is currently up or down, and whether the key has gone down since the last time `WinGetPhysKeyState` was called. If the high bit is set (0x8000), the key is currently down, otherwise the key is currently up. If the low bit is set (0x0001), the key has gone down since the last time `WinGetPhysKeyState` was called.

See Also `WinGetKeyState`

■ WinGetPS

HPS `WinGetPS(hwnd)`

HWND `hwnd`; /* handle of the window */

The `WinGetPS` function retrieves a cache presentation space that is a cache micro presentation space present in the system. This space can be used for simple drawing operations that do not depend on long-term data being stored in the presentation space.

Parameters `hwnd` Identifies the window to retrieve a presentation space for.

Return Value The return value identifies presentation space that can be used for drawing in the window.

Comments The initial state of the presentation space is the same as that of a presentation space created using the `GpiCreatePS` function. The color table is in default color-index mode. The visible region associated with the presentation space depends on the window and class styles of the window identified by the `hwnd` parameter. The visible region can have one of the following values:

Value	Meaning
<code>WS_CLIPCHILDREN</code>	All the window's child windows are excluded.
<code>WS_CLIPSIBLINGS</code>	All the window's sibling windows are excluded.
<code>WS_PARENTCLIP</code>	The visible region is the same as that of the window's parent window.

Note that any presentation spaces created with the `WinGetPS` function must be released by calling the `WinReleasePS` function. This must be done before the application terminates.

Example This example processes an application-defined message (`IDM_FILL`). It calls `WinGetPS` to get a presentation space to the entire window. It gets the dimensions of the current window, fills the window, and calls `WinReleasePS` to release the presentation space.

```
case IDM_FILL:
    hps = WinGetPS(hwnd); /* get ps for the entire window */
    WinQueryWindowRect(hwnd, &rc1); /* get window dimensions */
    WinFillRect(hps, &rc1, CLR_WHITE); /* clear entire window */
    WinReleasePS(hps); /* release the ps */
    return 0L;
```

See Also `GpiCreatePS`, `WinGetClipPS`, `WinGetScreenPS`, `WinReleasePS`

■ WinGetScreenPS

HPS WinGetScreenPS(*hwndDesktop*)

HWND *hwndDesktop*; /* handle of the desktop */

The **WinGetScreenPS** function returns a presentation space that can be used for drawing anywhere on the screen.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be HWND_DESKTOP or the desktop window handle.

Return Value The return value is a presentation space, or NULL if an error occurs.

Comments When your application finishes using the presentation space, the space should be released by calling the **WinReleasePS** function.

The **WinLockWindowUpdate** function should be used to avoid updating the same part of the screen at the same time.

See Also WinGetPS, WinLockWindowUpdate, WinReleasePS

■ WinGetSysBitmap

HBITMAP WinGetSysBitmap(*hwndDesktop*, *ibm*)

HWND *hwndDesktop*; /* handle of the desktop */

USHORT *ibm*; /* index of the system bitmap */

The **WinGetSysBitmap** function returns a handle to one of the standard bitmaps provided by the system. This bitmap can be used for any of the normal bitmap operations. When your application is done with the bitmap, it should free it by calling **GpiDeleteBitmap**.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be HWND_DESKTOP or the desktop window handle.

ibm Specifies the system-bitmap index value. It can be one of the following values:

Value	Meaning
SBMP_BTNCORNERS	Push button corners.
SBMP_CHECKBOXES	Check box/radio button check mark.
SBMP_CHILDSYSTEMMENU	Smaller version of the system menu bitmap to use in child windows.
SBMP_DRIVE	A symbol used by the file system to indicate a disk drive.
SBMP_FILE	A symbol used by the file system to indicate a file.
SBMP_FOLDER	A symbol used by the file system to show subdirectories.
SBMP_MAXBUTTON	Maximize button.
SBMP_MENUATTACHED	A symbol used to indicate that a menu item has an attached hierarchical menu.

Value	Meaning
SBMP_MENUCHECK	Menu check mark.
SBMP_MINBUTTON	Minimize button.
SBMP_PROGRAM	A symbol used by the file system to indicate that a file is an executable program.
SBMP_RESTOREBUTTON	Restore button.
SBMP_SBDNARROW	Scroll-bar down arrow.
SBMP_SBLFARROW	Scroll-bar left arrow.
SBMP_SBRGARROW	Scroll-bar right arrow.
SBMP_SBUPARROW	Scroll-bar up arrow.
SBMP_SIZEBOX	A symbol used to indicate an area of a window that a user can click to resize the window.
SBMP_SYSMENU	System menu.
SBMP_TREEMINUS	A symbol used by the file system to show that an entry in the directory tree contains no more files.
SBMP_TREEPLUS	A symbol used by the file system to show that an entry in the directory tree contains more files.

Return Value The return value is a handle to a bitmap, or NULL if an error occurs.

See Also GpiDeleteBitmap, WinDrawBitmap

■ WinInflateRect

BOOL WinInflateRect(*hab*, *prcl*, *cx*, *cy*)

HAB *hab*; /* handle of the anchor block */
PRECTL *prcl*; /* address of structure with expanded rectangle */
SHORT *cx*; /* amount to expand width */
SHORT *cy*; /* amount to expand height */

The **WinInflateRect** function expands the coordinates of a rectangle. If the specified expansion values are positive, the rectangle is expanded on all sides. If the specified expansion values are negative, the horizontal expansion value is subtracted from the left and added to the right, and the vertical expansion value is subtracted from the bottom and added to the top.

Parameters *hab* Identifies the anchor block.

prcl Points to the **RECTL** structure that contains the rectangle to be expanded. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG xLeft;
    LONG yBottom;
    LONG xRight;
    LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

cx Specifies the amount of horizontal expansion.

cy Specifies the amount of vertical expansion.

Return Value The return value is always TRUE.

See Also WinOffsetRect

■ WinInitialize

HAB WinInitialize (*fsOptions*)

USHORT *fsOptions*; /* initialization options */

The WinInitialize function initializes a thread for making Presentation Manager function calls. This must be the first Presentation Manager function called by any thread that will be calling Presentation Manager functions.

Parameters *fsOptions* Specifies the initialization options. Currently this parameter must be zero.

Return Value The return value is the handle of an anchor block, or NULL if an error occurred.

Example This example calls WinInitialize so that the thread can use Presentation Manager functions, processes the message loop, and calls WinTerminate when it is finished calling Presentation Manager functions.

```
HAB hab; /* handle to the anchor block */

VOID cdecl main() {
    hab = WinInitialize(NULL);
    . /* any other initialization */
    while (WinGetMsg(hab, &qmsg, NULL, 0, 0))
        WinDispatchMsg(hab, &qmsg);
    WinTerminate(hab);
}
```

See Also WinCreateMsgQueue, WinTerminate

■ WinInSendMsg

BOOL WinInSendMsg (*hab*)

HAB *hab*; /* handle of the anchor block */

The WinInSendMsg function determines whether the current thread is processing a message sent by another thread.

Parameters *hab* Identifies the anchor block.

Return Value The return value is TRUE if the current thread is processing a message sent by another thread, or FALSE if it is not processing a message.

Comments The WinInSendMsg function can be used to tell if a function is being called recursively.

See Also WinIsThreadActive, WinSendMsg

■ WinIntersectRect

BOOL WinIntersectRect(*hab, prclDst, prclSrc1, prclSrc2*)

HAB *hab*; /* handle of the anchor block */
PRECTL *prclDst*; /* address of structure for intersection of rectangles */
PRECTL *prclSrc1*; /* address of structure with first rectangle */
PRECTL *prclSrc2*; /* address of structure with second rectangle */

The **WinIntersectRect** function calculates the intersection of two source rectangles and places the coordinates of the intersection rectangle into the destination rectangle. If the rectangles do not intersect, an empty rectangle (0, 0, 0, 0) is placed into the destination rectangle.

Parameters *hab* Identifies the anchor block.

prclDst Points to the **RECTL** structure that receives the intersection of the rectangles designated by the *prclSrc1* and *prclSrc2* parameters. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

prclSrc1 Points to the **RECTL** structure that contains the first source rectangle.

prclSrc2 Points to the **RECTL** structure that contains the second source rectangle.

Return Value The return value is **TRUE** if the source rectangles intersect, or **FALSE** if they do not.

See Also **WinUnionRect**

■ WinInvalidateRect

BOOL WinInvalidateRect(*hwnd, prcl, flincludeChildren*)

HWND *hwnd*; /* handle of window with changed update region */
PRECTL *prcl*; /* address of structure with rectangle */
BOOL *flincludeChildren*; /* invalidation-scope flag */

The **WinInvalidateRect** function adds a rectangle to a window’s update region. The update region represents the area of the window that must be redrawn.

Parameters *hwnd* Identifies the window whose update region has changed. If this parameter is **HWND_DESKTOP**, this function updates the entire screen.

prcl Points to the **RECTL** structure that contains the coordinates of the rectangle to add to the window’s update region. If this parameter is **NULL**, the entire window is put into the update region. The **RECTL** structure has the following form:


```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

flIncludeChildren Specifies whether child windows of *hwnd* are included in the invalid region. If TRUE, child windows are included. If FALSE, they are not.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments If the window style is WS_SYNCPAINT, the window is redrawn before returning from the WinInvalidateRect function.

If the window style is WS_CLIPCHILDREN and part of the window’s update region overlaps child windows that have the style WS_SYNCPAINT, those child windows are updated before WinInvalidateRect returns.

Example This example gets the dimensions of the window and calls WinInvalidateRect to invalidate the window. The application will be sent a WM_PAINT message with the entire window as the update rectangle.

```
WinQueryWindowRect(hwnd, &rcl);
WinInvalidateRect(hwnd, /* window to invalidate */
                  &rcl, /* invalid rectangle */
                  FALSE); /* do not include children */
```

See Also WinInvalidateRegion

WinInvalidateRegion

BOOL WinInvalidateRegion(*hwnd*, *hrgn*, *flIncludeChildren*)

HWND *hwnd*; /* handle of window with changed update region */
HRGN *hrgn*; /* handle of the region to add */
BOOL *flIncludeChildren*; /* invalidation-scope flag */

The WinInvalidateRegion function adds a region to a window’s update region. The update region represents the area of the window that needs to be redrawn.

Parameters *hwnd* Identifies the window whose update region has changed. If this parameter is HWND_DESKTOP, this function updates the entire screen.

hrgn Identifies the region to be added to the window’s update region. If this parameter is NULL, the entire window is put into the update region.

flIncludeChildren Specifies whether child windows of *hwnd* are included in the invalid region. If TRUE, child windows are included. If FALSE, they are not.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

- Comments** If the window style is `WS_SYNCPAINT`, the window is redrawn before returning from the `WinInvalidateRegion` function.
- If the window style is `WS_CLIPCHILDREN` and part of the window's update region overlaps child windows that have the style `WS_SYNCPAINT`, those child windows are updated before `WinInvalidateRegion` returns.
- See Also** `WinInvalidateRect`

■ WinInvertRect

```

BOOL WinInvertRect(hps, prcl)
HPS hps;          /* handle of the presentation space */
PRECTL prcl;     /* address of structure with rectangle to invert */

```

The `WinInvertRect` function inverts a rectangular area. Inversion is a logical NOT operation and flips the bits of each pel.

- Parameters** *hps* Identifies the presentation space.
- prcl* Points to the `RECTL` structure that contains the coordinates of the rectangle to invert. The `RECTL` structure has the following form:

```

typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;

```

For a full description, see Chapter 4, "Types, Macros, Structures."

- Return Value** The return value is `TRUE` if the function is successful or `FALSE` if an error occurs.
- See Also** `GpiBitBlt`

■ WinIsChild

```

BOOL WinIsChild(hwnd, hwndParent)
HWND hwnd;          /* handle of the child window */
HWND hwndParent;   /* handle of the parent window */

```

The `WinIsChild` function tests whether a specified window is a child of a specified parent window.

- Parameters** *hwnd* Identifies the child window.
- hwndParent* Identifies the parent window.
- Return Value** The return value is `TRUE` if the child window is a descendant of the parent window. The return value is `FALSE` if the child window is not a descendant of the parent or if an error occurs.
- See Also** `WinSetParent`

■ WinIsPhysInputEnabled

BOOL WinIsPhysInputEnabled(*hwndDesktop*)

HWND *hwndDesktop*; /* handle of the desktop */

The **WinIsPhysInputEnabled** function returns the status of hardware input (on or off).

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be **HWND_DESKTOP** or the desktop window handle.

Return Value The return value is **TRUE** if input is enabled or **FALSE** if input is disabled.

See Also **WinEnablePhysInput**

■ WinIsRectEmpty

BOOL WinIsRectEmpty(*hab, prcl*)

HAB *hab*; /* handle of the anchor block */

PRECTL *prcl*; /* address of structure with rectangle to check */

The **WinIsRectEmpty** function tests whether a rectangle is empty. (An empty rectangle is one with no area. The right side is less than or equal to the left and the bottom side is less than or equal to the top.)

Parameters *hab* Identifies the anchor block.

prcl Points to the **RECTL** structure that contains the rectangle to be tested. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG xLeft;
    LONG yBottom;
    LONG xRight;
    LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is **TRUE** if the rectangle is empty or **FALSE** if the rectangle is not empty.

See Also **WinSetRectEmpty**

■ WinIsThreadActive

BOOL WinIsThreadActive(*hab*)

HAB *hab*; /* handle of the anchor block */

The **WinIsThreadActive** function determines whether the active window belongs to the calling thread.

Parameters *hab* Identifies the anchor block of the calling thread.

Return Value The return value is TRUE if the active window belongs to the calling thread, or it is FALSE if the active window does not belong to the thread.

See Also WinInSendMessage

■ WinIsWindow

BOOL WinIsWindow(*hab*, *hwnd*)

HAB *hab*; /* handle of the anchor block */

HWND *hwnd*; /* handle of window to test */

The **WinIsWindow** function determines whether a specified window is valid.

Parameters *hab* Identifies the anchor block.

hwnd Identifies the window whose validity is to be checked.

Return Value The return value is TRUE if the window is valid or FALSE if it is not.

Comments An invalid window is one which has been destroyed. If *hwnd* contains the handle of an invalid window, or it contains the handle of something other than a window, this function will return FALSE.

See Also WinIsWindowEnabled, WinIsWindowVisible

■ WinIsWindowEnabled

BOOL WinIsWindowEnabled(*hwnd*)

HWND *hwnd*; /* handle of window to check */

The **WinIsWindowEnabled** function determines whether a specified window is enabled or disabled.

Parameters *hwnd* Identifies the window to check.

Return Value The return value is TRUE if the window is enabled, or FALSE if it is disabled.

See Also WinEnableWindow, WinIsWindowVisible

■ WinIsWindowVisible

BOOL WinIsWindowVisible(*hwnd*)

HWND *hwnd*; /* handle of the window to test */

The **WinIsWindowVisible** function determines the visibility state of a window.

Parameters *hwnd* Identifies the window to test.

Return Value The return value is TRUE if the specified window and all its child windows have the WS_VISIBLE style bit set. The return value is FALSE if the window is not visible. Because the return value reflects only the visibility state of the window, not the WS_VISIBLE flag, it may be TRUE even if the window is not visible to other windows.

- Comments** 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 displayed as long as the window or any of the windows in the parent chain 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 displayed.
- If the value of the `WS_VISIBLE` style bit has been changed, the message `WM_WINDOWPOSCHANGED` is sent to the window of the `hwnd` parameter before the function returns.
- Drawing to a window with a `WS_VISIBLE` style will not be displayed if the window is covered by other windows, or clipped by its parent.
- See Also** `WinIsWindowEnabled`, `WinShowWindow`

■ WinLoadAccelTable

HACCEL `WinLoadAccelTable(hab, hmod, idAccelTable)`

HAB *hab*; /* handle of the anchor block */

HMODULE *hmod*; /* handle of the module */

USHORT *idAccelTable*; /* accelerator table identifier */

The `WinLoadAccelTable` function loads an accelerator table.

- Parameters** *hab* Identifies the anchor block.
- hmod* Identifies the module that contains the accelerator table. This parameter can be either the module handle returned by the `DosLoadModule` function or `NULL` for the application's module.
- idAccelTable* Identifies the accelerator table.
- Return Value** The return value is the handle of the accelerator table.
- See Also** `DosLoadModule`, `WinCreateAccelTable`, `WinDestroyAccelTable`, `WinSetAccelTable`

■ WinLoadDlg

HWND `WinLoadDlg(hwndParent, hwndOwner, pfnDlgProc, hmod, idDlg, pCreateParams)`

HWND *hwndParent*; /* handle of the parent window */

HWND *hwndOwner*; /* handle of the owner window */

PFNWP *pfnDlgProc*; /* pointer to the dialog procedure */

HMODULE *hmod*; /* handle of resource with dialog template */

USHORT *idDlg*; /* dialog window and template */

PVOID *pCreateParams*; /* address of dialog-procedure data */

The `WinLoadDlg` function creates a dialog window from a dialog template and returns the handle of the dialog window created.

- Parameters** *hwndParent* Identifies the parent window of the dialog window.
- hwndOwner* Identifies the owner window for the dialog window.

pfndlgProc Points to the dialog procedure.

hmod Identifies the module that contains the dialog template. This parameter can be either the module handle returned by the `DosLoadModule` function or NULL for the application's module.

idDlg Identifies the dialog window and the dialog-resource file.

pCreateParams Points to dialog-procedure data (application-specific data passed to the dialog procedure with the `WM_INITDLG` message).

Return Value The return value is the handle of the dialog window created, or it is NULL if an error occurs.

See Also `DosLoadModule`, `WinCreateDlg`, `WinDestroyWindow`, `WinDlgBox`, `WinProcessDlg`, `WinSubstituteStrings`

■ WinLoadMenu

HWND `WinLoadMenu`(*hwndFrame*, *hmod*, *idMenu*)

HWND *hwndFrame*; /* handle of the frame window */

HMODULE *hmod*; /* handle of the module with resource */

USHORT *idMenu*; /* menu template identifier */

The `WinLoadMenu` function creates a menu window from the menu template.

Parameters *hwndFrame* Identifies the frame window for the menu.

hmod Identifies the module that contains the menu template. This parameter can be either the module handle returned by the `DosLoadModule` function or NULL for the application's module.

idMenu Identifies the menu template in the resource identified by the *hmod* parameter.

Return Value The return value is the handle of the menu window.

Comments Menus are created as child windows of the frame window and are initially visible. If the menu contains submenus, these submenus are initially created as object windows that are owned by the menu window. If the submenus contain other submenus, these new submenus are also object windows whose owner is the submenu that contains it. The menu hierarchy is defined by the owner-window chain.

See Also `DosLoadModule`, `WinCreateMenu`, `WinQueryObjectWindow`

■ WinLoadMessage

SHORT WinLoadMessage (*hab, hmod, id, cchMax, pszBuffer*)

HAB *hab*; /* handle of the anchor block */
HMODULE *hmod*; /* module handle */
USHORT *id*; /* message identifier */
SHORT *cchMax*; /* buffer size */
PSZ *pszBuffer*; /* address of buffer for message */

The **WinLoadMessage** function loads a message from a resource, copies the message to the specified buffer, and appends a terminating null character.

Parameters

hab Identifies the anchor block.

hmod Identifies the module that contains the message. This parameter can be either the module handle returned by the **DosLoadModule** function or NULL for the application's module.

id Identifies the message.

cchMax Specifies the size of the buffer.

pszBuffer Points to the buffer that receives the message.

Return Value

The return value is the length of the returned message (excluding the terminating null character). The return value can have a maximum value of (*cchMax* - 1). The return value is zero if an error occurs.

Comments

Message resources contain up to 16 messages each. The resource identifier is calculated from the *id* parameter value passed to **WinLoadMessage** as follows:

$$\text{resource identifier} = (id / 16) + 1$$

To save storage on disk and in memory, applications should group their message resources sequentially, starting at a multiple of 16.

See Also

DosLoadModule, **WinLoadString**

■ WinLoadPointer

HPOINTER WinLoadPointer (*hwndDesktop, hmod, idPtr*)

HWND *hwndDesktop*; /* handle of the desktop */
HMODULE *hmod*; /* handle of the module with the resource */
USHORT *idPtr*; /* resource identifier */

The **WinLoadPointer** function loads a pointer. The pointer can then be used as the mouse pointer by calling the **WinSetPointer** function.

Parameters

hwndDesktop Identifies the desktop window. This parameter can be **HWND_DESKTOP** or the desktop window handle.

hmod Identifies the module that contains the pointer. This parameter can be either the module handle returned by the **DosLoadModule** function or NULL for the application's module.

idPtr Identifies the pointer.

Return Value The return value is a handle to the pointer if the function is successful, or NULL if an error occurs.

Example This example calls `WinLoadPointer` to load an application-defined pointer. When processing the `WM_MOUSEMOVE` message, the loaded pointer is displayed by calling `WinSetPointer`.

```
case WM_CREATE:
    hptrCrossHair = WinLoadPointer(HWND_DESKTOP,
        NULL, /* load from .exe file */
        IDP_CROSSHAIR); /* identifies the pointer */

case WM_MOUSEMOVE:
    WinSetPointer(HWND_DESKTOP, hptrCrossHair);
```

See Also `DosLoadModule`, `WinCreatePointer`, `WinDestroyPointer`, `WinDrawPointer`, `WinQuerySysPointer`, `WinSetPointer`

■ WinLoadString

SHORT `WinLoadString(hab, hmod, id, cchMax, pszBuffer)`

HAB *hab*; /* handle of the anchor block */
HMODULE *hmod*; /* handle of the module with the string */
USHORT *id*; /* string identifier */
SHORT *cchMax*; /* size of the buffer */
PSZ *pszBuffer*; /* address of the buffer for the string */

The `WinLoadString` function loads a string from a resource, copies the string to the specified buffer, and appends a terminating null character.

Parameters *hab* Identifies the anchor block.

hmod Identifies the module that contains the string. This parameter can be either the module handle returned by the `DosLoadModule` function or NULL for the application's module.

id Identifies the string identifier.

cchMax Specifies the size of the supplied buffer.

pszBuffer Points to the buffer that receives the string.

Return Value The return value is the length of the returned string (excluding the terminating null character). The return value can have a maximum value of $(cchMax - 1)$. The return value is zero if an error occurs.

Comments String resources contain up to 16 strings each. The resource identifier is calculated from the *id* value passed to `WinLoadString` as follows:

$$\text{resource identifier} = (id / 16) + 1$$

To save storage on disk and in memory, applications should group their string resources sequentially, starting at a multiple of 16.

See Also `DosLoadModule`, `WinLoadMessage`

■ WinLockHeap

PVOID WinLockHeap(*hHeap*)

HHEAP *hHeap*; /* handle of the heap */

The **WinLockHeap** function returns a far address to the beginning of the heap.

Parameters *hHeap* Identifies the heap. This parameter is returned by the **WinCreateHeap** function.

Return Value The return value is a far pointer to the beginning of the segment that contains the passed heap.

See Also **WinAllocMem**, **WinCreateHeap**

■ WinLockVisRegions

BOOL WinLockVisRegions(*hwndDesktop*, *fLock*)

HWND *hwndDesktop*; /* handle of the desktop */

BOOL *fLock*; /* lock/unlock flag */

The **WinLockVisRegions** function locks or unlocks the visible regions of all windows on the screen, preventing any of the visible regions from changing.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be **HWND_DESKTOP** or the desktop window handle.

fLock Specifies whether to lock or unlock the visible regions. If **TRUE**, the visible regions are locked. If **FALSE**, the visible regions are unlocked.

Comments This function is used to prevent a window's visible regions from changing while a thread performs a screen operation. For example, **WinLockVisRegions** should be used if the application is moving bits from one part of a window to another. Calling **WinLockVisRegions** during this operation ensures that no other window will appear on top of the window that bits are being copied from and therefore no physical change in the bits will take place. Using **WinLockWindowUpdate** for this bit copying will not work, because although no bits will be changed in the locked area, it is still possible that the visible region of the presentation space being used for the bit copying might change.

While the visible regions are locked, no messages should be sent and no functions called that could send messages.

Only one thread can lock the visible regions at any one time. The same thread can call **WinLockVisRegions** multiple times. A lock count is maintained by the system and is incremented each time a locking call is made and decremented each time an unlocking call is made. The visible regions are unlocked when the count is zero.

If **WinLockVisRegions** is called while another thread has locked the visible regions, the function will not return until the thread locking the visible regions has unlocked them.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also **WinLockWindowUpdate**

■ WinLockWindow

HWND WinLockWindow (*hwnd*, *fLock*)

HWND *hwnd*; /* handle of the window */

BOOL *fLock*; /* lock/unlock flag */

The **WinLockWindow** function locks or unlocks a specified window. A window cannot be destroyed while it is locked.

Parameters *hwnd* Identifies the window to be locked or unlocked.

fLock Specifies whether the window is to be locked or unlocked. If TRUE, the window is locked. If FALSE, the window is unlocked.

Return Value The return value is the handle of the window that was locked or unlocked if the function is successful. It is NULL if an error occurred.

Comments If the **WinDestroyWindow** function is called with a locked window handle, the window is not destroyed until the window is unlocked.

A count is maintained of the number of times a window has been locked without a corresponding call to unlock the window. The window cannot be destroyed until the count is zero. The **WinQueryWindowLockCount** function can be called to get the current lock count.

See Also **WinDestroyWindow**, **WinLockWindowUpdate**, **WinQueryWindowLockCount**

■ WinLockWindowUpdate

BOOL WinLockWindowUpdate (*hwndDesktop*, *hwndLockUpdate*)

HWND *hwndDesktop*; /* handle of the desktop */

HWND *hwndLockUpdate*; /* handle of the window to lock/unlock */

The **WinLockWindowUpdate** function prevents or allows the updating of a window and its child windows. While updating is locked, no drawing will take place on the screen. When updating is unlocked, portions of the screen are invalidated and repainted.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be **HWND_DESKTOP** or the desktop window handle.

hwndLockUpdate Identifies the window to be locked. If this parameter is NULL, all windows are unlocked.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also **WinLockVisRegions**

■ WinMapDlgPoints

```

BOOL WinMapDlgPoints(hwndDlg, pptl, cwpt, fCalcWindowCoords)
HWND hwndDlg;          /* handle of the dialog window          */
PPOINTL pptl;          /* address of array of structures with points to map */
USHORT cwpt;          /* number of POINTL structures          */
BOOL fCalcWindowCoords; /* type of points                        */

```

The **WinMapDlgPoints** function converts points of a dialog window from dialog coordinates to window coordinates or from window coordinates to dialog coordinates.

Parameters *hwndDlg* Identifies the dialog window.

pptl Points to the array of **POINTL** structures that contain the points to be converted. The converted points are substituted in the array. The **POINTL** structure has the following form:

```

typedef struct _POINTL {
    LONG   x;
    LONG   y;
} POINTL;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

cwpt Specifies the number of **POINTL** structures in the *pptl* array.

fCalcWindowCoords Specifies the type of points to convert. If **TRUE**, the points are dialog coordinates and are converted to window coordinates relative to the dialog window. If **FALSE**, the points are window coordinates relative to the dialog window and are converted to dialog coordinates.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also **WinMapWindowPoints**

■ WinMapWindowPoints

```

BOOL WinMapWindowPoints(hwndFrom, hwndTo, pptl, cwpt)
HWND hwndFrom;        /* handle of the window to be mapped from      */
HWND hwndTo;          /* handle of the window to be mapped to        */
PPOINTL pptl;        /* address of array of structures with points to map */
SHORT cwpt;          /* number of POINTL structures                  */

```

The **WinMapWindowPoints** function converts a set of points from a coordinate space relative to one window to a coordinate space relative to another window.

- Parameters**
- hwndFrom* Identifies the window from which points are converted. If this parameter is NULL or HWND_DESKTOP, the points are assumed to be in screen coordinates.
- hwndTo* Identifies the window to which points are converted. If this parameter is NULL or HWND_DESKTOP, the points are converted to screen coordinates.
- pptl* Points to an array of POINTL structures that contain the set of points. This parameter can also point to a RECTL structure, in which case the *cwpt* parameter should be set to 2. The POINTL structure has the following form:
- ```
typedef struct _POINTL {
 LONG x;
 LONG y;
} POINTL;
```
- The RECTL structure has the following form:
- ```
typedef struct _RECTL { /* rcl */
    LONG xLeft;
    LONG yBottom;
    LONG xRight;
    LONG yTop;
} RECTL;
```
- For a full description, see Chapter 4, “Types, Macros, Structures.”
- cwpt* Specifies the number of POINTL structures in the *pptl* array.
- Return Value** The return value is TRUE if the function is successful or FALSE if an error occurs.
- See Also** WinMapDlgPoints

■ WinMessageBox

USHORT WinMessageBox(*hwndParent*, *hwndOwner*, *pszText*, *pszCaption*, *idWindow*, *flStyle*)

HWND *hwndParent*; /* handle of the parent window */
HWND *hwndOwner*; /* handle of the owner window */
PSZ *pszText*; /* address of text in message box */
PSZ *pszCaption*; /* address of title of message box */
USHORT *idWindow*; /* message-box identifier */
USHORT *flStyle*; /* type of message box */

The **WinMessageBox** function creates, displays, and operates a message-box window. The message-box window consists of a message and a simple dialog with the user.

- Parameters**
- hwndParent* Identifies the parent window of the newly created message-box window. This parameter is the desktop-window handle, HWND_DESKTOP, or NULL if the message box is a main window.
- hwndOwner* Identifies the owner window of the message-box window. The owner window is activated when **WinMessageBox** returns.

pszText Points to the text displayed as the message within the message-box window. The text will be automatically wrapped as necessary to fit within the message box. The “\n” characters can be used to force a line break, however this is not recommended except between paragraphs, as different fonts could change the appearance of the text.

pszCaption Points to the title of the message-box window. If this parameter is NULL, “Error” (the default title) is displayed. The maximum length of the text is device-dependent. If the text is too long, it will be clipped.

idWindow Identifies the identifier of the message-box window. This value is passed to the HK_HELP hook if the WM_HELP message is received by the message-box window.

flStyle Specifies the type of message-box window created. This parameter consists of a button flag, an icon flag, a default button flag, and any number of special flags. The following four lists describe the available flags which can be combined using the OR operator together for this parameter:

Buttons	Meaning
MB_ABORTRETRYIGNORE	Message box contains Abort, Retry, and Ignore push buttons.
MB_ENTER	Message box contains an Enter push button.
MB_ENTERCANCEL	Message box contains Enter and Cancel push buttons.
MB_OK	Message box contains an OK push button.
MB_OKCANCEL	Message box contains OK and Cancel push buttons.
MB_RETRYCANCEL	Message box contains Retry and Cancel push buttons.
MB_YESNO	Message box contains Yes and No push buttons.
MB_YESNOCANCEL	Message box contains Yes, No, and Cancel push buttons.
Icon	Meaning
MB_ICONASTERISK	Message box contains asterisk icon.
MB_ICONEXCLAMATION	Message box contains exclamation-point icon.
MB_ICONHAND	Message box contains hand icon.
MB_ICONQUESTION	Message box contains question-mark icon.
MB_NOICON	Message box does not contain an icon.

Default button	Meaning
MB_DEFBUTTON1	First button is the default (first button is always the default unless MB_DEFBUTTON2 or MB_DEFBUTTON3 is specified).
MB_DEFBUTTON2	Second button is the default.
MB_DEFBUTTON3	Third button is the default.
Special flags	Meaning
MB_APPLMODAL	Message box is application modal.
MB_SYSTEMMODAL	Message box is system modal.
MB_HELP	Message box contains Help push button.
MB_MOVEABLE	Message box is movable.

Return Value

The return value indicates the user's response to the message. It can be one of the following values:

Value	Meaning
MBID_ABORT	Abort button was selected.
MBID_CANCEL	Cancel button was selected.
MBID_ENTER	Enter button was selected.
MBID_IGNORE	Ignore button was selected.
MBID_NO	No button was selected.
MBID_OK	OK button was selected.
MBID_RETRY	Retry button was selected.
MBID_YES	Yes button was selected.
MDID_ERROR	The WinMessageBox function failed—an error occurred.

If a message box has a Cancel button, MBID_CANCEL is returned if the ESCAPE key is pressed or if the Cancel button is selected. If the message box has no Cancel button, pressing the ESCAPE key has no effect.

Comments

If a message-box window is created as part of the processing of a dialog window, the dialog window should be made the owner of the message-box window.

If a system modal message-box window is created to tell the user that the system is running out of memory, the strings passed into this function should not be taken from a resource file because an attempt to load the resource file may fail due to lack of memory. Such a message-box window can safely use the hand icon (MB_ICONHAND), however, because this icon is always memory-resident.

Example

This example shows a typical use of the `WinMessageBox` function when debugging an application. The C run-time function `sprintf` is used to format the body of the message. In this case, it converts the coordinates of the mouse pointer (retrieved with the `WinQueryPointerPos` function) into a string. The string is then displayed by calling `WinMessageBox`.

```
CHAR szMsg[100];
POINTL ptl;

WinQueryPointerPos(HWND_DESKTOP, &ptl);
sprintf(szMsg, "x = %ld y = %ld", ptl.x, ptl.y);
WinMessageBox(HWND_DESKTOP,
    hwndClient,          /* client-window handle */
    szMsg,               /* body of the message */
    "Debugging information", /* title of the message */
    0,                   /* message box id */
    MB_NOICON | MB_OK); /* icon and button flags */
```

See Also

WinFlashWindow

WinMsgMuxSemWait

USHORT WinMsgMuxSemWait(*pisemCleared*, *pmxsl*, *lTimeOut*)

PUSHORT *pisemCleared*; /* address of variable that receives index number */

PVOID *pmxsl*; /* address of structure with semaphore list */

LONG *lTimeOut*; /* length of time to wait */

The `WinMsgMuxSemWait` function waits for one or more of the specified semaphores to clear. This function checks the specified list. If any of the semaphores are clear, the function returns. Otherwise, the function waits until the specified time elapses or until one of the semaphores in the list clears.

Parameters

pisemCleared Points to the variable that receives the index number of the most recently cleared semaphore.

pmxsl Points to the `MUXSEMLIST` structure containing a semaphore list that defines the semaphores to be cleared. The semaphore list consists of one or more semaphore handles. The `MUXSEMLIST` structure has the following form:

```
typedef struct _MUXSEMLIST {
    USHORT cmxs;
    MUXSEM amxs[16];
} MUXSEMLIST;
```

The structure may contain up to 16 semaphores.

For a full description, see Chapter 4, "Types, Macros, Structures."

lTimeOut Specifies how long to wait for the semaphores to become available. If the value is greater than zero, this parameter specifies the number of milliseconds to wait before returning. If the value is `SEM_IMMEDIATE_RETURN`, the function returns immediately. If the value is `SEM_INDEFINITE_WAIT`, the function waits indefinitely.

- Return Value** The return value is zero if the function is successful. Otherwise, it is an error value, which may be one of the following:
- ERROR_EXCL_SEM_ALREADY_OWNED
 - ERROR_INTERRUPT
 - ERROR_INVALID_EVENT_COUNT
 - ERROR_INVALID_HANDLE
 - ERROR_INVALID_LIST_FORMAT
 - ERROR_SEM_TIMEOUT
 - ERROR_TOO_MANY_MUXWAITERS
- Comments** This function is identical to the `DosMuxSemWait` function with the following exceptions: Window messages sent via the `WinSendMsg` function by another thread may be received; the function can wait for 15 semaphores simultaneously (`DosMaxSemWait` can wait for 16).
- This function should be used to wait for a semaphore when the semaphore owner may need to issue a `WinSendMsg` function (or another `Win` function that implicitly issues a `WinSendMsg` function) before clearing the semaphore.
- See Also** `DosMuxSemWait`, `WinMsgSemWait`, `WinSendMsg`

■ WinMsgSemWait

USHORT WinMsgSemWait(*hsem*, *lTimeOut*)

HSEM *hsem*; /* handle of the semaphore */

LONG *lTimeOut*; /* time-out value */

The `WinMsgSemWait` function waits for a specified semaphore to be cleared. `WinMsgSemWait` waits until a thread uses the `DosSemClear` function to clear the semaphore or until a time-out occurs. If no previous thread has set the semaphore, `WinMsgSemWait` returns immediately.

- Parameters**
- hsem* Identifies the semaphore to set. This value can be the handle of a system semaphore that has been previously created or opened by using the `DosCreateSem` or `DosOpenSem` function, or it can be the address of a RAM semaphore.
- lTimeOut* Specifies how long to wait for the semaphore to clear. If the value is greater than zero, this parameter specifies the number of milliseconds to wait before returning. If the value is `SEM_IMMEDIATE_RETURN`, the function returns immediately. If the value is `SEM_INDEFINITE_WAIT`, the function waits indefinitely.

- Return Value** The return value is zero if the function is successful. Otherwise, it is an error value, which may be one of the following:

- ERROR_EXCL_SEM_ALREADY_OWNED
- ERROR_INTERRUPT
- ERROR_INVALID_HANDLE
- ERROR_SEM_TIMEOUT

Comments This function is identical to **DosSemWait** except that in addition to waiting on the specified semaphore, window messages sent via the **WinSendMsg** function by another thread may be received.

See Also **DosCreateSem**, **DosOpenSem**, **DosSemClear**, **DosSemWait**, **WinMsgMuxSemWait**, **WinSendMsg**

■ WinMultWindowFromIDs

SHORT WinMultWindowFromIDs(*hwndParent*, *phwnd*, *idFirst*, *idLast*)

HWND *hwndParent*; /* handle of the parent window */

PHWND *phwnd*; /* address of array of window handles */

USHORT *idFirst*; /* first window identifier in range */

USHORT *idLast*; /* last window identifier in range */

The **WinMultWindowFromIDs** function finds the handles of specified child windows that have window-identifier values within a specified range. This function may be used to enumerate all the items in a dialog group, for example, or to enumerate all the frame controls of a standard window. This function is faster than making individual calls to the **WinWindowFromID** function.

Parameters *hwndParent* Identifies the parent window.

phwnd Points to the array that contains the window handles. This array must contain (*idLast* - *idFirst* + 1) elements; the zero-based index of a window in the array is (*idWindow* - *idFirst*), where *idWindow* is the window identifier of the specified window. If there is no window for a window identifier within the range, the corresponding element in the array is NULL.

idFirst Identifies the first window-identifier value in the range (inclusive).

idLast Identifies the last window-identifier value in the range (inclusive).

Return Value The return value is the number of window handles returned in the array. The return value is zero if no window handles are returned.

See Also **WinWindowFromID**

■ WinNextChar

PSZ WinNextChar(*hab*, *idcp*, *idcc*, *psz*)

HAB *hab*; /* handle of the anchor block */

USHORT *idcp*; /* code page */

USHORT *idcc*; /* country code */

PSZ *psz*; /* address of character in string */

The **WinNextChar** function moves to the next character in a string.

Parameters *hab* Identifies the anchor block.

idcp Identifies the code page.

idcc Identifies the country code.

psz Points to a character in a null-terminated string.

Return Value The return value points to the next character in the string or the NULL terminating character.

See Also WinPrevChar

■ WinOffsetRect

BOOL WinOffsetRect(*hab, prcl, cx, cy*)

HAB *hab*; /* handle of the anchor block */
PRECTL *prcl*; /* address of the structure with rectangle */
SHORT *cx*; /* horizontal offset */
SHORT *cy*; /* vertical offset */

The **WinOffsetRect** function offsets a rectangle by adding a specified value to both the left and right coordinates and adding a specified value to both the top and bottom coordinates.

Parameters *hab* Identifies an anchor block.

prcl Points to the **RECTL** structure that contains the rectangle to be offset. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

cx Specifies the value of the horizontal offset. This value is added to the left and right sides of the rectangle.

cy Specifies the value of the vertical offset. This value is added to the top and bottom sides of the rectangle.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also WinInflateRect

■ WinOpenClipbrd

BOOL WinOpenClipbrd(*hab*)

HAB *hab*; /* handle of the anchor block */

The **WinOpenClipbrd** function opens the clipboard and prevents other threads and processes from examining or changing the clipboard contents. If another thread or process already has the clipboard open, this function does not return until the clipboard is closed.

Messages can be received from other threads and processes during the processing of this function.

Parameters *hab* Identifies the anchor block.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinCloseClipbrd, WinEnumClipbrdFmts

■ WinOpenWindowDC

HDC WinOpenWindowDC(*hwnd*)

HWND *hwnd*; /* handle of the window */

The WinOpenWindowDC function opens a device context for a window.

Parameters *hwnd* Identifies the window with the device context.

Return Value The return value is the handle of the device context.

Comments Only one device context can be opened for each window.

A handle to a device context is used to associate a presentation space with the window.

The window device context is automatically closed when its associated window is destroyed. It must not be closed with the DevCloseDC function.

This function is used by applications that typically have a lot of state information associated with a presentation space, such as coordinate mapping transforms, attributes, fonts, etc. This interface provides a “global” presentation space for a given window. This global can be kept in the application until the window is destroyed.

See Also WinQueryWindowDC

■ WinPeekMsg

BOOL WinPeekMsg(*hab*, *pqmsg*, *hwndFilter*, *msgFilterFirst*, *msgFilterLast*, *fs*)

HAB *hab*; /* handle of the anchor block */

PQMSG *pqmsg*; /* address of structure */

HWND *hwndFilter*; /* handle of the filter window */

USHORT *msgFilterFirst*; /* first message */

USHORT *msgFilterLast*; /* last message */

USHORT *fs*; /* status of message in queue */

The WinPeekMsg function inspects the thread’s message queue.

Parameters *hab* Identifies the anchor block.

pqmsg Points to the QMSG structure. The QMSG structure has the following form:

```
typedef struct _QMSG {
    HWND hwnd;
    USHORT msg;
    MPARAM mp1;
    MPARAM mp2;
    ULONG time;
    POINTL ptl;
} QMSG;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

hwndFilter Identifies the window filter.

msgFilterFirst Specifies the first message.

msgFilterLast Specifies the last message.

fs Specifies whether to remove the message from the queue. If this parameter is `PM_REMOVE`, the message is removed from the queue. If this parameter is `PM_NOREMOVE`, the message remains in the queue. An application should specify only one of these flags. If neither flag is specified, the message is not removed. If both are specified, the message is removed.

Return Value	The return value is <code>TRUE</code> if a message is available, or it is <code>FALSE</code> if no message is available.
Comments	<p>This function is identical to the <code>WinGetMsg</code> function, except that it does not wait for the arrival of a message and allows for leaving the message in the queue.</p> <p>The constants <code>WM_MOUSEFIRST</code> and <code>WM_MOUSELAST</code> can be used for <i>msgFilterFirst</i> and <i>msgFilterLast</i> to filter all but mouse messages.</p> <p>The constants <code>WM_BUTTONCLICKFIRST</code> and <code>WM_BUTTONCLICKLAST</code> can be used for <i>msgFilterFirst</i> and <i>msgFilterLast</i> to filter all but mouse button messages.</p> <p>The constants <code>WM_DDE_FIRST</code> and <code>WM_DDE_LAST</code> can be used for <i>msgFilterFirst</i> and <i>msgFilterLast</i> to filter all but dynamic data exchange messages.</p>
See Also	<code>WinGetMsg</code>

■ WinPostMsg

BOOL `WinPostMsg`(*hwnd*, *msg*, *mp1*, *mp2*)

HWND *hwnd*; /* handle of the window to post message to */

USHORT *msg*; /* message */

MPARAM *mp1*; /* first message parameter */

MPARAM *mp2*; /* second message parameter */

The `WinPostMsg` function posts a message to the message queue for the specified window.

Parameters	<p><i>hwnd</i> Identifies the window to post the message to. If this parameter is <code>NULL</code>, the message is posted to the queue associated with the current thread.</p> <p><i>msg</i> Specifies the message.</p> <p><i>mp1</i> Specifies the first message parameter.</p> <p><i>mp2</i> Specifies second message parameter.</p>
-------------------	---

Return Value The return value is `TRUE` if the function is successful or `FALSE` if an error occurs.

Comments The following are some of the differences between `WinPostMsg` and `WinSendMsg`:

- **WinPostMsg** returns immediately. **WinSendMsg** waits for the receiver to return.
- A thread that does not have a message queue can still call **WinPostMsg**. It cannot call **WinSendMsg**.
- Calling **WinSendMsg** to send a message to another thread is costly in terms of CPU time. This is not true of the **WinPostMsg**.

See Also WinBroadcastMsg, WinGetMsg, WinPeekMsg, WinSendMsg

■ WinPostQueueMsg

BOOL WinPostQueueMsg(*hmq, msg, mp1, mp2*)

HMQ *hmq*; /* handle of the message queue */
USHORT *msg*; /* message */
MPARAM *mp1*; /* first message parameter */
MPARAM *mp2*; /* second message parameter */

The **WinPostQueueMsg** function posts a message to a message queue. This function can be used to post messages to any queue in the system.

Parameters

- hmq* Identifies the message queue.
- msg* Specifies the message.
- mp1* Specifies the first message parameter.
- mp2* Specifies the second message parameter.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs or if the queue was full.

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

See Also WinPostMsg, WinSendMsg

■ WinPrevChar

PSZ WinPrevChar(*hab, idcp, idcc, pszStart, psz*)

HAB *hab*; /* handle of the anchor block */
USHORT *idcp*; /* code page */
USHORT *idcc*; /* country code */
PSZ *pszStart*; /* address of string with character */
PSZ *psz*; /* address of character in string */

The **WinPrevChar** function moves to the previous character in a string.

Parameters

- hab* Identifies the anchor block.
- idcp* Identifies the code page.
- idcc* Identifies the country code.

pszStart Points to the character string that contains the character pointed to by the *psz* parameter.

psz Points to a character in the string pointed to by the *pszStart* parameter.

Return Value The return value points to the previous character in a string, or to the first character if the *psz* parameter equals the *pszStart* parameter.

See Also WinNextChar

■ WinProcessDlg

USHORT WinProcessDlg(*hwndDlg*)

HWND *hwndDlg*; /* handle of the dialog queue */

The **WinProcessDlg** function processes messages intended for a dialog window. This function does not return until the **WinDismissDlg** function is called by the dialog procedure.

Parameters *hwndDlg* Identifies a dialog window.

Return Value The return value is set to the value returned by the **WinDismissDlg** function.

See Also WinDismissDlg, WinDlgBox, WinLoadDlg

■ WinPtInRect

BOOL WinPtInRect(*hab, prcl, pptl*)

HAB *hab*; /* handle of the anchor block */

PRECTL *prcl*; /* address of structure with rectangle coordinates */

PPOINTL *pptl*; /* address of structure with point coordinates */

The **WinPtInRect** function determines whether a point lies within a rectangle.

Parameters *hab* Identifies an anchor block.

prcl Points to a **RECTL** structure containing the rectangle to be checked. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

pptl Points to a **POINTL** structure containing the point to be checked. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG   x;
    LONG   y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is TRUE if the point lies within the rectangle, or FALSE if the point is outside the rectangle.

Example This example processes a WM_BUTTON1UP message, converts the mouse pointer coordinates into a POINTL structure, and calls WinPtInRect to determine if the mouse was clicked in the predefined global rectangle.

```
RECTL rclGlobal; /* global set to some predefined rectangle */
POINTL ptl;
HPS hps;

case WM_BUTTON1UP:
    ptl.x = (LONG) SHORT1FROMMP (mp1);
    ptl.y = (LONG) SHORT2FROMMP (mp1);
    if (WinPtInRect (hab, /* anchor-block handle */
                    &rclGlobal, /* address of the rectangle */
                    &ptl) { /* address of the point */
```

See Also GpiPtInRegion

■ WinQueryAccelTable

HACCEL WinQueryAccelTable (*hab*, *hwndFrame*)

HAB *hab*; /* handle of the anchor block */

HWND *hwndFrame*; /* handle of the frame window */

The WinQueryAccelTable function queries the window or queue accelerator table.

Parameters *hab* Identifies an anchor block.

hwndFrame Identifies the frame window. This parameter can be NULL.

Return Value The return value is an accelerator-table handle if the function is successful, or NULL if an error occurred.

Comments If the *hwndFrame* parameter is NULL, then the handle of the queue accelerator is returned. Otherwise, the handle of the window accelerator table is returned by sending the WM_QUERYACCELTABLE message to the frame window specified by *hwndFrame*.

See Also WinCreateAccelTable

■ WinQueryActiveWindow

HWND WinQueryActiveWindow (*hwndDesktop*, *fLock*)

HWND *hwndDesktop*; /* handle of the desktop */

BOOL *fLock*; /* lock/unlock flag */

The WinQueryActiveWindow function retrieves the active frame window.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be HWND_DESKTOP or the desktop window handle.

fLock Specifies whether the active window should be locked. If TRUE, the window is locked. If FALSE, it is not.

- Return Value** The return value is the active window handle or NULL if no window was active at the time of the call or the *hwndDesktop* handle is invalid.
- Comments** If this function is called while the active window is changing, then it may return NULL, indicating that no window was active at the time of the call. Since a NULL value can also be returned if the *hwndDesktop* handle is invalid, you must call **WinGetLastError** to determine if a NULL return value is caused by an invalid *hwndDesktop* handle or because the active window was changing when you made the call.
- If the active window is locked by this function, then you must at some point call the **WinLockWindow** function to unlock the window. The reason for locking the window is so that the window cannot be destroyed until you are done using it.
- See Also** **WinLockWindow**, **WinQueryFocus**

■ WinQueryAtomLength

USHORT WinQueryAtomLength(*hAtomTbl*, *atom*)

HATOMTBL *hAtomTbl*; /* handle of the atom table */

ATOM *atom*; /* atom */

The **WinQueryAtomLength** function queries the length of a string associated with the passed atom. The purpose of this function is to allow an application to determine the size of the buffer to pass to the **WinQueryAtomName** function.

- Parameters** *hAtomTbl* Identifies an atom table. This handle must have been created by a previous call to the **WinCreateAtomTable** function.
- atom* Specifies the atom whose length is to be returned.

Return Value The return value is the length of the string associated with the atom, not including the null terminating byte. If the specified atom or atom table is invalid, the return value is zero. Integer atoms always return a length of 6.

See Also **WinCreateAtomTable**, **WinQueryAtomName**

■ WinQueryAtomName

USHORT WinQueryAtomName(*hAtomTbl*, *atom*, *pszBuffer*, *cchBufferMax*)

HATOMTBL *hAtomTbl*; /* handle of the atom table */

ATOM *atom*; /* atom */

PSZ *pszBuffer*; /* address of the buffer */

USHORT *cchBufferMax*; /* length of the buffer */

The **WinQueryAtomName** function retrieves an atom name associated with an atom.

- Parameters** *hAtomTbl* Identifies an atom table. This handle must have been created by a previous call to the **WinCreateAtomTable** function.
- atom* Specifies an atom identifying the character string to retrieve. For integer atoms, the format of the string is *#dddd*, where *dddd* are decimal digits in the system code page (which will be an ASCII code page). No leading zeros are generated, and the length can be from three to seven characters.

pszBuffer Points to the buffer to receive the character string.

cchBufferMax Specifies the maximum size (in bytes) of the buffer pointed to by *pszBuffer*.

Return Value The return value is the actual number of bytes copied to the buffer, excluding the null terminating byte. If the specified atom or the atom table is invalid, the return value is zero.

See Also WinCreateAtomTable, WinFindAtom, WinQueryAtomLength

■ WinQueryAtomUsage

USHORT WinQueryAtomUsage (*hAtomTbl*, *atom*)

HATOMTBL *hAtomTbl*; /* handle of the atom table */

ATOM *atom*; /* atom */

The **WinQueryAtomUsage** function returns the number of times an atom has been used.

Parameters *hAtomTbl* Identifies an atom table. This handle must have been created by a previous call to the **WinCreateAtomTable** function.

atom Specifies the atom whose use count is to be returned.

Return Value The return value is the use count of the atom. It is 0xFFFF for integer atoms. If the atom table or atom is invalid, then the return value is zero.

See Also WinAddAtom, WinCreateAtomTable

■ WinQueryCapture

HWND WinQueryCapture (*hwndDesktop*, *fLock*)

HWND *hwndDesktop*; /* handle of the desktop */

BOOL *fLock*; /* lock/unlock flag */

The **WinQueryCapture** function returns the window handle of the window that has the mouse capture.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be **HWND_DESKTOP** or the desktop window handle.

fLock Specifies whether the window that has the mouse capture should be locked. If **TRUE**, the window is locked. If **FALSE**, it is not.

Return Value The return value is the window handle with the mouse capture, or **NULL** if no window has the capture or an error occurred.

Comments If the window that has the mouse capture is locked by this function, then you must at some point call the **WinLockWindow** function to unlock the window. The reason for locking the window is so that the window cannot be destroyed until you are done using it.

See Also WinLockWindow, WinSetCapture

■ WinQueryClassInfo

BOOL WinQueryClassInfo (*hab*, *pszClassName*, *pcls*)

HAB *hab*; /* handle of the anchor block */
PSZ *pszClassName*; /* address of the class name */
PCLASSINFO *pcls*; /* address of structure for class information */

The **WinQueryClassInfo** function retrieves window class information. This function is used in creating subclasses of a given class.

Parameters *hab* Identifies an anchor block.

pszClassName Points to a null-terminated string containing the class name. The class name is either an application-specified name as defined by the **WinRegisterClass** function or the name of a preregistered WC class.

pcls Points to a **CLASSINFO** structure that will contain the retrieved information about the class. The **CLASSINFO** structure has the following form:

```
typedef struct _CLASSINFO {
    ULONG    flClassStyle;
    PFNWP    pfnWindowProc;
    USHORT   cbWindowData;
} CLASSINFO;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is TRUE if the function is successful. Otherwise, it is FALSE, indicating that the class does not exist.

See Also **WinRegisterClass**

■ WinQueryClassName

SHORT WinQueryClassName (*hwnd*, *cchMax*, *psz*)

HWND *hwnd*; /* handle of the window */
SHORT *cchMax*; /* length of the buffer */
PSZ *psz*; /* address of the buffer */

The **WinQueryClassName** function copies the window class name, as a null-terminated string, into a buffer.

If the class name is longer than (*cchMax* - 1), only the first (*cchMax* - 1) characters of the class name are copied.

If the specified window is of any of the preregistered WC classes, the class name returned is in the form #*nnnnn*, where *nnnnn* is up to five digits that correspond to the low word of the WC class-name constant.

Parameters *hwnd* Identifies a window.

cchMax Specifies the length of the buffer pointed to by the *psz* parameter.

psz Points to a buffer for the class name.

Return Value The return value is the length of the returned class name, not including the null termination character.

See Also **WinQueryClassInfo**, **WinRegisterClass**

■ WinQueryClipbrdData

ULONG WinQueryClipbrdData(*hab*, *fmt*)

HAB *hab*; /* handle of the anchor block */
USHORT *fmt*; /* specifies the format of the data */

The **WinQueryClipbrdData** function obtains a handle to the current clipboard data having a specified format.

The returned data handle cannot 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 **WinCloseClipbrd** is called.

The application should not free the data handle itself nor leave it locked in any way.

Parameters *hab* Identifies an anchor block.
 fmt Specifies the format of the data that is accessed by this function.

Return Value The return value is a handle to the data in the clipboard in the format specified by the *fmt* parameter. If the format does not exist or if an error occurred, the return value is NULL.

Comments For a description of the possible formats and data handles, see the **WinSetClipbrdData** function.

See Also **WinCloseClipbrd**, **WinOpenClipbrd**, **WinSetClipbrdData**

■ WinQueryClipbrdFmtInfo

BOOL WinQueryClipbrdFmtInfo(*hab*, *fmt*, *pfsFmtInfo*)

HAB *hab*; /* handle of the anchor block */
USHORT *fmt*; /* specifies data format */
PUSHORT *pfsFmtInfo*; /* receives memory model and usage flags */

The **WinQueryClipbrdFmtInfo** function determines whether a particular format of data is present in the clipboard and, if so, provides information about that format.

Parameters *hab* Identifies an anchor block.
 fmt Specifies the format of the data that this function queries. The following list describes the standard clipboard formats:

Value	Meaning
CF_BITMAP	The data is a bitmap. The CFI_HANDLE memory-model flag must be set in the <i>pfsFmtInfo</i> parameter.
CF_DSPBITMAP	The data is a bitmap representation of a private data format. The clipboard viewer can use this format to display a private format. The memory-model flag CFI_HANDLE must be set in the <i>pfsFmtInfo</i> parameter.

Value	Meaning
CF_METAFILE	The data is a metafile. The CFI_HANDLE memory-model flag must be set in the <i>pfsFmtInfo</i> parameter.
CF_DSPMETAFILE	The data is a metafile representation of a private data format. The clipboard viewer can use this format to display a private format. The memory-model flag CFI_HANDLE must be set in the <i>pfsFmtInfo</i> parameter.
CF_TEXT	The data is an array of text characters, which may include newline characters to mark line breaks. The null character indicates the end of the text data. The CFI_SELECTOR memory-model flag must be set in the <i>pfsFmtInfo</i> parameter.
CF_DSPTEXT	The data is a textual representation of a private data format. The clipboard viewer can use this format to display a private format. The memory-model flag CFI_SELECTOR must be set in the <i>pfsFmtInfo</i> parameter.

pfsFmtInfo Points to a variable that receives the memory-model and usage (CFI) flags. It consists of a memory-model flag and a usage flag from the following lists:

Memory-model flag	Meaning
CFI_SELECTOR	Handle is a selector plus zero offset to a segment in storage.
CFI_HANDLE	Handle is the handle to a metafile or bitmap.
Usage flag	Meaning
CFI_OWNERFREE	Handle is not freed by the <code>WinEmptyClipbrd</code> function. The application must free the data if necessary.
CFI_OWNERDISPLAY	Format will be drawn by the clipboard owner in the clipboard-viewer window by means of the <code>WM_PAINTCLIPBOARD</code> message. The <i>ulData</i> parameter should be NULL.

- Return Value** The return value is TRUE if the format exists. Otherwise, it is FALSE, indicating that the format does not exist.
- Comments** For a description of the possible formats and data handles, see the `WinSetClipbrdData` function.
- See Also** `WinOpenClipbrd`, `WinSetClipbrdData`

■ WinQueryClipbrdOwner

HWND WinQueryClipbrdOwner(*hab*, *fLock*)

HAB *hab*; /* handle of the anchor block */

BOOL *fLock*; /* lock/unlock viewer flag */

The **WinQueryClipbrdOwner** function obtains the handle of the window that currently owns the clipboard (if any).

Parameters *hab* Identifies an anchor block.

fLock Specifies whether the clipboard owner window should be locked. If TRUE, the window is locked. If FALSE, it is not.

Return Value The return value is the window handle of the current clipboard owner. If the clipboard is not owned by any window or if an error occurred, the return value is NULL.

Comments If the clipboard owner window is locked by this function, then you must at some point call the **WinLockWindow** function to unlock the window. The reason for locking the window is so that the window cannot be destroyed until you are done using it.

See Also **WinLockWindow**, **WinQueryClipbrdViewer**, **WinSetClipbrdOwner**

■ WinQueryClipbrdViewer

HWND WinQueryClipbrdViewer(*hab*, *fLock*)

HAB *hab*; /* handle of the anchor block */

BOOL *fLock*; /* lock/unlock viewer flag */

The **WinQueryClipbrdViewer** function obtains the handle of the current clipboard viewer window (if any).

Parameters *hab* Identifies an anchor block.

fLock Specifies whether the clipboard owner window should be locked. If TRUE, the window is locked. If FALSE, it is not.

Return Value The return value is the handle of the current clipboard viewer window. If the clipboard does not have a current viewer window or if an error occurred, the return value is NULL.

Comments If the clipboard owner window is locked by this function, then you must at some point call the **WinLockWindow** function to unlock the window. The reason for locking the window is so that the window cannot be destroyed until you are done using it.

See Also **WinLockWindow**, **WinQueryClipbrdOwner**, **WinSetClipbrdViewer**

■ WinQueryCp

USHORT WinQueryCp(*hmq*)

HMQ *hmq*; /* handle of the message queue */

The **WinQueryCp** function retrieves the code page for the specified message queue.

Parameters *hmq* Identifies a message queue.

Return Value The return value is the code page for the specified message queue if the function is successful. Otherwise, it is zero, indicating that an error occurred.

See Also **DosGetCp, GpiQueryCp, VioGetCp**

■ WinQueryCpList

USHORT WinQueryCpList(*hab, ccpMax, pacp*)

HAB *hab*; /* handle of the anchor block */

USHORT *ccpMax*; /* maximum number of code pages to retrieve */

PUSHORT *pacp*; /* address of array to receive code pages */

The **WinQueryCpList** function obtains available code pages.

Parameters *hab* Identifies an anchor block.

ccpMax Specifies the maximum number of code pages that will be returned.

pacp Points to an array that will receive the available code pages. It will include all code pages available to the **Gpi** and **Vio** functions, including any EBCDIC ones. This list is a superset of those code pages available to the **WinSetCp** function and the **Dos** functions.

Return Value The return value is the the total number of code pages in the system.

See Also **GpiSetCp, VioGetCp, WinQueryCp, WinSetCp**

■ WinQueryCursorInfo

BOOL WinQueryCursorInfo(*hwndDesktop, pcsri*)

HWND *hwndDesktop*; /* handle of the desktop */

PCURSORINFO *pcsri*; /* address of structure for cursor information */

The **WinQueryCursorInfo** function retrieves information about the current cursor.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be **HWND_DESKTOP** or the desktop window handle.

pcsri Points to a **CURSORINFO** structure that receives information about the current cursor. The values are equivalent to the parameters to the **WinCreateCursor** function, except that the *fs* field never includes the **CURSOR_SETPOS** flag. The size and position of the cursor are returned in window coordinates relative to the window identified by the *hwnd* field of the structure. The **CURSORINFO** structure has the following form:

```
typedef struct _CURSORINFO {
    HWND    hwnd;
    SHORT   x;
    SHORT   y;
    SHORT   cx;
    SHORT   cy;
    USHORT  fs;
    RECT    rclClip;
} CURSORINFO;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is TRUE if the cursor exists, or FALSE if it does not.

See Also WinCreateCursor, WinDestroyCursor, WinShowCursor

■ WinQueryDefinition

```
USHORT WinQueryDefinition(hab, hProgHandle, ppib, cbMax)
HAB hab; /* handle of the anchor block */
HPROGRAM hProgHandle; /* handle of the program */
PIBSTRUCT ppib; /* address of structure for program information */
USHORT cbMax; /* length of ppib buffer */
```

The WinQueryDefinition function retrieves information about a program or program group.

Parameters *hab* Identifies the anchor block.

hProgHandle Identifies the program or group.

ppib Points to a PIBSTRUCT structure that receives the program information data. If the *hProgHandle* parameter is a group handle, only the program type and program title fields are significant. The structure has the following form:

```
typedef struct _PIBSTRUCT {
    PROCTYPE prog;
    CHAR      szTitle[MAXNAME+1];
    CHAR      szIconFileName[MAXPATH+1];
    CHAR      szExecutable[MAXPATH+1];
    CHAR      szStartupDir[MAXPATH+1];
    XYWINSIZE xywinInitial;
    USHORT    res1;
    LHANDLE   res2;
    USHORT    cchEnvironmentVars;
    PCH       pchEnvironmentVars;
    USHORT    cchProgramParameter;
    PCH       pchProgramParameter;
} PIBSTRUCT;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

cbMax Specifies the maximum length of data (in bytes) that can be returned in the data structure pointed to by *ppib*. If *cbMax* is zero, this function returns the number of bytes in the program information block.

Return Value The return value is the length of the data actually returned in the data structure, or zero if an error occurred.

If the target is a program rather than a program group, the data returned in *ppib* is in a format usable by the WinAddProgram function.

See Also WinAddProgram

■ WinQueryDesktopWindow

HWND WinQueryDesktopWindow(*hab*, *hdc*)

HAB *hab*; /* handle of the anchor block */

HDC *hdc*; /* handle of the device context */

The **WinQueryDesktopWindow** function retrieves the desktop window handle.

Parameters *hab* Identifies an anchor block.

hdc Identifies a device context. For MS OS/2, version 1.1, this parameter must be NULL.

Return Value The return value is the desktop window handle or NULL if the device does not support windowing.

Comments For most **Win** calls the constant **HWND_DESKTOP** can be used for the desktop window handle.

See Also **WinCreateWindow**, **WinQueryObjectWindow**

■ WinQueryDlgItemShort

BOOL WinQueryDlgItemShort(*hwndDlg*, *idItem*, *pResult*, *fSigned*)

HWND *hwndDlg*; /* handle of the dialog box */

USHORT *idItem*; /* dialog-item identifier */

PSHORT *pResult*; /* address of variable for result */

BOOL *fSigned*; /* signed/unsigned flag */

The **WinQueryDlgItemShort** function translates the text of a dialog item into an integer value. This function is useful in translating a numerical input field into a numeric value for further processing.

Parameters *hwndDlg* Identifies a dialog-box window.

idItem Identifies the dialog item whose text is translated. The dialog-item text is assumed to be an ASCII string.

pResult Points to the integer value resulting from the translation.

fSigned Specifies whether the item text is treated as signed or unsigned. If this parameter is **TRUE**, the item text is treated as signed, in which case the translation checks for a minus sign in the text. If it is **FALSE**, the item text is treated as unsigned.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

Comments If the text string does not contain a valid representation of a number, as in "size," or "-3" when *fSigned* is **FALSE**, then the return value will be **FALSE**.

See Also **WinSetDlgItemShort**

■ WinQueryDlgItemText

USHORT WinQueryDlgItemText(*hwndDlg, idItem, cchBufferMax, pszBuf*)

HWND *hwndDlg*; /* handle of the dialog box */

USHORT *idItem*; /* identifies the dialog item */

SHORT *cchBufferMax*; /* size of the buffer */

PSZ *pszBuf*; /* address of the buffer */

The **WinQueryDlgItemText** function retrieves the text associated with the specified dialog item.

Parameters

- hwndDlg* Identifies the dialog box.
- idItem* Identifies the dialog item.
- cchBufferMax* Specifies the maximum number of characters to place in the buffer pointed to by the *pszBuf* parameter.
- pszBuf* Points to a buffer that receives the dialog item text.

Return Value The return value is the length of the dialog item text, or zero if an error occurred.

See Also WinQueryDlgItemTextLength

■ WinQueryDlgItemTextLength

SHORT WinQueryDlgItemTextLength(*hwndDlg, idItem*)

HWND *hwndDlg*; /* handle of the dialog box */

USHORT *idItem*; /* dialog-item identifier */

The **WinQueryDlgItemTextLength** function retrieves the length of the dialog item text, not including any null termination character.

This function sends a **WM_QUERYFRAMEINFO** message to the window identified by the *hwndDlg* parameter.

Parameters

- hwndDlg* Identifies the dialog box.
- idItem* Identifies the dialog item.

Return Value The return value is the length of the dialog item text, or zero if an error occurred.

See Also WinQueryDlgItemText

■ WinQueryFocus

HWND WinQueryFocus(*hwndDesktop, fLock*)

HWND *hwndDesktop*; /* handle of the desktop */

BOOL *fLock*; /* lock/unlock flag */

The **WinQueryFocus** function returns the handle of the window that currently has the focus.

Parameters	<i>hwndDesktop</i> Identifies the desktop window. This parameter can be <code>HWND_DESKTOP</code> or the desktop window handle. <i>fLock</i> Specifies whether the focus window should be locked. If <code>TRUE</code> , the window is locked. If <code>FALSE</code> , it is not.
Return Value	The return value is a handle to the focus window or <code>NULL</code> if there is no focus window or an error occurs.
Comments	If the focus window is locked by this function, then you must at some point call the <code>WinLockWindow</code> function to unlock the window. The reason for locking the window is so that the window cannot be destroyed until you are done using it.
See Also	<code>WinFocusChange</code> , <code>WinLockWindow</code> , <code>WinQueryActiveWindow</code> , <code>WinSetFocus</code>

■ WinQueryMsgPos

`BOOL WinQueryMsgPos(hab, pptl)`

`HAB hab`; /* handle of the anchor block */
`PPOINTL pptl`; /* address of structure for pointer position */

The `WinQueryMsgPos` function retrieves the pointer position, in screen coordinates, when the last message obtained from the current message queue was posted. To obtain the current position of the pointer, use the `WinQueryPointerPos` function.

Parameters *hab* Identifies an anchor block.
pptl Points to a `POINTL` structure that receives the pointer position in screen coordinates. The pointer position is the same as that in the `ptl` field of the `QMSG` structure. The `POINTL` structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is `TRUE` if the function is successful or `FALSE` if an error occurs.

See Also `WinQueryPointerPos`

■ WinQueryMsgTime

`ULONG WinQueryMsgTime(hab)`

`HAB hab`; /* handle of the anchor block */

The `WinQueryMsgTime` function retrieves the message time for the last message retrieved by the `WinGetMsg` or `WinPeekMsg` function from the current message queue.

The message time is the time, in milliseconds, when the message was posted. The time value is the same as that in the `time` field of the `QMSG` structure.

You cannot assume that time values are always increasing. Since the time value

- Parameters** *hptr* Identifies a pointer.
pptri Points to a **POINTERINFO** structure that receives information about the mouse pointer. The **POINTERINFO** structure has the following form:
- ```
typedef struct _POINTERINFO {
 BOOL fPointer;
 SHORT xHotspot;
 SHORT yHotspot;
 HBITMAP hbmPointer;
} POINTERINFO;
```
- For a full description, see Chapter 4, “Types, Macros, Structures.”
- Return Value** The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.
- See Also** **WinQueryPointerPos**

## ■ WinQueryPointerPos

```
BOOL WinQueryPointerPos(hwndDesktop, pptl)
HWND hwndDesktop; /* handle of desktop window */
PPOINTL pptl; /* address of structure for pointer position */
```

The **WinQueryPointerPos** function retrieves the mouse pointer position. The position returned is the position of the pointer at the time **WinQueryPointerPos** is called and is not synchronized with the **WinGetMsg** and **WinPeekMsg** functions. Use the **WinQueryMsgPos** function to get the pointer position of the last message obtained via **WinGetMsg** or **WinPeekMsg**.

- Parameters** *hwndDesktop* Identifies the desktop window. This parameter can be **HWND\_DESKTOP** or the desktop window handle.  
*pptl* Points to a **POINTL** structure that receives the pointer position in screen coordinates. The **POINTL** structure has the following form:
- ```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

- Return Value** The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.
- Comments** The position retrieved is in screen coordinates, not window coordinates. You can convert screen coordinates to window coordinates with the **WinMapWindowPoints** function.
- See Also** **WinGetMsg**, **WinPeekMsg**, **WinQueryMsgPos**, **WinQueryPointerInfo**

■ WinQueryProfileData

BOOL WinQueryProfileData(*hab*, *pszAppName*, *pszKeyName*, *pvBuf*, *cbBuf*)

HAB *hab*; /* handle of the anchor block */
PSZ *pszAppName*; /* address of the application name */
PSZ *pszKeyName*; /* address of the keyname */
PVOID *pvBuf*; /* address of the buffer */
PUSHORT *cbBuf*; /* length of the buffer */

The **WinQueryProfileData** function retrieves binary data from the *os2.ini* file. Where the data is located is determined by an application name and a keyname which are passed to the function.

Parameters *hab* Identifies an anchor block.

pszAppName Points to a null-terminated text string that contains the name of the application. Its length must be less than 1024 bytes, including the null termination character. The application name is always case-dependent.

pszKeyName Points to a null-terminated text string that contains the keyname. Its length must be less than 1024 bytes, including the null termination character. If *pszKeyName* is NULL, all keynames and their data are deleted. The keyname is always case-dependent.

pvBuf Points to a buffer that receives the data.

cbBuf Points to a variable that contains the size of the buffer pointed to by *pvBuf*. When the function returns, this variable will contain the actual number of bytes placed into the buffer.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments You can find out the size of the data prior to calling this function by calling **WinQueryProfileSize**.

See Also **WinQueryProfileSize**, **WinWriteProfileData**

■ WinQueryProfileInt

SHORT WinQueryProfileInt(*hab*, *pszAppName*, *pszKeyName*, *sError*)

HAB *hab*; /* handle of the anchor block */
PSZ *pszAppName*; /* address of the application name */
PSZ *pszKeyName*; /* address of the keyname */
SHORT *sError*; /* value returned if keyname not found */

The **WinQueryProfileInt** function retrieves an integer from the *os2.ini* file. Where the integer is located is determined by an application name and a keyname which are passed to this function. The integer must have been previously stored as a text string using the **WinWriteProfileString** function. For example, a text string stored as "123" would be returned as the integer 123. The text string may contain a leading minus sign if the number is negative.

Parameters *hab* Identifies the anchor block.

pszAppName Points to a null-terminated text string that contains the name of the application. Its length must be less than 1024 bytes, including the null termination character. The application name is always case-dependent.

pszKeyName Points to a null-terminated text string that contains the keyname. Its length must be less than 1024 bytes, including the null termination character. If *pszKeyName* is NULL, all keynames and their data are deleted. The keyname is always case-dependent.

sError Specifies the error value returned if the keyname (*pszKeyName*) cannot be found.

Return Value The return value is the integer representation of the text string. If the keyname cannot be found, the error value specified by *sError* is returned.

See Also WinQueryProfileData, WinWriteProfileString

■ WinQueryProfileSize

USHORT WinQueryProfileSize (*hab*, *pszAppName*, *pszKeyName*, *pcb*)

HAB *hab*; /* handle of the anchor block */
PSZ *pszAppName*; /* points to the application name */
PSZ *pszKeyName*; /* points to the keyname */
PUSHORT *pcb*; /* points to variable with length of the data */

The **WinQueryProfileSize** function retrieves the size of the data stored at a specified location in the *os2.ini* file. Where the data is located is determined by an application name and a keyname which are passed to this function. This function is typically called prior to calling **WinQueryProfileData** in order to determine how much memory to allocate for the data.

Parameters *hab* Identifies an anchor block.

pszAppName Points to a null-terminated text string that contains the name of the application. Its length must be less than 1024 bytes, including the null termination character. The application name is always case-dependent.

pszKeyName Points to a null-terminated text string that contains the keyname. Its length must be less than 1024 bytes, including the null termination character. If *pszKeyName* is NULL, all keynames and their data are deleted. The keyname is always case-dependent.

pcb Points to a variable that will receive the length of the data. If an error occurs, the length will not be returned.

Return Value The return value is zero if the function is successful. Otherwise, it is an error value.

See Also WinQueryProfileData, WinQueryProfileString

■ WinQueryProfileString

USHORT WinQueryProfileString(*hab, pszAppName, pszKeyName, pszError, pszBuf, cchBuf*)

HAB *hab*; /* handle of the anchor block */
PSZ *pszAppName*; /* points to the application name */
PSZ *pszKeyName*; /* points to the keyname */
PSZ *pszError*; /* points to a default string */
PSZ *pszBuf*; /* address of the buffer for the string */
USHORT *cchBuf*; /* size of buffer */

The **WinQueryProfileString** function retrieves a string from the *os2.ini* file. Where the string is located is determined by an application name and a keyname which are passed to this function.

Parameters

hab Identifies an anchor block.

pszAppName Points to a null-terminated text string that contains the name of the application. Its length must be less than 1024 bytes, including the null termination character. The application name is always case-dependent.

pszKeyName Points to a null-terminated text string that contains the keyname. Its length must be less than 1024 bytes, including the null termination character. If *pszKeyName* is NULL, all keynames and their data are deleted. The keyname is always case-dependent.

pszError Points to a null-terminated string that is placed in *pszBuf* if the key is not found.

pszBuf Points to a buffer that will receive the null-terminated string.

cchBuf Specifies the length of the buffer (*pszBuf*). If the retrieved string is longer than this value, it will be truncated.

Return Value

The return value is the number of characters in the buffer pointed to by *pszBuf*.

See Also

WinWriteProfileString

■ WinQueryProgramTitles

USHORT WinQueryProgramTitles(*hab, hGroup, paproge, cbBuf, pcTitles*)

HAB *hab*; /* handle of the anchor block */
HPROGRAM *hGroup*; /* handle of the group */
PPROGRAMENTRY *paproge*; /* array of PROGRAMENTRY structures */
USHORT *cbBuf*; /* length of paproge buffer */
PUSHORT *pcTitles*; /* receives number of titles */

The **WinQueryProgramTitles** function obtains information about programs within a specified program group.

This function can be used to find out the number of entries within a group by passing a buffer of zero bytes. The function will return the total number of entries within the group.

The list of returned program entries may contain group handles. This allows the tree structure to be built by the caller. Note, though, that information from only one level of the tree structure is returned by this call.

The handle specified can also be a program handle, in which case the buffer will contain only the entry for one program. Thus, this call can be used to get the program title.

Parameters

hab Identifies the anchor block.

hGroup Identifies the group for which information is returned. This handle is either the handle of a program group or SGH_ROOT for the root group.

paproge Points to a storage area where the program information is returned. This is an array of PROGRAMENTRY structures. The PROGRAMENTRY structure has the following form:

```
typedef struct _PROGRAMENTRY {
    HPROGRAM hprog;
    PROGTYP prog;
    CHAR      szTitle[MAXNAMEL+1];
} PROGRAMENTRY;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

cbBuf Specifies the total length (in bytes) of the area pointed to by the *paproge* parameter. Values of *cbBuf* less than the size of a PROGRAMENTRY structure are invalid.

pcTitles Points to a variable that receives the count of the available titles. If the *hGroup* parameter is SGH_ROOT or SGH_MASTER and the buffer length specified in the *cbBuf* parameter is too small to hold all the titles, the return value is zero, none of the titles are copied to the buffer, and *pcTitles* contains the number of titles available. If *hGroup* is a program handle, both the return value and *pcTitles* are the number of handles available.

Return Value

The return value is the count of available titles. If an error occurred, the return value is zero.

See Also

WinAddProgram

■ WinQueryQueueInfo

BOOL WinQueryQueueInfo(*hmq, pmqi, cbCopy*)

HMQ *hmq*; /* handle of the message queue */

PMQINFO *pmqi*; /* address of structure for queue information */

USHORT *cbCopy*; /* number of bytes of information to copy */

The WinQueryQueueInfo function is used to obtain information about a specified queue, such as the process and thread identifier associated with the queue, the maximum number of messages the queue can hold, and the queue procedure address.

Parameters

hmq Identifies the message queue. This handle must either have been created by a previous call to WinCreateMsgQueue, or it must be HMQ_CURRENT to specify the message queue of the thread that is calling this function.

pmqi Points to an MQINFO structure that will receive information about the message queue. This MQINFO structure has the following form:


```
typedef struct _MQINFO {
    USHORT cb;
    PID pid;
    TID tid;
    USHORT cmsgs;
    PVOID pReserved;
} MQINFO;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

cbCopy Specifies the number of bytes of data that will be copied into the MQINFO structure. Normally, it should be set to the length of the structure.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments An important side effect of this function is that a DLL procedure or hook can use this function to find out if the current thread has a message queue associated with it.

See Also WinCreateMsgQueue

■ WinQueryQueueStatus

```
ULONG WinQueryQueueStatus(hwndDesktop)
HWND hwndDesktop; /* handle of the desktop */
```

The WinQueryQueueStatus function returns a code that indicates the status of the message queue associated with the current queue.

This function is very fast and is typically used inside loops to determine whether the WinGetMsg or WinPeekMsg function should be called to process input.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be HWND_DESKTOP or the desktop window handle.

Return Value The high word of the return value indicates the types of messages currently in the queue. The low word of the return value shows the types of messages added to the queue since the last call to WinQueryQueueStatus that are still in the queue.

The following list describes the types of messages that may be in the queue:

Value	Meaning
QS_KEY	A WM_CHAR message is in the queue.
QS_MOUSE	A WM_MOUSEMOVE or WM_BUTTON message is in the queue.
QS_MOUSEBUTTON	A WM_BUTTON message is in the queue.
QS_MOUSEMOVE	A WM_MOUSEMOVE message is in the queue.
QS_PAINT	A WM_PAINT message is in the queue.
QS_POSTMSG	A posted message other than those listed above is in the queue.
QS_SEM1	A WM_SEM1 message is in the queue.
QS_SEM2	A WM_SEM2 message is in the queue.
QS_SEM3	A WM_SEM3 message is in the queue.

Value	Meaning
QS_SEM4	A WM_SEM4 message is in the queue.
QS_SENDMSG	A message sent by another application is in the queue.
QS_TIMER	A WM_TIMER message is in the queue.

See Also WinGetMsg, WinPeekMsg

■ WinQuerySysColor

COLOR WinQuerySysColor(*hwndDesktop*, *clr*, *lReserved*)
HWND *hwndDesktop*; /* handle of the desktop */
COLOR *clr*; /* color index of color to retrieve */
LONG *lReserved*; /* reserved */

The WinQuerySysColor function retrieves a specified system color.

Parameters

hwndDesktop Identifies the desktop window. This parameter can be HWND_DESKTOP or the desktop window handle.

clr Specifies the system color-index value to be returned. This parameter must be one of the SYSCLR index values. For a description of the possible color indexes, see the WinSetSysColors function.

lReserved Reserved; must be zero.

Return Value

The return value is the RGB value corresponding to the index value specified by the *clr* parameter.

See Also

WinSetSysColors

■ WinQuerySysModalWindow

HWND WinQuerySysModalWindow(*hwndDesktop*, *fLock*)
HWND *hwndDesktop*; /* handle of the desktop */
BOOL *fLock*; /* lock/unlock flag */

The WinQuerySysModalWindow function returns the current system modal window.

Parameters

hwndDesktop Identifies the desktop window. This parameter can be HWND_DESKTOP or the desktop window handle.

fLock Specifies whether the system modal window should be locked during processing. If this parameter is TRUE, the window is locked. If FALSE, it is not.

Return Value

The return value is the handle of the current system modal window. If there is none, the return value is NULL.

Comments If the system modal window is locked by this function, then you must at some point call the `WinLockWindow` function to unlock the window. The reason for locking the window is so that the window cannot be destroyed until you are done using it.

See Also `WinLockWindow`, `WinSetSysModalWindow`

■ WinQuerySysPointer

HPOINTER `WinQuerySysPointer(hwndDesktop, iptr, fLoad)`

HWND *hwndDesktop*; /* handle of the desktop */
SHORT *iptr*; /* system-pointer identifier */
BOOL *fLoad*; /* load/unload flag */

The `WinQuerySysPointer` function returns the system pointer handle.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be `HWND_DESKTOP` or the desktop window handle.

iptr Specifies the system pointer from the following list. The appearance of system pointers is implementation dependent.

Value	Meaning
<code>SPTR_APPICON</code>	Standard application icon
<code>SPTR_ARROW</code>	Normal arrow pointer
<code>SPTR_ICONERROR</code>	Exclamation-mark icon
<code>SPTR_ICONINFORMATION</code>	Hand icon
<code>SPTR_ICONQUESTION</code>	Question-mark icon
<code>SPTR_ICONWARNING</code>	Note icon
<code>SPTR_MOVE</code>	Move pointer
<code>SPTR_SIZENESW</code>	Upward sloping double-headed arrow
<code>SPTR_SIZENS</code>	Vertical double-headed arrow
<code>SPTR_SIZENWSE</code>	Downward sloping double-headed arrow
<code>SPTR_SIZEWE</code>	Horizontal double-headed arrow
<code>SPTR_TEXT</code>	Text I-beam pointer
<code>SPTR_WAIT</code>	Hourglass pointer

fLoad Specifies whether to copy the system pointer. If this parameter is `TRUE`, the system pointer is copied and the handle to the new pointer is returned. If it is `FALSE`, then the system pointer handle is returned. You should specify `TRUE` if you intend to modify an existing pointer.

Return Value The return value is a pointer handle.

Example This example calls `WinQuerySysPointer` to get a handle to the system pointer, and then loads an application-defined pointer. After it is done using the application-defined pointer, it restores it to the system pointer.

```
/* get the system pointer */  
hptrDefault = WinQuerySysPointer(HWND_DESKTOP, SPTR_ARROW, FALSE);  
/* load an application-defined pointer */  
hptrCrossHair = WinLoadPointer(HWND_DESKTOP, NULL, IDP_CROSSHAIR);  
/* change the pointer to the application pointer */  
WinSetPointer(HWND_DESKTOP, hptrCrossHair);  
/* restore the system pointer */  
WinSetPointer(HWND_DESKTOP, hptrDefault);
```

See Also `WinQueryPointer`, `WinQueryPointerInfo`

■ WinQuerySystemAtomTable

HATOMTBL WinQuerySystemAtomTable(VOID)

The `WinQuerySystemAtomTable` function returns the handle of the system atom table. This allows two different applications to share an atom table.

Parameters This function has no parameters.

Return Value The return value is the handle of the system atom table.

See Also `WinCreateAtomTable`

■ WinQuerySysValue

LONG WinQuerySysValue(hwndDesktop, iSysValue)

HWND hwndDesktop; /* handle of the desktop */

SHORT iSysValue; /* system value to retrieve */

The `WinQuerySysValue` function retrieves a specified system value.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be `HWND_DESKTOP` or the desktop window handle.

iSysValue Specifies the system value.

Return Value The return value is the system value if the function was successful, or zero if an error occurred.

Comments

The following list describes the system values:

Value	Meaning
SV_CMOUSEBUTTONS	Specifies the number of mouse buttons: 1, 2, or 3.
SV_MOUSEPRESENT	Specifies whether the mouse is present. A value of TRUE means the mouse is present.
SV_SWAPBUTTON	Specifies if the mouse buttons are swapped. TRUE if mouse buttons are swapped.
SV_CXDBLCLK	Specifies the mouse double click horizontal spacing. The horizontal spatial requirement for considering two mouse clicks a double click is met if the horizontal distance between two mouse clicks is less than this value.
SV_CYDBLCLK	Specifies the mouse double click vertical spacing. The vertical spatial requirement for considering two mouse clicks a double click is met if the vertical distance between two mouse clicks is less than this value.
SV_DBLCLKTIME	Specifies the mouse double click time in milliseconds. The temporal requirement for considering two mouse clicks a double click is met if the time between two mouse clicks is less than this value.
SV_CXSIZEBORDER	Specifies the count of pels along the x-axis in the left and right parts of a window sizing border.
SV_CYSIZEBORDER	Specifies the count of pels along the y-axis in the top and bottom sections of a window sizing border.
SV_ALARM	Specifies whether calls to WinAlarm generate a sound. A value of TRUE means sound is generated.
SV_CURSORRATE	Specifies the cursor blinking rate in milliseconds. The blinking rate is the time that the cursor remains visible or invisible. Twice this value is the length of a complete cursor visible/invisible cycle.
SV_FIRSTSCROLLRATE	Specifies the delay (in milliseconds) until scroll bar autorepeat activity begins when the mouse is held down on a scroll bar arrow or within a scroll bar.
SV_SCROLLRATE	Specifies the delay (in milliseconds) between scroll bar autorepeat events.
SV_NUMBEREDLISTS	Reserved.

Value	Meaning
SV_ERRORFREQ	Specifies the frequency (in cycles per second) of a WinAlarm WA_ERROR sound.
SV_NOTEFREQ	Specifies the frequency (in cycles per second) of a WinAlarm WA_NOTE sound.
SV_WARNINGFREQ	Specifies the frequency (in cycles per second) of a WinAlarm WA_WARNING sound.
SV_ERRORDURATION	Specifies the duration (in milliseconds) of a WinAlarm WA_ERROR sound.
SV_NOTEDURATION	Specifies the duration (in milliseconds) of a WinAlarm WA_NOTE sound.
SV_WARNINGDURATION	Specifies the duration (in milliseconds) of a WinAlarm WA_WARNING sound.
SV_CXSCREEN	Specifies the count of pels along the screen's x-axis.
SV_CYSCREEN	Specifies the count of pels along the screen's y-axis.
SV_CXVSCROLL	Specifies the count of pels along the x-axis of a vertical scroll bar.
SV_CYHSCROLL	Specifies the count of pels along the y-axis of a horizontal scroll bar.
SV_CXHSCROLLARROW	Specifies the count of pels along the x-axis of a horizontal scroll bar arrow.
SV_CYVSCROLLARROW	Specifies the count of pels along the y-axis of a vertical scroll bar arrow.
SV_CXBORDER	Specifies the count of pels along the x-axis of a window border.
SV_CYBORDER	Specifies the count of pels along the y-axis of a window border.
SV_CXDLGFRAME	Specifies the count of pels along the x-axis of a dialog frame.
SV_CYDLGFRAME	Specifies the count of pels along the y-axis of a dialog frame.
SV_CYTITLEBAR	Specifies the count of pels along the y-axis of a title-bar window.
SV_CXHSLIDER	Specifies the count of pels along the x-axis of a horizontal scroll bar slider.
SV_CYVSLIDER	Specifies the count of pels along the y-axis of a vertical scroll bar slider.
SV_CXMINMAXBUTTON	Specifies the width (in pels) of a minimize/maximize button.
SV_CYMINMAXBUTTON	Specifies the height (in pels) of a minimize/maximize button.

Value	Meaning
SV_CYMENU	Specifies the height (in pels) of an action-bar menu.
SV_CXFULLSCREEN	Specifies the count of pels along the x-axis of a maximized frame window's client window.
SV_CYFULLSCREEN	Specifies the count of pels along the y-axis of a maximized frame window's client window.
SV_CXICON	Specifies the count of pels along an icon's x-axis.
SV_CYICON	Specifies the count of pels along an icon's y-axis.
SV_CXPOINTER	Specifies the count of pels along the mouse pointer's x-axis.
SV_CYPOINTER	Specifies the count of pels along the mouse pointer's y-axis.
SV_DEBUG	Reserved.
SV_CURSORLEVEL	Specifies the cursor display count. The cursor is visible only when the display count is zero.
SV_POINTERLEVEL	Specifies the mouse pointer display count. The mouse is visible only when the display count is zero.
SV_TRACKRECTLEVEL	Specifies the tracking rectangle display count. The tracking rectangle is visible only when the display count is zero.
SV_CTIMERS	Specifies the number of available timers.
SV_CXBYTEALIGN	Set by a device driver at initialization time to indicate any horizontal alignment that is more efficient for the driver.
SV_CYBYTEALIGN	Set by a device driver at initialization time to indicate any vertical alignment that is more efficient for the driver.
SV_CSYSVALUES	Specifies the number of system values.

See Also

WinSetSysValue

■ WinQueryTaskTitle

USHORT WinQueryTaskTitle (*idProcess*, *pszTitle*, *cbTitle*)

USHORT *idProcess*; /* identifies the process */

PSZ *pszTitle*; /* address of the buffer */

USHORT *cbTitle*; /* length of the buffer */

The **WinQueryTaskTitle** function obtains the title under which a specified application was started or added to the switch list. If this function is used after a switch-list entry is created for the application, the title in the switch-list entry is obtained.

This function is useful when an application should use the same name in its window title and its entry in the switch list as the end-user invokes to start the application. This provides a visual link for the user.

Parameters	<p><i>idProcess</i> Identifies the application whose title is requested.</p> <p><i>pszTitle</i> Points to the buffer to receive the title. The received string will be null-terminated.</p> <p><i>cbTitle</i> Specifies the length, in bytes, of the <i>pszTitle</i> buffer. If the retrieved title is longer than this length, it will be truncated.</p>
Return Value	The return value is zero if the function is successful. Otherwise, it is an error value.
Comments	The length of the title is guaranteed not to exceed MAXNAMEL bytes, plus one for the null-terminating character.
See Also	WinAddSwitchEntry

■ WinQueryUpdateRect

```

BOOL WinQueryUpdateRect(hwnd, prcl)
HWND hwnd; /* handle of the window */
PRECTL prcl; /* address of structure for update rectangle */

```

The **WinQueryUpdateRect** function retrieves the rectangle that bounds the update region of a specified window. This function, in conjunction with the **WinValidateRect** function, is useful for implementing an incremental update scheme as an alternative to the **WinBeginPaint** and **WinEndPaint** functions. You can use the returned update rectangle as the clip region for a presentation space so that drawing output can be clipped to the window's update region.

Parameters	<p><i>hwnd</i> Identifies the window whose update rectangle is retrieved.</p> <p><i>prcl</i> Points to a RECTL structure that receives the coordinates of the rectangle bounding the window's update region. The RECTL structure has the following form:</p>
-------------------	--

```

typedef struct _RECTL {
    LONG xLeft;
    LONG yBottom;
    LONG xRight;
    LONG yTop;
} RECTL;

```

For a full description, see Chapter 4, "Types, Macros, Structures."

Return Value	The return value is TRUE if the function is successful or FALSE if an error occurs.
See Also	WinBeginPaint, WinEndPaint, WinQueryUpdateRegion, WinValidateRect

■ WinQueryUpdateRegion

SHORT WinQueryUpdateRegion(*hwnd*, *hrgn*)

HWND *hwnd*; /* handle of the window */

HRGN *hrgn*; /* handle of the region */

The **WinQueryUpdateRegion** function obtains the update region of a window. This function, in conjunction with the **WinValidateRegion** function, is useful for implementing an alternative update scheme to the use of the **WinBeginPaint** and **WinEndPaint** functions. You 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.

Parameters *hwnd* Identifies the window whose update region is to be retrieved.
 hrgn Identifies the region that will receive the window's update region.

Return Value The return value is the type of the region identified by the *hrgn* parameter, as defined by the **GpiCombineRegion** function.

See Also **GpiCombineRegion**, **WinBeginPaint**, **WinEndPaint**, **WinValidateRegion**

■ WinQueryVersion

ULONG WinQueryVersion(*hab*)

HAB *hab*; /* handle of the anchor block */

The **WinQueryVersion** function returns the version and revision level of MS OS/2.

Parameters *hab* Identifies an anchor block.

Return Value The return value is the version number, consisting of the major and minor version number and the revision character. The low word contains the minor version in the low byte and the major version in the high byte. The high word contains the revision character in the low byte. You can use the following macros to extract this information:

Macro	Result
LOBYTE (LOUSHORT (<i>return</i>))	Retrieves the major version number.
HIBYTE (LOUSHORT (<i>return</i>))	Retrieves the minor version number.
LOUCHAR (HIUSHORT (<i>return</i>))	Retrieves the revision character.

See Also **DosGetVersion**

■ WinQueryWindow

HWND WinQueryWindow (*hwnd*, *cmd*, *fLock*)

HWND *hwnd*; /* handle of the window */

SHORT *cmd*; /* which window to retrieve */

BOOL *fLock*; /* lock/unlock flag */

The **WinQueryWindow** function retrieves the handle of a window that has a specified relationship to a specified window.

If **WinQueryWindow** is used to enumerate windows of other threads, it is not guaranteed that all the windows are enumerated, because the z ordering of the windows may change during the enumeration. The **WinGetNextWindow** function must be used for this purpose.

Parameters

hwnd Identifies a window. The window handle retrieved is relative to this window, based on the value in the *cmd* parameter.

cmd Specifies which window to retrieve. The following are the possible values:

Value	Meaning
QW_BOTTOM	Bottommost child window.
QW_FRAMEOWNER	Returns the owner of <i>hwnd</i> , normalized so that it shares the same parent as <i>hwnd</i> .
QW_NEXT	Next window in z order (window below).
QW_NEXTTOP	Next main window in the enumeration order defined for the ALT+ESCAPE function of the user interface.
QW_OWNER	Owner of window.
QW_PARENT	Parent of window; HWND_OBJECT if object window.
QW_PREV	Previous window in z order (window above).
QW_PREVTOP	Previous main window, in the enumeration order defined by QW_NEXTTOP.
QW_TOP	Topmost child window.

fLock Specifies whether the retrieved window is to be locked or unlocked. If TRUE, the window is locked. If FALSE, it is not.

Return Value

The return value is the handle of the window related to the window identified by the *hwnd* parameter.

Comments

If the retrieved window is locked by this function, then you must at some point call the **WinLockWindow** function to unlock the window. The reason for locking the window is so that the window cannot be destroyed until you have finished using it.

See Also

WinGetNextWindow, **WinLockWindow**

■ WinQueryWindowDC

HDC WinQueryWindowDC(*hwnd*)

HWND *hwnd*; /* handle of the window */

The WinQueryWindowDC function retrieves the device context created by a call to the WinOpenWindowDC function for the specified window.

Parameters *hwnd* Identifies the window that has the device context.

Return Value The return value is the handle of the device context or NULL if an error occurred.

See Also WinOpenWindowDC

■ WinQueryWindowLockCount

SHORT WinQueryWindowLockCount(*hwnd*)

HWND *hwnd*; /* handle of the window */

The WinQueryWindowLockCount function returns a window's lock count. Since a window may be locked by another thread or process at any time, the value returned by this function may also change at any time.

Parameters *hwnd* Identifies the window whose lock count is being retrieved.

Return Value The return value is the window lock count if the window is locked. Otherwise, it is zero, indicating that the window is not locked or that an error occurred.

See Also WinLockWindow

■ WinQueryWindowPos

BOOL WinQueryWindowPos(*hwnd*, *pswp*)

HWND *hwnd*; /* handle of the window */

PSWP *pswp*; /* address of the structure for window information */

The WinQueryWindowPos function retrieves a window's size and position.

Parameters *hwnd* Identifies the window to get the size and position of.

pswp Points to an SWP structure that receives the window's size and position. The SWP structure has the following form:

```
typedef struct _SWP {
    USHORT fs;
    SHORT cy;
    SHORT cx;
    SHORT y;
    SHORT x;
    HWND hwndInsertBehind;
    HWND hwnd;
} SWP;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinSetWindowPos

■ WinQueryWindowProcess

BOOL WinQueryWindowProcess(*hwnd*, *ppid*, *ptid*)
HWND *hwnd*; /* handle of the window */
PPID *ppid*; /* address of variable for process identifier */
PTID *ptid*; /* address of variable for thread identifier */

The **WinQueryWindowProcess** function obtains the process identifier and thread identifier of the thread that created a window.

Parameters *hwnd* Identifies the window.
ppid Specifies the process identifier of the thread that created the window. It can be NULL if you aren't interested in this value.
ptid Specifies the thread identifier of the thread that created the window. It can be NULL if you aren't interested in this value.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also DosGetPID

■ WinQueryWindowPtr

PVOID WinQueryWindowPtr(*hwnd*, *index*)
HWND *hwnd*; /* handle of the window */
SHORT *index*; /* index to the pointer */

The **WinQueryWindowPtr** function retrieves a pointer value from a specified window's reserved memory.

The window handle that is passed to this function can be the handle of a window with the same or a different message queue from the caller; that is, this function allows the caller to obtain data from windows belonging to other threads.

Parameters *hwnd* Identifies the window that contains the pointer to retrieve.
index Specifies the zero-based index of the pointer to retrieve. Valid values are in the range zero through the number of bytes of window data (for example, a value of 8 would be an index to the third pointer), or QWP_PFNWP to address the index of the window procedure.

Return Value The return value is the specified ULONG value in the window's reserved memory.

Comments The specified index is valid only if all the bytes referenced are within the reserved memory. For example, this function would fail if an index value of zero was specified and only two bytes had been reserved.

See Also WinQueryWindowULong, WinSetWindowPtr

■ WinQueryWindowRect

BOOL WinQueryWindowRect(*hwnd*, *prcl*)

HWND *hwnd*; /* handle of the window */

PRECTL *prcl*; /* address of structure for window coordinates */

The **WinQueryWindowRect** function retrieves the coordinates of a window.

Parameters *hwnd* Identifies the window whose coordinates are retrieved.

prcl Points to a **RECTL** structure that receives the window's coordinates. The **xLeft** and **yBottom** fields will be zero. The **xRight** and **yTop** fields will contain the width and height of the window. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

Example This example calls **WinQueryWindowRect** to get the dimensions of the window, and then calls **WinFillRect** to fill the window.

```
HPS hps;
RECTL rcl;

WinQueryWindowRect(hwnd, &rcl);      /* get window dimensions */
WinFillRect(hps, &rcl, CLR_WHITE);  /* clear entire window */
```

See Also WinQueryWindowPos

■ WinQueryWindowText

SHORT WinQueryWindowText(*hwnd*, *cbBuf*, *pszBuf*)

HWND *hwnd*; /* handle of the window */

SHORT *cbBuf*; /* length of the buffer */

PSZ *pszBuf*; /* address of the buffer */

The **WinQueryWindowText** function copies window text into a buffer. If the window is a frame window, the title-bar window text is copied.

If the window text is longer than (*cbBuf* - 1), only the first (*cbBuf* - 1) characters of window text are copied.

This function sends a **WM_QUERYWINDOWPARAMS** message to the window identified by the *hwnd* parameter.

Parameters *hwnd* Identifies the window containing the text.

cbBuf Specifies the length of the buffer pointed to by the *pszBuf* parameter. If the text is larger than this value, it will be truncated.

pszBuf Points to a buffer that receives the window text.

- Return Value** The return value is the length of the returned text.
- Comments** You can determine the size of the window text ahead of time by calling the `WinQueryWindowTextLength` function.
- See Also** `WinQueryWindowTextLength`

■ WinQueryWindowTextLength

SHORT `WinQueryWindowTextLength(hwnd)`

HWND *hwnd*; /* handle of the window */

The `WinQueryWindowTextLength` function retrieves the length of the window text, not including any null termination character.

This function sends a `WM_QUERYFRAMEINFO` message to the window identified by the *hwnd* parameter.

- Parameters** *hwnd* Identifies the window containing the text.
- Return Value** The return value is the length of the window text.
- See Also** `WinQueryWindowText`

■ WinQueryWindowULong

ULONG `WinQueryWindowULong(hwnd, index)`

HWND *hwnd*; /* handle of the window */

SHORT *index*; /* index of value to retrieve */

The `WinQueryWindowULong` function retrieves an unsigned long integer value at a specified offset into the reserved memory of a given window.

The window handle that is passed to this function can be the handle of a window with the same or a different message queue from the caller; that is, this function allows the caller to obtain data from windows belonging to other threads.

- Parameters** *hwnd* Identifies the window to query.

index Specifies the zero-based index of the **ULONG** value to retrieve. Valid values are in the range zero through the number of bytes of window data (for example, a value of 8 would be an index to the third long integer), or any of the following QWL values:

Value	Meaning
<code>QWL_HHEAP</code>	Handle of the heap.
<code>QWL_HMQ</code>	Handle of the message queue of the window.
<code>QWL_HWNDFOCUSSAVE</code>	Handle of the window that last had the focus.
<code>QWL_STYLE</code>	Window style.

Value	Meaning
QWL_USER	<p>ULONG value present in windows of the following preregistered window classes:</p> <p>WC_DIALOG WC_FRAME WC_LISTBOX WC_BUTTON WC_STATIC WC_ENTRYFIELD WC_SCROLLBAR WC_MENU</p> <p>This value can be used to retrieve application-specific data in controls.</p>

- Return Value** The return value is the specified ULONG value in the window's reserved memory.
- Comments** The specified index is valid only if all the bytes referenced are within the reserved memory. For example, this function would fail if an index value of zero was specified and only two bytes had been reserved.
- See Also** WinQueryWindowUShort, WinRegisterClass, WinSetWindowULong

■ WinQueryWindowUShort

USHORT WinQueryWindowUShort(*hwnd*, *index*)

HWND *hwnd*; /* handle of the window */

SHORT *index*; /* index of value to retrieve */

The WinQueryWindowUShort function retrieves an unsigned short integer value at a specified offset into the reserved memory of a given window.

The window handle that is passed to this function can be the handle of a window with the same or a different message queue from the caller; that is, this function allows the caller to obtain data from windows belonging to other threads.

- Parameters**
- hwnd* Identifies the window to query.
- index* Specifies the zero-based index of the USHORT value to retrieve. Valid values are in the range zero through the number of bytes of window data (for example, a value of 8 would be an index to the fifth integer), or any of the following QWS values:

Value	Meaning
QWS_ID	Window identifier (as passed by the WinCreateWindow function).
QWS_USER	<p>USHORT value present in windows of the following pre-registered window classes:</p> <p>WC_DIALOG WC_FRAME WC_LISTBOX</p>

Value	Meaning
	WC_BUTTON
	WC_STATIC
	WC_ENTRYFIELD
	WC_SCROLLBAR
	WC_MENU
	This value can be used to retrieve application-specific data in controls.

Return Value The return value is the USHORT value in the window's reserved memory.

See Also WinCreateWindow, WinQueryWindowULong, WinRegisterClass, WinSetWindowUShort

■ WinReallocMem

NPBYTE WinReallocMem(*hHeap*, *npMem*, *cbOld*, *cbNew*)

HHEAP *hHeap*; /* handle of the heap */
 NPBYTE *npMem*; /* address of memory block to reallocate */
 USHORT *cbOld*; /* old memory block length */
 USHORT *cbNew*; /* new memory block length */

The **WinReallocMem** function reallocates the size of a memory block on the heap.

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, then this function calls the **WinAllocMem** function to allocate the new, larger object, copies the number of bytes specified by the *cbOld* parameter from the old object to the new, frees the old object, and returns a pointer to the new object. (It never causes an object to grow in place.)

If the passed heap is created with the **HM_MOVEABLE** option, then the value of the *cbOld* parameter is ignored and the value in the size word of the allocated object is used. On completion, the size word contains the value of the *cbNew* parameter. If this function has to move the object in order to satisfy the request, then the handle value word is updated by adding to it the distance of the move, in bytes. The returned address is then the address of the first reserved word.

Parameters

hHeap Identifies the heap. This parameter must have been returned from a previous call to the **WinCreateHeap** function.

npMem Points to the memory block to be reallocated. The low two bits of *npMem* are ignored, although they are preserved in the return value of this function, even if the memory object is moved as a result of growing. Except for the two low bits, the value of the *npMem* parameter must have been returned by either the **WinAllocMem** function or a previous call to **WinReallocMem**.

cbOld Specifies the old size of the memory block, in bytes.

cbNew Specifies the new size of the memory block, in bytes.

Return Value The return value is a pointer to the reallocated memory block if the function was successful. Otherwise, it is `NULL`, indicating that the memory could not be reallocated to the requested size.

The return pointer is a 16-bit offset from the start of the segment containing the heap of the reallocated memory object. The function returns `NULL` when the memory object cannot be reallocated because an invalid heap handle is specified, there is not enough room in the heap to increase the object to the specified size, or the *npMem* parameter points to memory outside the bounds of the passed heap.

See Also WinAllocMem, WinCreateHeap, WinFreeMem

■ WinRegisterClass

BOOL WinRegisterClass(*hab*, *pszClassName*, *pfnWndProc*, *flStyle*, *cbWindowData*)

HAB *hab*; /* handle of the anchor block */
PSZ *pszClassName*; /* points to the class name */
PFNWP *pfnWndProc*; /* address of the window procedure */
ULONG *flStyle*; /* window-style flags */
USHORT *cbWindowData*; /* amount of reserved data */

The `WinRegisterClass` function registers a window class.

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 `WinRegisterClass`.

Private classes are deleted when the process that registers them terminates.

Parameters *hab* Identifies the anchor block.

pszClassName Points to the window classname. It can either be an application-specified name or the name of one of the following preregistered classes:

Class	Description
WC_BUTTON	A button control, including push buttons, radio buttons, check boxes, and user buttons.
WC_ENTRYFIELD	An entry-field control that allows single line-text editing.
WC_FRAME	A standard frame window.
WC_LISTBOX	A list box that displays a scrolling list of items.
WC_MENU	A menu, including the action bar and pull-down menus.
WC_SCROLLBAR	A scroll bar that allows a user to scroll the contents of a window.
WC_STATIC	A static control that displays text, icon, or bitmap data.
WC_TITLEBAR	A title-bar control that displays the title of a window across the top of the frame and also allows the user to drag the frame window to a new location.

pfWndProc Points to the window procedure, which can be NULL if the application does not provide its own window procedure.

flStyle Specifies the default window style, which can be any of the standard CS class styles, in addition to any class-specific styles that may be 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. Public classes are available for window creation from any process. Private classes are only available to the registering process.

The following list describes the standard classes:

Style	Meaning
CS_CLIPCHILDREN	Sets the WS_CLIPCHILDREN style for windows created using this class.
CS_CLIPSIBLINGS	Sets the WS_CLIPSIBLINGS style for windows created using this class.
CS_FRAME	Identifies windows created using this class as frame windows. Frame windows receive the special WM_FRAMEDESTROY message when they are being destroyed.
CS_HITTEST	Directs the system to send WM_HITTEST messages to windows of this class whenever the mouse moves in the window.
CS_MOVENOTIFY	Directs the system to send WM_MOVE messages to the window whenever the window moves.
CS_PARENTCLIP	Sets the WS_PARENTCLIP style for windows created using this class.
CS_PUBLIC	Creates a public window class.
CS_SAVEBITS	Sets the WS_SAVEBITS style for windows created using this class.
CS_SIZEREDRAW	Directs the system to invalidate the entire window whenever the size of the window changes.
CS_SYNCPAINT	Sets the WS_SYNCPAINT style for windows created using this class.

cbWindowData Specifies the number of bytes of storage reserved per window created of this class for application use.

Return Value

The return value is TRUE if the function is successful or FALSE if an error occurs.

Example

This example calls WinRegisterClass to register a class, return FALSE if an error occurs.

```

HAB hab;
CHAR szClassName[] = "Generic"; /* window class name */

if (!WinRegisterClass(hab, /* anchor-block handle */
                    szClassName, /* class name */
                    GenericWndProc, /* window procedure */
                    OL, /* window style */
                    0)) /* amount of reserved memory */
    return (FALSE);

```

See Also WinQueryClassInfo, WinQueryClassName, WinQueryWindowPtr, WinQueryWindowULong, WinQueryWindowUShort

■ WinRegisterWindowDestroy

BOOL WinRegisterWindowDestroy(*hwnd*, *fRegister*)

HWND *hwnd*; /* handle of the window */

BOOL *fRegister*; /* register flag */

The **WinRegisterWindowDestroy** function notifies other applications when the specified window is destroyed.

Parameters *hwnd* Identifies the window being destroyed.

fRegister Specifies whether the window is to be registered. If *fRegister* is TRUE, this function 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. If *fRegister* is FALSE, this routine unregisters the window by decreasing the register count by one, although the window is not fully unregistered until the count reaches zero.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinDestroyWindow

■ WinReleaseHook

BOOL WinReleaseHook(*hab*, *hmq*, *iHook*, *pfnHook*, *hmod*)

HAB *hab*; /* handle of the anchor block */

HMQ *hmq*; /* handle of the message queue */

SHORT *iHook*; /* hook identifier */

PFN *pfnHook*; /* address of the hook procedure */

HMODULE *hmod*; /* handle of the module with hook procedure */

The **WinReleaseHook** function releases an application hook from a hook chain.

Parameters *hab* Identifies the anchor block.

hmq Specifies the message queue from which the hook is to be released. If *hmq* is NULL, the hook is released from the system hook chain. If *hmq* is HMQ_CURRENT, the hook is released from the message queue associated with the current thread (calling thread).

iHook Specifies the type of hook chain. This parameter can be one of the following values:

Hook type	Description
HK_HELP	Monitors the WM_HELP message.
HK_INPUT	Monitors messages in a message queue.
HK_JOURNALPLAYBACK	Allows applications to insert events into the system input queue.
HK_JOURNALRECORD	Allows applications to record system input queue events.
HK_MSGFILTER	Monitors input events during system modal loops.
HK_SENDMSG	Monitors messages sent with WinSendMsg.

pfnHook Points to the hook routine.

hmod Identifies the module that contains the hook procedure. This parameter can be either the module handle returned by the DosLoadModule function or NULL for the application's module.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments If a system hook is called by a process other than the process that set the hook, the DLL containing the hook will not be unloaded until every process is exited that referenced it. For most system hooks, this applies until the machine is rebooted. For the most part, this is not a problem; as with swapping, the unused DLL will end up somewhere in your swap space, never to be used again. The only complication is that when you are developing the hook the DLL containing the hook is still in use, and you cannot copy over it or link into it.

See Also DosLoadModule, WinSendMsg, WinSetHook

■ WinReleasePS

BOOL WinReleasePS(*hps*)

HPS *hps*; /* handle of the presentation space */

The WinReleasePS function releases a cache presentation space obtained using the WinGetPS function.

Only a cache presentation space should be released using this function. The presentation space is returned to the cache for reuse. The presentation space handle should not be used following this function.

Parameters *hps* Identifies the cache presentation space to release.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments Before an application terminates, it must call WinReleasePS to release any cache presentation spaces obtained by using the WinGetPS function.

Example

This example processes an application-defined message (IDM_FILL). It calls `WinGetPS` to get a presentation space to the entire window. It gets the dimensions of the current window, fills the window, and calls `WinReleasePS` to release the presentation space.

```
case IDM_FILL:
    hps = WinGetPS(hwnd);          /* get ps for the entire window */
    WinQueryWindowRect(hwnd, &rcl); /* get window dimensions      */
    WinFillRect(hps, &rcl, CLR_WHITE); /* clear entire window        */
    WinReleasePS(hps);           /* release the ps              */
```

See Also

`WinGetPS`

■ WinRemoveSwitchEntry

USHORT WinRemoveSwitchEntry (*hSwitch*)

HSWITCH *hSwitch*; /* handle of the switch list */

The `WinRemoveSwitchEntry` function removes a specified entry from the switch list.

Switch-list entries for full-screen applications cannot be removed using this function. These entries are removed automatically by the system when the application terminates.

Parameters

hSwitch Identifies the switch-list entry for the application to remove.

Return Value

The return value is zero if the function is successful. Otherwise, it is nonzero, indicating that an error occurred.

See Also

`WinAddSwitchEntry`, `WinChangeSwitchEntry`

■ WinScrollWindow

SHORT WinScrollWindow (*hwnd, dx, dy, prclScroll, prclClip, hrgnUpdate, prclUpdate, fs*)

HWND *hwnd*; /* handle of the window to scroll */
SHORT *dx*; /* amount of horizontal scrolling */
SHORT *dy*; /* amount of vertical scrolling */
PRECTL *prclScroll*; /* address of structure with scroll rectangle */
PRECTL *prclClip*; /* address of structure with clip rectangle */
HRGN *hrgnUpdate*; /* handle of the update region */
PRECTL *prclUpdate*; /* address of the structure for the update rectangle */
USHORT *fs*; /* scrolling flags */

The `WinScrollWindow` function scrolls the contents of a window rectangle.

No application should move bits in its own window by any other method than by using `WinScrollWindow`.

The cursor and the track rectangle are also scrolled when they intersect with the scrolled region.

Parameters

hwnd Identifies the window to scroll.

dx Specifies the amount of horizontal scrolling (in device units).

dy Specifies the amount of vertical scrolling (in device units).

prclScroll Points to a **RECTL** structure that specifies the scroll rectangle. If *prclScroll* is **NULL**, the entire window will be scrolled. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

prclClip Points to a **RECTL** structure that specifies the clip rectangle. This structure takes precedence over the *prclScroll* rectangle. Bits outside of the *prclClip* rectangle are not affected even if they are in *prclScroll*.

hrgnUpdate Identifies the region that is modified to hold the region invalidated by scrolling. This parameter may be **NULL**.

prclUpdate Points to a **RECTL** structure that receives the boundaries of the rectangle invalidated by scrolling. This parameter may be **NULL**.

fs Specifies flags controlling the scrolling. It can be a combination of the following values:

Value	Meaning
SW_SCROLLCHILDREN	All child windows are scrolled.
SW_INVALIDATERGN	The invalid region created as a result of scrolling will be added to update regions of those windows affected. This may result in the sending of WM_PAINT messages to WS_SYNCPAINT windows before the WinScrollWindow function returns.

Return Value

The return value is a code indicating the type of invalid region created by scrolling:

Value	Meaning
ERROR	Error in the passed region.
NULLREGION	Scrolling produced no invalidation.
SIMPLEREGION	Scrolling produced rectangular invalidation.
COMPLEXREGION	Scrolling produced a nonrectangular invalidation.

Comments

To quickly repeat scrolling, omit the **SW_INVALIDATERGN** flag from the *fs* parameter and accumulate the update area by specifying a region for the *hrgnUpdate* parameter or a rectangle for the *prclUpdate* parameter. When scrolling is completed, you can repaint the window by calling **WinInvalidateRegion** or **WinInvalidateRect**, depending on whether you specified the *hrgnUpdate* or *prclUpdate* parameter.

Typically, an application will set the `SW_INVALIDATERGN` flag in *fs* and let the system handle the update regions of the affected windows. In this case, the *hrgnUpdate* and *prclUpdate* parameters can both be set to `NULL`.

If the *hwnd* parameter does not have the `WS_CLIPCHILDREN` style, the bits of any child window falling inside the scrolled area will also be scrolled. If this is the case, `WinScrollWindow` should be called with `SW_SCROLLCHILDREN`.

The cursor and tracking rectangle are always hidden if they are in the window being scrolled or a child of that window, and the cursor is always offset by the distance scrolled. The tracking rectangle's position is left alone. Both are then shown once scrolling is done.

See Also `GpiCombineRegion`

■ WinSendDlgItemMsg

```

MRESULT WinSendDlgItemMsg(hwndDlg, idItem, msg, mp1, mp2)
HWND hwndDlg; /* handle of the dialog box */
USHORT idItem; /* dialog-item identifier */
USHORT msg; /* message */
MPARAM mp1; /* first message parameter */
MPARAM mp2; /* second message parameter */

```

The `WinSendDlgItemMsg` function sends a message to the specified dialog item in the dialog window. The function does not return until the message has been processed by the dialog item.

This function is equivalent to the following:

```
WinSendMessage(hwndDlg, idItem, msg, mp1, mp2);
```

Parameters

- hwndDlg* Identifies the dialog window.
- idItem* Identifies the dialog item that receives the message.
- msg* Specifies the message.
- mp1* Specifies message parameter 1.
- mp2* Specifies message parameter 2.

Return Value The return value is the result returned by the dialog item to which the message was sent.

See Also `WinSendMessage`, `WinWindowFromID`

■ WinSendMessage

```
MRESULT WinSendMessage(hwnd, msg, mp1, mp2)
HWND hwnd;      /* handle of the receiving window */
USHORT msg;     /* message */
MPARAM mp1;    /* first message parameter */
MPARAM mp2;    /* second message parameter */
```

The `WinSendMessage` function sends a message to the specified window.

This function does not return until the message has been processed by the window procedure. If the window receiving the message belongs to the same thread, the window function is called immediately as a subroutine. If the window is of another thread or process, Presentation Manager switches to the appropriate thread and calls the appropriate window function, passing the message to the window function. The message is not placed in the destination thread's queue.

Parameters

hwnd Identifies the window to send the message to.

msg Specifies the message.

mp1 Specifies message parameter 1.

mp2 Specifies message parameter 2.

Return Value The return value is the result returned by the invoked window procedure.

Comments The `WM_USER` constant marks the beginning of values you can use for your own messages. For example, you might have a section of a header file that looks like this:

```
#define WM_USERMSG00 (WM_USER + 0)
#define WM_USERMSG01 (WM_USER + 1)
#define WM_USERMSG02 (WM_USER + 2)
#define WM_USERMSG03 (WM_USER + 3)
```

The following lists some of the differences between `WinPostMsg` and `WinSendMessage`:

- `WinPostMsg` returns immediately. `WinSendMessage` waits for the receiver to return.
- A thread that does not have a message queue can still call `WinPostMsg`. It cannot call `WinSendMessage`.
- Calling `WinSendMessage` to send a message to another thread is costly in terms of CPU time. This is not true of `WinPostMsg`.

Example This example gets the window handle of the system menu and calls `WinSendMessage` to send a message to disable the Close menu item.

```
HWND hwndSysMenu;

hwndSysMenu = WinWindowFromID(hwndDlg, FID_SYSMENU);
WinSendMessage(hwndSysMenu, MM_SETITEMATTR,
    MPFROM2SHORT(SC_CLOSE, TRUE),
    MPFROM2SHORT(MIA_DISABLED, MIA_DISABLED));
```

See Also `WinBroadcastMsg`, `WinPostMsg`

■ WinSetAccelTable

BOOL WinSetAccelTable(*hab*, *haccel*, *hwndFrame*)

HAB *hab*; /* handle of the anchor block */

HACCEL *haccel*; /* handle of the accelerator table */

HWND *hwndFrame*; /* handle of the frame window */

The **WinSetAccelTable** function sets the window or queue accelerator table.

Parameters

hab Identifies the anchor block.

haccel Identifies the accelerator table. If *haccel* is NULL, the effect of this function is to remove any accelerator table in effect for the window or queue.

hwndFrame Identifies the frame window. If *hwndFrame* is NULL, the queue accelerator table is set. Otherwise, the window accelerator table is set, by sending the WM_SETACCELTABLE message to *hwndFrame*.

Return Value

The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also

WinCreateAccelTable, WinLoadAccelTable

■ WinSetActiveWindow

BOOL WinSetActiveWindow(*hwndDesktop*, *hwnd*)

HWND *hwndDesktop*; /* handle of the desktop */

HWND *hwnd*; /* handle of the window to make active */

The **WinSetActiveWindow** function makes the frame window of *hwnd* the active window. It does this by finding the first frame window parent of *hwnd*, using *hwnd* if it is a frame window. It then finds the last window associated with this frame window that had the focus. Then the focus is set to this window that previously had the focus, using the function **WinSetFocus**. This sets the focus to this window and activates all frame windows that are parents of this window.

The window handle of the window that receives the focus is stored in the frame window's reserved memory. This memory can be queried by using a QWL_HWNDFOCUSSAVE index with the **WinQueryWindowULong** function.

If the active window is changing, the following events occur:

- If the action of setting the active window results in a different window receiving the focus, the window that currently has the focus will receive a WM_SETFOCUS message indicating the loss of focus.
- If the action of setting the active window results in a different window becoming active, a WM_ACTIVATE message is sent to the current active window, indicating a loss of the active status.
- The new active window is established.
- A WM_ACTIVATE message indicating the acquisition of active status is sent to the new window.

Parameters	<i>hwndDesktop</i> Identifies the desktop window. This parameter can be <code>HWND_DESKTOP</code> or the desktop window handle. <i>hwnd</i> Identifies either a frame window or the child of a frame window. If it is a child, the parent frame window will become the active window.
Return Value	The return value is <code>TRUE</code> if the function is successful. Otherwise, it is <code>FALSE</code> , indicating that an error occurred.
See Also	<code>WinQueryActiveWindow</code> , <code>WinQueryFocus</code> , <code>WinQueryWindowULong</code> , <code>WinSetFocus</code>

■ WinSetCapture

BOOL WinSetCapture(*hwndDesktop*, *hwnd*)

HWND *hwndDesktop*; /* handle of the desktop */

HWND *hwnd*; /* handle of the window to receive all mouse messages */

The `WinSetCapture` function sets the mouse capture to the specified window. With the mouse capture set to a window, all mouse input is directed to that window, regardless of whether the mouse is over that window. Only one window can have the mouse captured at a time.

When the `WinSetCapture` function is called to release the mouse capture, a `WM_MOUSEMOVE` message is posted, regardless of whether the mouse pointer has actually moved.

Parameters	<i>hwndDesktop</i> Identifies the desktop window. This parameter can be <code>HWND_DESKTOP</code> or the desktop window handle. <i>hwnd</i> Identifies the window that is to receive all mouse messages. This parameter can take the special value <code>HWND_THREADCAPTURE</code> to capture the mouse to the current thread rather than to a particular window. If <i>hwnd</i> is <code>NULL</code> , mouse capture is released.
Return Value	The return value is <code>TRUE</code> if the function is successful or <code>FALSE</code> if an error occurs. If an application calls this function while the mouse is currently captured by a different window, the function will fail and return <code>FALSE</code> .
See Also	<code>WinQueryCapture</code>

■ WinSetClipbrdData

BOOL WinSetClipbrdData(*hab*, *ulData*, *fmt*, *fsFmtInfo*)

HAB *hab*; /* handle of the anchor block */

ULONG *ulData*; /* data object */

USHORT *fmt*; /* specifies the format */

USHORT *fsFmtInfo*; /* specifies the data type */

The `WinSetClipbrdData` function puts data into the clipboard. Data of the specified format already in the clipboard is freed by this function.

Parameters

hab Identifies an anchor block.

ulData Specifies the data object being put into the clipboard. If this parameter is NULL, a WM_RENDERFMT message is sent to the clipboard owner window, to render the format when the WinQueryClipbrdData function is called with the specified format. Once the data object has been put into the clipboard, the object it refers to (if given by a reference such as a selector or handle) is no longer accessible by the application. To access the data after it has been placed into the clipboard, use the WinQueryClipbrdData function.

fmt Specifies the format of the data object specified by the *ulData* parameter. The following list describes the standard clipboard formats:

Value	Meaning
CF_BITMAP	The data is a bitmap. The CFI_HANDLE memory-model flag must be set in the <i>fsFmtInfo</i> parameter.
CF_DSPBITMAP	The data is a bitmap representation of a private data format. The clipboard viewer can use this format to display a private format. The memory-model flag CFI_HANDLE must be set in the <i>fsFmtInfo</i> parameter.
CF_METAFILE	The data is a metafile. The CFI_HANDLE memory-model flag must be set in the <i>fsFmtInfo</i> parameter.
CF_DSPMETAFILE	The data is a metafile representation of a private data format. The clipboard viewer can use this format to display a private format. The memory-model flag CFI_HANDLE must be set in the <i>fsFmtInfo</i> parameter.
CF_TEXT	The data is an array of text characters, which may include newline characters to mark line breaks. The null character indicates the end of the text data. The CFI_SELECTOR memory-model flag must be set in the <i>fsFmtInfo</i> parameter.
CF_DSPTEXT	The data is a textual representation of a private data format. The clipboard viewer can use this format to display a private format. The memory-model flag CFI_SELECTOR must be set in the <i>fsFmtInfo</i> parameter.

fsFmtInfo Specifies the type of data specified by the *ulData* parameter. This consists of memory-model and usage flags, as follows:

Memory-model flag	Meaning
CFI_HANDLE	Handle is the handle to a metafile or bitmap.
CFI_SELECTOR	Handle is a selector plus zero offset to a segment in storage.

Usage flag	Meaning
CFL_OWNERDISPLAY	Format will be drawn by the clipboard owner in the clipboard-viewer window by means of the WM_PAINTCLIPBOARD message. The <i>ulData</i> parameter should be NULL.
CFL_OWNERFREE	Handle is not freed by the WinEmptyClipboard function. The application must free the data, if necessary.

Any number of the usage flags may be specified, but only one of the memory models may be specified. When using WinSetClipbrdData for user-defined formats, an application puts a user-defined format into the clipboard. It may then specify the CFL_SELECTOR memory model. The system then saves the selector so that if the calling application terminates, normally or abnormally, the data is still available. The system frees the selector from the calling process; therefore, the calling process may no longer use the selector.

Return Value The return value is TRUE if data is placed in the clipboard, or FALSE if an error occurred.

See Also WinEmptyClipboard, WinQueryClipboardData

■ WinSetClipbrdOwner

```

BOOL WinSetClipbrdOwner(hab, hwnd)
HAB hab;          /* handle of the anchor block */
HWND hwnd;       /* handle of the clipboard owner */

```

The WinSetClipbrdOwner function sets the current clipboard owner window. The clipboard owner window receives the following clipboard-related messages at appropriate times:

```

WM_RENDERFMT
WM_DESTROYCLIPBOARD
WM_SIZECLIPBOARD
WM_VSCROLLCLIPBOARD
WM_HSCROLLCLIPBOARD
WM_PAINTCLIPBOARD

```

Parameters *hab* Identifies an anchor block.

hwnd Identifies a new clipboard owner window. If this parameter is NULL, the clipboard owner is released and no new owner is established.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinQueryClipboardOwner, WinSetClipboardViewer

■ WinSetClipbrdViewer

BOOL WinSetClipbrdViewer(*hab*, *hwnd*)

HAB *hab*; /* handle of the anchor block */

HWND *hwnd*; /* handle of the clipboard viewer */

The **WinSetClipbrdViewer** function sets the current clipboard-viewer window to a specified window.

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.

Parameters

hab Identifies an anchor block.

hwnd Identifies a new clipboard viewer window. If this parameter is **NULL**, the clipboard viewer is released and no new viewer is established.

Return Value

The return value is **TRUE** if there is a clipboard-viewer window on completion of the function. Otherwise, it is **FALSE**, indicating that there is no clipboard-viewer window.

See Also

WinQueryClipbrdViewer, **WinSetClipbrdOwner**

■ WinSetCp

BOOL WinSetCp(*hmq*, *idcp*)

HMQ *hmq*; /* handle of the message queue */

USHORT *idcp*; /* code page */

The **WinSetCp** function sets the queue code page for the message queue.

Parameters

hmq Identifies a message queue.

idcp Specifies a code page. It must be one of the ASCII code pages defined in the *config.sys* file.

Return Value

The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also

DosSetCp, **GpiSetCp**, **VioSetCp**, **WinQueryCp**, **WinQueryCpList**

■ WinSetDlgItemShort

```

BOOL WinSetDlgItemShort(hwndDlg, idItem, usValue, fSigned)
HWND hwndDlg; /* handle of the dialog box */
USHORT idItem; /* dialog-item identifier */
USHORT usValue; /* value to set */
BOOL fSigned; /* signed/unsigned flag */

```

The **WinSetDlgItemShort** function sets the text of a dialog-box item to the string representation of a specified integer value. The item is identified by its identifier. The text produced is always an ASCII string.

Parameters

hwndDlg Identifies the dialog-box window.

idItem Identifies the dialog item that is changed.

usValue Specifies the integer value used to generate the item text.

fSigned Specifies whether the *usValue* parameter is signed or unsigned. If this parameter is **TRUE**, *usValue* is signed. If it is **FALSE**, *usValue* is unsigned.

Return Value

The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also

WinQueryDlgItemShort

■ WinSetDlgItemText

```

BOOL WinSetDlgItemText(hwndDlg, idItem, pszText)
HWND hwndDlg; /* handle of the dialog box */
USHORT idItem; /* dialog-item identifier */
PSZ pszText; /* text to set */

```

The **WinSetDlgItemText** function sets the text in a dialog item. It is equivalent to the following:

```
WinSetWindowText(WinWindowFromID(hwndDlg, idItem), pszText);
```

Parameters

hwndDlg Identifies a dialog window.

idItem Identifies the identifier of the dialog item.

pszText Points to a null-terminated string that contains the text to be set for the dialog item.

Return Value

The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also

WinSetWindowText, **WinWindowFromID**

■ WinSetFocus

BOOL WinSetFocus(*hwndDesktop*, *hwndSetFocus*)

HWND *hwndDesktop*; /* handle of the desktop */

HWND *hwndSetFocus*; /* handle of the window receiving the focus */

The **WinSetFocus** function sets the focus window.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be **HWND_DESKTOP** or the desktop window handle.

hwndSetFocus Identifies the window that receives the focus. This parameter must be equal to or be a descendant of the window identified by the *hwndDesktop* parameter. If *hwndSetFocus* identifies a desktop window or is **NULL**, no window on the device associated with *hwndDesktop* receives the focus.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

Comments If *hwndSetFocus* does not have the focus when this function is called, the following events occur:

- If a window currently has the focus, it receives a **WM_SETFOCUS** message, indicating the loss of the focus.
- If a window currently is selected, it receives a **WM_SETSELECTION** message, indicating the window is deselected.
- If changing the focus causes a change in the active window, a **WM_ACTIVATE** message is sent to the active window, indicating the loss of active status.
- If a new application is being made the active application, a **WM_ACTIVATE** message is sent to the active application, indicating the loss of active status.
- The new active and focus windows and the active application are established.
- If a new application is being made the active application, a **WM_ACTIVATE** message is sent to the new application, indicating the acquisition of active status.
- If the active window is changing, a **WM_ACTIVATE** message is sent to the new main window, indicating the acquisition of active status.
- If a new window is selected, it receives a **WM_SETSELECTION** message, indicating the window has been selected.
- The new focus window is sent a **WM_SETFOCUS** message, indicating the acquisition of focus.

Using the **WinQueryActiveWindow** or the **WinQueryFocus** function during processing of the **WinSetFocus** function results in the previous active and focus windows being returned until the new active and focus windows are established. In other words, even though a **WM_SETFOCUS** message with the *fFocus* parameter set to **FALSE** or a **WM_ACTIVATE** message with the *fActive* parameter set to **FALSE** may have been sent to the previous windows, those windows are considered to be active and have the focus until the system establishes the new active and focus windows.

If **WinSetFocus** is called during **WM_ACTIVATE** message processing, a **WM_SETFOCUS** message with the *fFocus* parameter set to **FALSE** is not sent, since no window has the focus.

If no window has the input focus, then WM_CHAR messages are posted to the active window's queue.

Example

This example retrieves an integer from a dialog entry field. It then checks for a valid number. If not found, it displays a message box indicating that an error occurred, and then calls `WinSetFocus` to set the focus back to the entry field that caused the error.

```
fError = WinQueryDlgItemShort(hwndDlg, idEntryField, &iValue, TRUE);
if (fError || (iValue < iLoRange) || (iValue > iHiRange)) {
    WinMessageBox(HWND_DESKTOP, hwndFrame, (PSZ) szErrMsg,
        NULL, idMessageBox, MB_OK );
    WinSetFocus(HWND_DESKTOP, WinWindowFromID(hwndDlg, idEntryField));
}
else {
```

See Also

`WinFocusChange`, `WinQueryActiveWindow`, `WinQueryFocus`, `WinSetActiveWindow`

■ WinSetHook

```
BOOL WinSetHook(hab, hmq, iHook, pfnHook, hmod)
HAB hab; /* handle of the anchor block */
HMQ hmq; /* handle of the message queue */
SHORT iHook; /* type of hook chain */
PFN pfnHook; /* address of the hook procedure */
HMODULE hmod; /* handle of the module with the hook procedure */
```

The `WinSetHook` function installs an application procedure into a specified hook chain. In this function, queue hooks are called before system hooks.

A call to `WinSetHook` installs the hook at the head of either the system or queue chain. The most recently installed hook is called first.

Parameters

hab Identifies an anchor block.

hmq Identifies the queue to which the hook chain belongs. If this parameter is NULL, the hook is installed in the system hook chain. If it is HMQ_CURRENT, the hook is installed in the message queue associated with the current thread (calling thread).

iHook Specifies the type of hook chain. This parameter can be one of the following values:

Hook type	Description
HK_HELP	Monitors the WM_HELP message. Returns BOOL. If FALSE, next hook in chain is called. If TRUE, the next hook in the chain is not called.
HK_INPUT	Monitors messages in specified message queue. Returns BOOL. If FALSE, next hook in the chain is called. If TRUE, the message is not passed on to the next hook in the chain.

Hook type	Description
HK_JOURNALPLAYBACK	Allows applications to insert events into the system input queue. Returns LONG time-out value. This value is the time to wait (in milliseconds) before processing the current message. Never calls the next hook in the chain.
HK_JOURNALRECORD	Allows applications to record system input queue events. Returns VOID. Next hook in chain is always called.
HK_MSGFILTER	Monitors input events during system modal loops. Returns BOOL. If FALSE, next hook in the chain is called. If TRUE, the message is not passed on to the next hook in the chain.
HK_SENDMSG	Monitors messages sent with WinSendMsg. Returns VOID. Next hook in chain is always called.

pfnHook Points to an application hook procedure.

hmod Identifies the module that contains the hook procedure. This parameter can be either the module handle returned by the `DosLoadModule` function or NULL for the application's module.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments You should use the handle returned from `WinCreateMsgQueue` for the *hmq* parameter. If this is not available, you can use the `WinQueryWindowULong` function with the index `QWL_HMQ` to obtain the queue handle associated with a window handle.

Note: If a system hook is installed, the procedure must be contained in a DLL; the procedure may be called from different applications, which do not have access to code segments that are contained in a `.exe` file.

See Also `DosLoadModule`, `WinCreateMsgQueue`, `WinQueryWindowULong`, `WinReleaseHook`, `WinSendMsg`

■ WinSetKeyboardStateTable

BOOL WinSetKeyboardStateTable (*hwndDesktop*, *pKeyStateTable*, *fSet*)

HWND *hwndDesktop*; /* handle of the desktop */

PBYTE *pKeyStateTable*; /* address of the key table */

BOOL *fSet*; /* set/copy flag */

The `WinSetKeyboardStateTable` function gets or sets the keyboard state. This function does not change the physical state of the keyboard; it changes the value returned by the `WinGetKeyState` function, not the value returned by the `WinGetPhysKeyState` function.

To set the state of a single key you must get the entire table, modify the individual key, and then set the table from the modified value.

- Parameters**
- hwndDesktop* Identifies the desktop window. This parameter can be `HWND_DESKTOP` or the desktop window handle.
- pKeyStateTable* Points to a 256-byte table indexed by virtual-key value. For any virtual key, the 0x80 bit is set if the key is down and cleared if it is up. The 0x01 bit is set if the key is toggled (pressed an odd number of times) and cleared otherwise.
- fSet* Specifies whether the keyboard state is set or copied. If this parameter is `TRUE`, the keyboard state is set from *pKeyStateTable*. If this parameter is `FALSE`, the keyboard state is copied to *pKeyStateTable*.
- Return Value** The return value is `TRUE` if the function is successful or `FALSE` if an error occurs.
- See Also** `WinGetKeyState`, `WinGetPhysKeyState`

■ WinSetMultWindowPos

BOOL WinSetMultWindowPos(*hab*, *pswp*, *cswp*)

HAB *hab*; /* handle of the anchor block */
PSWP *pswp*; /* address of array of SWP structures */
USHORT *cswp*; /* number of SWP structures */

The `WinSetMultWindowPos` function performs the `WinSetWindowPos` function for specified windows using the *pswp* parameter, an array of structures whose elements correspond to the input parameters of `WinSetWindowPos`. All windows being positioned must have the same parent window.

It is more efficient to use this function than to issue multiple `WinSetWindowPos` calls, as it causes less screen updating.

- Parameters**
- hab* Identifies an anchor block.
- pswp* Points to an array of `SWP` data structures whose elements correspond to the input parameters of `WinSetWindowPos`. The `SWP` structure has the following form:

```
typedef struct _SWP {
    USHORT fs;
    SHORT cy;
    SHORT cx;
    SHORT y;
    SHORT x;
    HWND hwndInsertBehind;
    HWND hwnd;
} SWP;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

cswp Specifies the number of SWP structures.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments This function sends the following messages. If you process these messages, you must be careful to not cause an infinite loop by calling the WinSetWindowPos or WinSetMultWindowPos functions.

```
WM_ACTIVATE
WM_ADJUSTWINDOWPOS
WM_CALCVALIDRECTS
WM_MOVE
WM_SHOW
WM_SIZE
```

See Also WinSetWindowPos

■ WinSetOwner

BOOL WinSetOwner(*hwnd*, *hwndNewOwner*)

HWND *hwnd*; /* handle of the window whose owner is changed */

HWND *hwndNewOwner*; /* handle of the new owner window */

The WinSetOwner function changes the owner of a specified window. The owner window and the owned window must have been created by the same thread.

The WinQueryWindow function can be used to obtain the handle of the owner window.

Parameters *hwnd* Identifies the window whose owner is changed.

hwndNewOwner Identifies the new owner window. If this parameter is NULL, the window’s owner is set to NULL.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinQueryWindow, WinSetParent

■ WinSetParent

BOOL WinSetParent (*hwnd*, *hwndNewParent*, *fRedraw*)

HWND *hwnd*; /* handle of the window whose parent is changed */

HWND *hwndNewParent*; /* handle of the new parent window */

BOOL *fRedraw*; /* redraw flag */

The **WinSetParent** function sets the parent window for the window identified by the *hwnd* parameter to the window identified by the *hwndNewParent* parameter.

Parameters *hwnd* Identifies the window whose parent will be changed.

hwndNewParent Identifies the new parent window. If this parameter is a desktop-window handle or **HWND_DESKTOP**, the *hwnd* window becomes a main window. The new parent window cannot be a descendant of the *hwnd* window. If *hwndNewParent* is not equal to **HWND_OBJECT**, the windows identified by the *hwndNewParent* and *hwnd* parameters must both be descendants of the same desktop window.

fRedraw Specifies a redraw indicator. If this parameter is **TRUE**, any necessary redrawing of both the old parent and the new parent windows is performed (if the *hwnd* window is visible). If this parameter is **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.

Return Value The return value is **TRUE** if the parent window is successfully changed. Otherwise, it is **FALSE**, indicating that the parent window was not successfully changed.

See Also **WinIsChild**, **WinSetOwner**

■ WinSetPointer

BOOL WinSetPointer (*hwndDesktop*, *hptrNew*)

HWND *hwndDesktop*; /* handle of the desktop */

HPOINTER *hptrNew*; /* handle of the pointer */

The **WinSetPointer** function changes the mouse pointer.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be **HWND_DESKTOP** or the desktop window handle.

hptrNew Identifies the new pointer. If the *hptrNew* parameter is **NULL**, the pointer is removed from the screen.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

Comments If you process the **WM_MOUSEMOVE** message, and you don't call the **WinDefWindowProc** function as part of that processing, then you should call this function to set the mouse pointer. This function is quite fast if the mouse pointer is not changed.

The following functions can be used to obtain a handle to a pointer that can be used as the mouse pointer:

Function	Description
WinCreatePointer	Creates a pointer from a bitmap.
WinQueryPointer	Retrieves a handle for the desktop pointer.
WinLoadPointer	Loads a pointer from a resource file or dynamic-link module.
WinQuerySysPointer	Retrieves a handle for one of the system pointers.

Example

This example calls **WinLoadPointer** to load an application-defined pointer. When processing the **WM_MOUSEMOVE** message, the loaded pointer is displayed by calling **WinSetPointer**.

```
case WM_CREATE:
    hptrCrossHair = WinLoadPointer(HWND_DESKTOP,
        NULL, /* load from .exe file */
        IDP_CROSSHAIR); /* identifies the pointer */
case WM_MOUSEMOVE:
    WinSetPointer(HWND_DESKTOP, hptrCrossHair);
```

See Also

WinCreatePointer, **WinDefWindowProc**, **WinLoadPointer**, **WinQueryPointer**, **WinQuerySysPointer**

■ WinSetPointerPos

```
BOOL WinSetPointerPos(hwndDesktop, x, y)
HWND hwndDesktop; /* handle of the desktop */
SHORT x; /* horizontal position */
SHORT y; /* vertical position */
```

The **WinSetPointerPos** function sets the mouse pointer position.

Parameters

hwndDesktop Identifies the desktop window. This parameter can be **HWND_DESKTOP** or the desktop window handle.

x Specifies the *x* position of the pointer (in screen coordinates).

y Specifies the *y* position of the pointer (in screen coordinates).

Return Value

The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also

WinQueryPointerPos

■ WinSetRect

```

BOOL WinSetRect(hab, prcl, xLeft, yBottom, xRight, yTop)
HAB hab;           /* handle of the anchor block */
PRECTL prcl;      /* address of structure with rectangle to set */
SHORT xLeft;      /* left side */
SHORT yBottom;    /* bottom side */
SHORT xRight;     /* right side */
SHORT yTop;       /* top side */

```

The **WinSetRect** function sets rectangle coordinates. This function is equivalent to assigning the left, top, right, and bottom arguments to the appropriate fields of the **RECTL** structure. The coordinates of the rectangle are sign-extended before being placed into the **RECTL** structure.

Parameters *hab* Identifies an anchor block.

prcl Points to a **RECTL** structure that contains the rectangle to be set. The **RECTL** structure has the following form:

```

typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

xLeft Specifies the left edge of the rectangle.

yBottom Specifies the bottom edge of the rectangle.

xRight Specifies the right edge of the rectangle.

yTop Specifies the top edge of the rectangle.

Return Value The return value is always **TRUE**.

See Also **WinSetRectEmpty**

■ WinSetRectEmpty

```

BOOL WinSetRectEmpty(hab, prcl)
HAB hab;           /* handle of the anchor block */
PRECTL prcl;      /* address of structure with rectangle to set to empty */

```

The **WinSetRectEmpty** function sets a rectangle to empty. This function is equivalent to **WinSetRect**(*hab*, *prcl*, 0, 0, 0, 0).

Parameters *hab* Identifies an anchor block.

prcl Points to a **RECTL** structure that contains the rectangle to be set to empty. The **RECTL** structure has the following form:

```

typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is always TRUE.

See Also WinCopyRect, WinSetRect

■ WinSetSysColors

BOOL WinSetSysColors(*hwndDesktop, flOptions, flFormat, clrFirst, cclr, pclr*)

HWND *hwndDesktop*; /* handle of the desktop */
ULONG *flOptions*; /* color options */
ULONG *flFormat*; /* format options */
COLOR *clrFirst*; /* first color to set */
ULONG *cclr*; /* number of colors to set */
PCOLOR *pcclr*; /* address of color definitions */

The **WinSetSysColors** function sets system color values. This function sends a WM_SYSCOLORCHANGE message to all main windows in the system 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 by using the WinQuerySysColor function.

After the WM_SYSCOLORCHANGE messages are sent, all windows in the system are invalidated so that they will be redrawn with the new system colors.

WinSetSysColors does not write any system color changes to the *os2.ini* file.

Parameters

hwndDesktop Identifies the desktop window. This parameter can be HWND_DESKTOP or the desktop window handle.

flOptions Specifies the following options:

Value	Meaning
LCOL_PURECOLOR	Indicates that color dithering should not be used to create colors not available in the physical palette. If this option is set, only pure colors will be used and no dithering will be done.
LCOL_RESET	Indicates that the system colors are all to be reset to default before processing the remainder of the data in this function.

flFormat Specifies the format of entries in the table, as follows:

Value	Meaning
LCOLF_CONSECRGB	Array of RGB values that correspond to color indexes. Each entry is 4 bytes.
LCOLF_INDRGB	Array of (index, RGB) values. Each pair of entries is 8 bytes—4 bytes index and 4 bytes color value.

clrFirst Specifies the starting system color index (this parameter is only relevant for the LCOLF_CONSECRGB format). The following system color indexes are defined (each successive index is one larger than its predecessor):

Value	Meaning
SYSCLR_ACTIVEBORDER	Border fill of active window
SYSCLR_ACTIVETITLE	Title bar of active window
SYSCLR_APPWORKSPACE	Background of certain main windows
SYSCLR_BACKGROUND	Screen background

Value	Meaning
SYSCLR_HELPBACKGROUND	Background of help panels
SYSCLR_HELPHILITE	Highlight of help text
SYSCLR_HELPTEXT	Help text
SYSCLR_INACTIVEBORDER	Border fill of inactive window
SYSCLR_INACTIVETITLE	Title bar of inactive window
SYSCLR_MENU	Menu background
SYSCLR_MENUTEXT	Menu text
SYSCLR_SCROLLBAR	Scroll bar
SYSCLR_TITLETEXT	Title text
SYSCLR_WINDOW	Window background
SYSCLR_WINDOWFRAME	Window border line
SYSCLR_WINDOWSTATICTEXT	Static text
SYSCLR_WINDOWTEXT	Window text

clr Specifies the number of elements supplied in *pclr*. This parameter may be zero if, for example, the color table is merely to be reset to the default. For LCOLF_INDRGB, this parameter must be an even number. The constant SYSCLR_CSYS_COLORS is set to the total number of system colors.

pclr Specifies the start address of the application data area containing the color-table definition data. The format depends on the value of the *flFormat* parameter. Each color value is a 4-byte integer. The low byte is the blue intensity value (0x000000FF), the second byte is the green intensity value (0x0000FF00), and the third byte is the red intensity value (0x00FF0000). The intensity for each color may range between 0 and 255.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinQuerySysColor

■ WinSetSysModalWindow

BOOL WinSetSysModalWindow(*hwndDesktop*, *hwnd*)

HWND *hwndDesktop*; /* handle of the desktop */
HWND *hwnd*; /* handle of the window that becomes system modal */

The WinSetSysModalWindow function makes a window the system modal window or ends the system modal state.

Parameters *hwndDesktop* Identifies the desktop window. This parameter can be HWND_DESKTOP or the desktop window handle.

hwnd Identifies the window that is to become the system modal window. If this parameter is NULL, the system modal state terminates and input processing returns to its normal state.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments Input processing can enter a system modal state. In this state, all mouse and keyboard input is directed to a special main window, called the system modal window, or to one of its child windows. All other main windows act as if they are disabled and no interaction is possible with them.

The disabled windows are not actually disabled but rather 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, replacing the previous system modal window, which then becomes a noninteractive window. When the system modal window is destroyed, the window activated as a result becomes the system modal window.

This function should be called only while processing keyboard or mouse input. The new system modal window is not locked during the processing of this function.

See Also WinQuerySysModalWindow

■ WinSetSysValue

```

BOOL WinSetSysValue(hwndDesktop, iSysValue, IValue)
HWND hwndDesktop;    /* handle of the desktop */
SHORT iSysValue;     /* system value to change */
LONG IValue;         /* new system value */

```

The `WinSetSysValue` function sets the system value.

Parameters

hwndDesktop Identifies the desktop window. This parameter can be `HWND_DESKTOP` or the desktop window handle.

iSysValue Specifies the system value. This parameter must be a valid SV value. For a complete list of possible system values, see the following “Comments” section.

IValue Specifies the system value. Durations are in milliseconds. Frequencies are in hertz, with values between `25H` and `7FFFH` being valid.

Return Value The return value is `TRUE` if the system value is successfully set. Otherwise, it is `FALSE`, indicating that an error occurred.

Comments The following list describes the system values:

Value	Meaning
<code>SV_CMOUSEBUTTONS</code>	Specifies the number of mouse buttons: 1, 2, or 3.
<code>SV_MOUSEPRESENT</code>	Specifies whether the mouse is present. A value of <code>TRUE</code> means the mouse is present.
<code>SV_SWAPBUTTON</code>	Specifies if the mouse buttons are swapped. <code>TRUE</code> if mouse buttons are swapped.

Value	Meaning
SV_CXDBLCLK	Specifies the mouse double click horizontal spacing. The horizontal spatial requirement for considering two mouse clicks a double click is met if the horizontal distance between two mouse clicks is less than this value.
SV_CYDBLCLK	Specifies the mouse double click vertical spacing. The vertical spatial requirement for considering two mouse clicks a double click is met if the vertical distance between two mouse clicks is less than this value.
SV_DBLCLKTIME	Specifies the mouse double click time in milliseconds. The temporal requirement for considering two mouse clicks a double click is met if the time between two mouse clicks is less than this value.
SV_CXSIZEBORDER	Specifies the count of pels along the x-axis in the left and right parts of a window sizing border.
SV_CYSIZEBORDER	Specifies the count of pels along the y-axis in the top and bottom sections of a window sizing border.
SV_ALARM	Specifies whether calls to WinAlarm generate a sound. A value of TRUE means sound is generated.
SV_CURSORRATE	Specifies the cursor blinking rate in milliseconds. The blinking rate is the time that the cursor remains visible or invisible. Twice this value is the length of a complete cursor visible/invisible cycle.
SV_FIRSTSCROLLRATE	Specifies the delay (in milliseconds) until scroll bar autorepeat activity begins when the mouse is held down on a scroll bar arrow or within a scroll bar.
SV_SCROLLRATE	Specifies the delay (in milliseconds) between scroll bar autorepeat events.
SV_NUMBEREDLISTS	Reserved.
SV_ERRORFREQ	Specifies the frequency (in cycles per second) of a WinAlarm WA_ERROR sound.
SV_NOTEFREQ	Specifies the frequency (in cycles per second) of a WinAlarm WA_NOTE sound.
SV_WARNINGFREQ	Specifies the frequency (in cycles per second) of a WinAlarm WA_WARNING sound.
SV_ERRORDURATION	Specifies the duration (in milliseconds) of a WinAlarm WA_ERROR sound.
SV_NOTEDURATION	Specifies the duration (in milliseconds) of a WinAlarm WA_NOTE sound.

Value	Meaning
SV_WARNINGDURATION	Specifies the duration (in milliseconds) of a WinAlarm WA_WARNING sound.
SV_CXSCREEN	Specifies the count of pels along the screen's x-axis.
SV_CYSCREEN	Specifies the count of pels along the screen's y-axis.
SV_CXVSCROLL	Specifies the count of pels along the x-axis of a vertical scroll bar.
SV_CYHSCROLL	Specifies the count of pels along the y-axis of a horizontal scroll bar.
SV_CXHSCROLLARROW	Specifies the count of pels along the x-axis of a horizontal scroll bar arrow.
SV_CYVSCROLLARROW	Specifies the count of pels along the y-axis of a vertical scroll bar arrow.
SV_CXBORDER	Specifies the count of pels along the x-axis of a window border.
SV_CYBORDER	Specifies the count of pels along the y-axis of a window border.
SV_CXDLGFRAME	Specifies the count of pels along the x-axis of a dialog frame.
SV_CYDLGFRAME	Specifies the count of pels along the y-axis of a dialog frame.
SV_CYTITLEBAR	Specifies the count of pels along the y-axis of a title bar window.
SV_CXHSLIDER	Specifies the count of pels along the x-axis of a horizontal scroll bar slider.
SV_CYVSLIDER	Specifies the count of pels along the y-axis of a vertical scroll bar slider.
SV_CXMINMAXBUTTON	Specifies the width (in pels) of a minimize/maximize button.
SV_CYMINMAXBUTTON	Specifies the height (in pels) of a minimize/maximize button.
SV_CYMENU	Specifies the height (in pels) of a menu.
SV_CXFULLSCREEN	Specifies the count of pels along the x-axis of a maximized frame window's client window.
SV_CYFULLSCREEN	Specifies the count of pels along the y-axis of a maximized frame window's client window.
SV_CXICON	Specifies the count of pels along an icon's x-axis.
SV_CYICON	Specifies the count of pels along an icon's y-axis.
SV_CXPOINTER	Specifies the count of pels along the mouse pointer's x-axis.
SV_CYPOINTER	Specifies the count of pels along the mouse pointer's y-axis.

Value	Meaning
SV_DEBUG	Reserved.
SV_CURSORLEVEL	Specifies the cursor display count. The cursor is visible only when the display count is zero.
SV_POINTERLEVEL	Specifies the mouse pointer display count. The mouse is visible only when the display count is zero.
SV_TRACKRECTLEVEL	Specifies the tracking rectangle display count. The tracking rectangle is visible only when the display count is zero.
SV_CTIMERS	Specifies the number of available timers.
SV_CXBYTEALIGN	Set by a device driver at initialization time to indicate any horizontal alignment that is more efficient for the driver.
SV_CYBYTEALIGN	Set by a device driver at initialization time to indicate any vertical alignment that is more efficient for the driver.
SV_CSYSVALUES	Specifies the number of system values.

See Also WinQuerySysValue

■ WinSetWindowBits

BOOL WinSetWindowBits (*hwnd*, *index*, *flData*, *flMask*)

HWND *hwnd*; /* handle of the window */
SHORT *index*; /* index of the bits */
ULONG *flData*; /* data to set */
ULONG *flMask*; /* mask of bits to set */

The **WinSetWindowBits** function sets particular bits in the reserved memory of a window.

Parameters

hwnd Identifies the window whose reserved memory is to be changed.

index Specifies the zero-based index of the **ULONG** value to set. Valid values are in the range zero through the number of bytes of window data (for example, a value of 8 would be an index to the third long integer), or any of the **QWL** values described in the **WinSetWindowULong** function.

flData Specifies the data to be written into the window's reserved memory.

flMask Specifies a mask value. The mask contains 1 where data is to be written and 0 where the data is to be unchanged.

Return Value The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also WinSetWindowULong

■ WinSetWindowPos

BOOL WinSetWindowPos(*hwnd*, *hwndInsertBehind*, *x*, *y*, *cx*, *cy*, *fs*)

HWND *hwnd*; /* handle of the window being set */
HWND *hwndInsertBehind*; /* placement-order handle */
SHORT *x*; /* horizontal position */
SHORT *y*; /* vertical position */
SHORT *cx*; /* width */
SHORT *cy*; /* height */
USHORT *fs*; /* window-positioning flags */

The **WinSetWindowPos** function sets the position of a window.

Parameters

hwnd Identifies the window whose position is being set.

hwndInsertBehind Identifies relative window-placement order. This parameter is ignored if the *fs* parameter's **SWP_ZORDER** option is not selected. Values that can be specified are as follows:

Value	Meaning
HWND_BOTTOM	Places the <i>hwnd</i> window behind all sibling windows.
HWND_TOP	Places the <i>hwnd</i> window on top of all sibling windows.
Other	Identifies the sibling window behind which the <i>hwnd</i> window is to be placed.

x Specifies the *x* position of the *hwnd* window in window coordinates relative to the lower-left corner of its parent window. This parameter is ignored if the *fs* parameter's **SWP_MOVE** option is not selected.

y Specifies the *y* position of the *hwnd* window in window coordinates relative to the lower-left corner of its parent window. This parameter is ignored if the *fs* parameter's **SWP_MOVE** option is not selected.

cx Specifies the horizontal window size (in device units). This parameter is ignored if the *fs* parameter's **SWP_SIZE** option is not selected.

cy Specifies the vertical window size (in device units). This parameter is ignored if the *fs* parameter's **SWP_SIZE** option is not selected.

fs Identifies the window-positioning options. One or more of the following options can be specified:

Value	Meaning
SWP_ACTIVATE	Causes the window to be activated and the focus to be set to the window that lost the focus the last time the frame window was deactivated. The activated window may not become the top window if it owns other frame windows.
SWP_DEACTIVATE	Deactivate the window, if it is the active window.
SWP_EXTSTATECHANGE	This flag is for application use. It is used to pass an additional flag to the portion of code that is handling messages.

Value	Meaning
SWP_FOCUSACTIVATE	Specifies that a frame window is receiving the focus. This flag is set so that an application that is processing the WM_ADJUSTWINDOWPOS message can tell if the message was sent as the result of a focus change.
SWP_FOCUSDEACTIVATE	Specifies that a frame window is losing the focus.
SWP_HIDE	Specifies that the window is to be hidden when created.
SWP_MAXIMIZE	With SWP_MINIMIZE, causes a window to be minimized, maximized, or restored. SWP_MAXIMIZE and SWP_MINIMIZE are mutually exclusive. If either SWP_MINIMIZE or SWP_MAXIMIZE is specified, then both SWP_MOVE and SWP_SIZE must also be specified. WinSetWindowPos and WinSetMultWindowPos depend on the previous state of the window; these flags cause the appropriate state to be toggled, as follows: the x, y, cx, and cy parameters specify the size and position to which the window will be restored if it is subsequently restored. This should be the normal size of the window.
SWP_MINIMIZE	See SWP_MAXIMIZE.
SWP_MOVE	Change the window x,y position.
SWP_NOADJUST	Do not send a WM_ADJUSTWINDOWPOS message to the window while processing (in other words, don't give the window a chance to readjust itself).
SWP_NOREDRAW	Do not redraw changes.
SWP_RESTORE	Restore a minimized or maximized window.
SWP_SHOW	Specifies that the window is to be shown when created.
SWP_SIZE	Change the window size.
SWP_ZORDER	Change the relative window placement.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments If a window created with the CS_SAVEBITS style is moved, reduced in size, or hidden, the saved screen image 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, a WM_CALCVALIDRECTS message 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 orders 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, then `SWP_SHOW` is turned off. If `SWP_SIZE` is present and the new size is equal to the previous size, `SWP_SIZE` is turned off. If the operations will create new results, the information is calculated and stored. For example, if being sized or moved, 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 being sized or moved. It is also at this point that the `WM_CALCVALIDRECTS` message is sent to any window that is being sized and does not have the `CS_SIZEREDRAW` window style.

When the new window state is 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, and so forth. 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 repainted are repainted and the entire process is complete.

If a synchronous-paint parent window has a size-sensitive area displayed that includes synchronous-paint child windows, the parent window will reposition those windows when it receives the `WM_SIZE` message. Their invalid regions will be added to the parent window's invalid region, resulting in one update after the parent window's `WM_SIZE` message, rather than many independent and subsequently duplicated updates.

The following messages are sent by this function:

Value	Meaning
<code>WM_CALCVALIDRECTS</code>	Sent to determine the area of a window that it may be possible to preserve as the window is sized.
<code>WM_SIZE</code>	Sent if the size of the window has changed, after the change has been effected.
<code>WM_MOVE</code>	Sent when a window with <code>CS_MOVENOTIFY</code> class style moves its absolute position.
<code>WM_ACTIVATE</code>	Sent if a different window becomes the active window. For more information, see the <code>WinSetActiveWindow</code> function.
<code>WM_ADJUSTWINDOWPOS</code>	Sent if <code>SWP_NOADJUST</code> is not specified. The message's <i>mp1</i> parameter points to an <code>SWP</code> structure that has been filled in by the <code>WinSetWindowPos</code> function with the proposed move/size data. The window can adjust this new position by changing the contents of the <code>SWP</code> structure.

Example

This example gets the dimensions of the desktop window, and calls `WinSetWindowPos` to place the application's frame window in the upper left corner. By positioning the window relative to the desktop window, the window position is

device-independent; it will work on any display adapter no matter what the vertical and horizontal resolution is.

```
RECTL rcl;

WinQueryWindowRect (HWND_DESKTOP, &rcl);
WinSetWindowPos (hwndFrame, HWND_TOP,
    rcl.xLeft,                               /* x pos */
    rcl.yTop - 60,                           /* y pos */
    140,                                     /* x size */
    60,                                     /* y size */
    SWP_ACTIVATE | SWP_MOVE | SWP_SIZE | SWP_SHOW); /* flags */
```

See Also WinSetActiveWindow, WinSetMultWindowPos

■ WinSetWindowPtr

```
BOOL WinSetWindowPtr (hwnd, index, p)
HWND hwnd;        /* handle of the window */
SHORT index;     /* index of the reserved memory */
PVOID p;         /* pointer to place into reserved memory */
```

The **WinSetWindowPtr** function places a pointer value into the reserved memory of a window.

Parameters *hwnd* Identifies the window whose reserved memory will be changed.
 index Specifies the zero-based index of the pointer value to set. Valid values are in the range zero through the number of bytes of window data (for example, a value of 8 would be an index to the third pointer). The value `QWP_PFNWP` can be used as the index for the address of the window procedure for the window.

p Specifies the pointer to store in the window's reserved memory.

Return Value The return value is `TRUE` if the function is successful or `FALSE` if an error occurs.

See Also WinQueryWindowPtr, WinSetWindowULong

■ WinSetWindowText

```
BOOL WinSetWindowText (hwnd, pszText)
HWND hwnd;        /* handle of the window */
PSZ pszText;     /* points to the text to set */
```

The **WinSetWindowText** function sets the window text for a window to the specified text. This function sends a `WM_SETWINDOWPARAMS` message to the *hwnd* window.

If this function is called with a frame-window handle, the text of the title-bar-frame control is changed.

Parameters *hwnd* Identifies the window to set the text for.
 pszText Points to the window text.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinQueryWindowText

■ WinSetWindowULong

BOOL WinSetWindowULong(*hwnd*, *index*, *ul*)

HWND *hwnd*; /* handle of the window */

SHORT *index*; /* index into reserved memory */

ULONG *ul*; /* value to place in reserved memory */

The WinSetWindowULong function places an unsigned long integer value into the reserved memory of a window.

Parameters

hwnd Identifies the window whose reserved memory is to be changed.

index Specifies the zero-based index of the ULONG value to set. Valid values are in the range zero through the number of bytes of window data (for example, a value of 8 would be an index to the third long integer), or any of the following QWL values:

Value	Meaning
QWL_HHEAP	Handle of the heap.
QWL_HMQ	Handle of the message queue of the window.
QWL_HWNDFOCUSSAVE	Handle of the window that last had the focus.
QWL_STYLE	Window style.
QWL_USER	ULONG value present in windows of the following preregistered window classes:
	WC_BUTTON
	WC_DIALOG
	WC_ENTRYFIELD
	WC_FRAME
	WC_LISTBOX
	WC_MENU
	WC_SCROLLBAR
	WC_STATIC

This value can be used to retrieve application-specific data in controls.

ul Specifies the unsigned long integer to place in the window's reserved memory.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments The specified index is valid only if all the bytes referenced are within the reserved memory. For example, this function would fail if an index value of zero was specified and only 2 bytes had been reserved.

See Also WinQueryWindowULong, WinRegisterClass, WinSetWindowBits, WinSetWindowUShort

■ WinSetWindowUShort

```

BOOL WinSetWindowUShort(hwnd, index, us)
HWND hwnd; /* handle of the window */
SHORT index; /* index into reserved memory */
USHORT us; /* value to place in reserved memory */

```

The **WinSetWindowUShort** function places an unsigned short integer into the reserved memory of a window.

Parameters

hwnd Identifies the window whose reserved memory is to be changed.

index Specifies the zero-based index of the **USHORT** value to set. Valid values are in the range zero through the number of bytes of window data (for example, a value of 8 would be an index to the fifth short integer), or any of the following QWS values:

Value	Meaning
QWS_ID	Index of the window identifier (as passed by the WinCreateWindow function).
QWS_USER	Index of an unsigned short value present in windows of the following preregistered window classes: WC_BUTTON WC_DIALOG WC_ENTRYFIELD WC_FRAME WC_LISTBOX WC_MENU WC_SCROLLBAR WC_STATIC

This value can be used to set application-specific data in controls.

us Specifies the unsigned short integer to place in the window's reserved memory.

Return Value

The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also

WinCreateWindow, **WinQueryWindowUShort**, **WinRegisterClass**, **WinSetWindowULong**

■ WinShowCursor

```

BOOL WinShowCursor(hwnd, fShow)
HWND hwnd; /* handle of the window */
BOOL fShow; /* show/hide flag */

```

The **WinShowCursor** function displays or hides the cursor associated with a specified window. A cursor show level count is kept internally. (You can retrieve this value using the **WinQuerySysValue** function with **SV_CURSORLEVEL** as the system value.) It is incremented by a hide operation and decremented by a show operation. The cursor is visible only if the count is zero.

Parameters	<i>hwnd</i> Identifies the window to which the cursor belongs. <i>fShow</i> Specifies whether the cursor is shown or hidden. If TRUE, the cursor is made visible. If FALSE, the cursor is made invisible.
Return Value	The return value is TRUE if the function is successful or FALSE if an error occurs.
See Also	WinCreateCursor, WinDestroyCursor, WinQueryCursorInfo, WinQuerySysValue

■ WinShowPointer

```

BOOL WinShowPointer(hwndDesktop, fShow)
HWND hwndDesktop; /* handle of the desktop */
BOOL fShow; /* show/hide flag */

```

The **WinShowPointer** function shows or hides the mouse pointer. A pointer show level count is kept internally. (You can retrieve this value using the **WinQuerySysValue** function with **SV_POINTERLEVEL** as the system value.) It is incremented by a hide operation and decremented by a show operation. The pointer is visible only if the count is zero. If a mouse exists, the initial setting of the pointer display level is 0. Otherwise, it is 1.

Parameters	<i>hwndDesktop</i> Identifies the desktop window. This parameter can be HWND_DESKTOP or the desktop window handle. <i>fShow</i> Specifies whether the pointer is shown or hidden. If TRUE, the pointer is made visible. If FALSE, the pointer is made invisible.
Return Value	The return value is TRUE if the function is successful or FALSE if an error occurs.
See Also	WinQuerySysValue

■ WinShowTrackRect

```

BOOL WinShowTrackRect(hwnd, fShow)
HWND hwnd; /* handle of the window */
BOOL fShow; /* show/hide flag */

```

The **WinShowTrackRect** function hides or shows the tracking rectangle. A track rectangle show level count is kept internally. (You can retrieve this value using the **WinQuerySysValue** function with **SV_TRACTRECTLEVEL** as the system value.) It is incremented by a hide operation and decremented by a show operation. The track rectangle is visible only if the count is zero.

Parameters	<i>hwnd</i> Identifies the window passed to the WinTrackRect function. <i>fShow</i> Specifies whether the rectangle is shown or hidden. If <i>fShow</i> is TRUE, the function shows the tracking rectangle. If <i>fShow</i> is FALSE, the function hides the tracking rectangle.
Return Value	The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments An application must call this function to hide a rectangle it is tracking if there is a possibility of corrupting the track rectangle while drawing (showing it afterward). Since `rcTrack` in the `TRACKINFO` structure is updating continuously, the application can examine the current tracking rectangle coordinates to determine whether temporary hiding is necessary.

An application needs to call `WinShowTrackRect` only during asynchronous drawing. If an application is drawing on one thread and issuing the `WinTrackRect` function on another, pieces of a tracking rectangle may be left behind. The drawing thread should call `WinShowTrackRect` when tracking may be in progress. The application should provide for communication between the two threads to ensure that if one thread is tracking, the drawing thread will issue `WinShowTrackRect`. This can be achieved with a semaphore.

See Also `WinQuerySysValue`, `WinTrackRect`

■ WinShowWindow

```
BOOL WinShowWindow(hwnd, fShow)
HWND hwnd; /* handle of the window */
BOOL fShow; /* show/hide flag */
```

The `WinShowWindow` function sets the window visibility state.

Parameters *hwnd* Identifies the window whose visibility state is being set.
fShow Identifies the new visibility state. If *fShow* is `TRUE`, the function sets the window state to visible. If *fShow* is `FALSE`, the function sets the window state to invisible.

Return Value The return value is `TRUE` if the function is successful or `FALSE` if an error occurs.

Comments 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 displayed, as long as the window or any of the windows in the parent chain 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 displayed.

If the value of the `WS_VISIBLE` style bit has been changed, the `WM_WINDOWPOSCHANGED` message is sent to the window of the *hwnd* parameter before the function returns.

Drawing to a window with a `WS_VISIBLE` style will not be displayed if the window is covered by other windows or clipped by its parent.

See Also `WinIsWindowVisible`

■ WinStartTimer

USHORT WinStartTimer(*hab*, *hwnd*, *idTimer*, *lTimeOut*)

HAB *hab*; /* handle of the anchor block */
HWND *hwnd*; /* handle of the window */
USHORT *idTimer*; /* timer identifier */
USHORT *lTimeOut*; /* time-out value */

The **WinStartTimer** function starts a timer. A time-out value is specified, and every time a time-out occurs, a **WM_TIMER** message is posted to the specified window.

A second call to **WinStartTimer** for a timer that already exists will reset that timer.

Parameters

hab Identifies the anchor block.

hwnd Identifies the window that is part of the timer identification. If the *hwnd* parameter is **NULL**, then the *idTimer* parameter is ignored and **WinStartTimer** returns a unique nonzero identification value that identifies the timer. The timer message is posted in the queue associated with the current thread, with the *hwnd* parameter equal to **NULL**.

idTimer Identifies the timer. If *hwnd* is **NULL**, this parameter is ignored.

lTimeOut Specifies the timer delay, in milliseconds. An *lTimeOut* value of zero will cause the timer to time out as fast as possible; generally, this is about 1/18 of a second.

Return Value

The return value is **TRUE** if the function is successful and if *hwnd* is not **NULL**. Otherwise, it is **FALSE**, indicating that an error occurred. If *hwnd* is **NULL**, the return value is a unique nonzero value, or zero if an error occurs.

Comments

If the *hwnd* parameter is **NULL**, then the return value from this function must be used as the *idTimer* parameter in any subsequent call to **WinStopTimer**.

See Also

WinStopTimer

■ WinStopTimer

BOOL WinStopTimer(*hab*, *hwnd*, *idTimer*)

HAB *hab*; /* handle of the anchor block */
HWND *hwnd*; /* handle of the window */
USHORT *idTimer*; /* timer identifier */

The **WinStopTimer** function stops a timer. 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 the **WinGetMsg** function.

Parameters

hab Identifies the anchor block.

hwnd Identifies the window containing the timer.

idTimer Identifies the timer.

Return Value

The return value is **TRUE** if the function is successful or **FALSE** if an error occurs.

See Also

WinGetMsg, **WinStartTimer**

■ WinSubclassWindow

PFNWP WinSubclassWindow (*hwnd*, *pfnewp*)

HWND *hwnd*; /* handle of the window to subclass */

PFNWP *pfnewp*; /* address of new window procedure */

The **WinSubclassWindow** function subclasses a window by replacing its window procedure with another window procedure specified by the *pfnewp* parameter.

- Parameters** *hwnd* Identifies the window to subclass.
 pfnewp Points to the address of the procedure used to subclass the window.
- Return Value** The return value, if the function is successful, is the address of the previous window procedure that belongs to the window specified by the *hwnd* parameter. Otherwise, the return value is 0L.
- Comments** To subclass a window effectively, the new window procedure should call the old window procedure, rather than the **WinDefWindowProc** function, for those messages it does not process itself.
- To reverse the effect of subclassing, call **WinSubclassWindow** again using the old window procedure address.
- It is not possible to subclass a window created by another process.
- See Also** **WinDefWindowProc**

■ WinSubstituteStrings

SHORT WinSubstituteStrings (*hwnd*, *pszSrc*, *cchDstMax*, *pszDst*)

HWND *hwnd*; /* handle of the window */

PSZ *pszSrc*; /* address of the source string */

SHORT *cchDstMax*; /* size of destination string buffer */

PSZ *pszDst*; /* address of buffer for destination string */

The **WinSubstituteStrings** function performs a substitution process on a text string, replacing certain marker characters with text supplied by the application. The **WinSubstituteStrings** function is particularly useful for displaying variable information in dialog boxes, menus, and other user-interface functions. Variable information can include things such as filenames, which cannot be statically declared within resource files.

This function is called by the system while creating the child windows in a dialog box. It allows the child windows to perform substitutions in their window text.

- Parameters** *hwnd* Identifies the window that processes the function.
 pszSrc Points to the null-terminated text string to perform the substitution.
 cchDstMax Specifies the maximum number of characters that can be returned in the *pszDst* parameter.
 pszDst Points to the null-terminated text string produced by the substitution process. The string is truncated if it would otherwise contain more than the number of characters specified by the *cchDstMax* parameter. When truncation occurs, the last character of the truncated string is always the NULL termination character.

- Return Value** The return value is the number of characters returned in *pszDst*, not including the terminating NULL character. The maximum value is (*cchDstMax* - 1). It is zero if an error occurred.
- Comments** When a string of the form *%n* is encountered in the source string, where *n* is a digit from 0 through 9, a WM_SUBSTITUTESTRING message is sent to the specified window. This message returns a text string to use as a substitution for the *%n* in the destination string, which is otherwise an exact copy of the source string.
- If *%* is encountered in the source, *%* is copied to the destination, but no other substitution occurs. If *%x* occurs in the source, where *x* is neither a digit nor *%*, then the source is copied to the destination unchanged.
- The source and destination strings must not overlap in memory.
- To use this function, your application must process WM_SUBSTITUTESTRING messages and perform the requested string substitution.

■ WinSubtractRect

```

BOOL WinSubtractRect(hab, prclDst, prclSrc1, prclSrc2)
HAB hab;          /* handle of the anchor block          */
PRECTL prclDst;   /* address of the destination rectangle structure */
PRECTL prclSrc1; /* address of the first rectangle structure      */
PRECTL prclSrc2; /* address of the second rectangle structure     */

```

The WinSubtractRect function subtracts one rectangle from another by subtracting the *prclSrc2* parameter from the *prclSrc1* parameter. Subtracting one rectangle from another does not always result in a rectangular area; in this case, WinSubtractRect returns *prclSrc1* in the *prclDst* parameter. For this reason, WinSubtractRect provides only an approximation of subtraction. However, the area described by *prclDst* is always greater than or equal to the “true” result of the subtraction. You can use the GpiCombineRegion function to calculate the true result of the subtraction of two rectangular areas, although WinSubtractRect does it much faster.

Parameters *hab* Identifies an anchor block.

prclDst Points to a RECTL structure that contains the result of the subtraction of the *prclSrc2* parameter from the *prclSrc1* parameter. The RECTL structure has the following form:

```

typedef struct _RECTL {
    LONG xLeft;
    LONG yBottom;
    LONG xRight;
    LONG yTop;
} RECTL;

```

For a full description, see Chapter 4, “Types, Macros, Structures.”

prclSrc1 Points to a RECTL structure that contains the first source rectangle.

prclSrc2 Points to a RECTL structure that contains the second source rectangle.

Return Value The return value is TRUE if the *prclDst* parameter points to a nonempty rectangle. Otherwise, it is FALSE, indicating that *prclDst* is an empty rectangle.

See Also GpiCombineRegion, WinUnionRect

■ WinTerminate

BOOL WinTerminate (*hab*)

HAB *hab*; /* handle of the anchor block */

The **WinTerminate** function terminates an application thread's use of Presentation Manager and releases all its associated resources. It is recommended that you call this function prior to termination of your application. However, if it is not issued, all Presentation Manager resources allocated to the thread are deallocated when the program terminates—whether normally or abnormally—by Presentation Manager code executed as part of the exit-list processing.

Parameters *hab* Identifies the anchor block.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinInitialize

■ WinThrow

VOID WinThrow (*pctchbf*, *sErrorReturn*)

PCATCHBUF *pctchbf*; /* address of structure with execution environment */

SHORT *sErrorReturn*; /* error code to return */

The **WinThrow** function restores the execution environment to the values saved in the buffer pointed to by the *pctchbf* parameter. Execution then transfers to the **WinCatch** function that copied the environment to *pctchbf*.

Parameters *pctchbf* Points to a **CATCHBUF** structure that contains the execution environment. It must have been set by a previous **WinCatch** function call. The **CATCHBUF** structure has the following form:

```
typedef struct _CATCHBUF {
    ULONG reserved[4];
} CATCHBUF;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

sErrorReturn Specifies the value to be returned to the **WinCatch** function. The meaning of the value is determined by the application.

Return Value This function does not return a value.

Comments The routine that calls **WinCatch** is responsible for freeing any resources allocated between the time **WinCatch** was called and the time **WinThrow** was called.

Example This example calls **WinCatch** to save the current execution environment before calling a recursive sort function. The first return from **WinCatch** is zero. If the *doSort* function calls **WinThrow**, execution will again return to the **WinCatch** function. This time, **WinCatch** will return the **STACKOVERFLOW** error

passed by the *doSort* function. The *doSort* function is recursive, that is, it calls itself. It maintains a variable, *usStackCheck*, that is used to check to see how much stack space has been used. If more than 3K of the stack has been used, it calls *WinThrow* to drop out of all the nested function calls back into the function that called *WinCatch*.

```
USHORT usStackCheck
CATCHBUF ctchbf;

main() {
    SHORT sErrorReturn;

    sErrorReturn = WinCatch(&ctchbf); /* save execution environment */
    if (sErrorReturn) {
        . /* error processing */
    }
    usStackCheck = 0; /* initialize stack usage count */
    doSort(1, 1000); /* call sort function */
}

VOID doSort(sLeft, sRight)
SHORT sLeft, sRight;
{
    SHORT i, sLast;

    /*
     * check to see if more than 3K of the stack has been used, and if
     * so, call WinThrow to drop back into the original calling program
     */

    usStackCheck += 10;
    if (usStackCheck > (3 * 1024))
        WinThrow(&ctchbf, STACKOVERFLOW);

    . /* sorting algorithm */
    doSort(sLeft, sLast - 1); /* note recursive call */
    usStackCheck -= 10; /* update stack check variable */
}
```

See Also WinCatch

■ WinTrackRect

```
BOOL WinTrackRect(hwnd, hps, pti)
HWND hwnd; /* handle of the window */
HPS hps; /* presentation-space handle */
PTRACKINFO pti; /* address of structure for tracking information */
```

The *WinTrackRect* function draws a tracking rectangle.

Parameters *hwnd* Identifies the window in which tracking is to take place. If *hwnd* identifies the desktop window or *HWND_DESKTOP*, tracking will take place over the entire screen. It is assumed that the style of this window is not *WS_CLIPCHILDREN*.

hps Identifies the presentation space to be used for drawing the clipping rectangle. If *hps* is *NULL*, the *hwnd* parameter is used to calculate a presentation space for tracking. (It is assumed that tracking takes place within the window identified by *hwnd* and that the style of this window is not set to *WS_CLIPCHILDREN*.) 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.

pti Points to a `TRACKINFO` structure. The `TRACKINFO` structure has the following form:

```
typedef struct _TRACKINFO {
    SHORT  cxBorder;
    SHORT  cyBorder;
    SHORT  cxGrid;
    SHORT  cyGrid;
    SHORT  cxKeyboard;
    SHORT  cyKeyboard;
    RECTL  rc1Track;
    RECTL  rc1Boundary;
    POINTL pt1MinTrackSize;
    POINTL pt1MaxTrackSize;
    USHORT fs;
} TRACKINFO;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

The return value is `TRUE` if the function is successful. Otherwise, it is `FALSE`, indicating that an error occurred or that the mouse was already captured when `WinTrackRect` was called. Only one tracking rectangle may be in use at a time.

Comments

This function provides general-purpose mouse tracking. `WinTrackRect` draws a rectangle and allows the user to position the entire rectangle or make a specific side or corner smaller or larger. The resulting rectangle is then returned to the application, which can use this new information for size and position data. For example, to move and size windows by using the wide sizing borders, the window manager interface simply calls `WinTrackRect`.

`WinTrackRect` allows the caller to control such limiting values as the following:

- A maximum and minimum tracking size
- The widths of the tracking rectangle’s sides

If the `TF_SETPOINTERPOS` flag is included, the mouse pointer is positioned at the center of the tracking rectangle. Otherwise, the mouse pointer is not moved from its current position. At this point, there is an established distance between the mouse 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 mouse pointer is repositioned along with the tracking rectangle’s new size or position.

During tracking, the following keys are active:

Value	Meaning
ENTER	Accepts the new position or size.
LEFT	Moves the mouse pointer and tracking rectangle left.
UP	Moves the mouse pointer and tracking rectangle up.
RIGHT	Moves the mouse pointer and tracking rectangle right.
DOWN	Moves the mouse pointer and tracking rectangle down.
ESC	Cancels the current tracking operation. In this case, the value of the tracking rectangle is undefined on exit.

The mouse and the keyboard interface can be intermixed. The caller need not include the `TF_SETPOINTERPOS` flag to be able to use the keyboard interface; this simply initializes the position of the mouse pointer.

Tracking movements using the keyboard arrow keys are in increments of the `cxGrid` and `cyGrid` fields, regardless of whether `TF_GRID` is specified. If `TF_GRID` is specified, the interior of the tracking rectangle is allowed only on multiples of `cxGrid` and `cyGrid`. The default value for `cxGrid` is the system-font character width, and the default value for `cyGrid` is half the height of the system font.

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 update while part of the tracking rectangle is above it.

Since 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 fragments of the tracking rectangle can be left behind. If the window doing the drawing is clipped from the window the tracking is occurring in, there is no problem.

To prevent a window that is currently processing a `WM_PAINT` message from drawing over the tracking rectangle, MS OS/2 treats the tracking rectangle as a system-wide resource, only one of which can be in use at any one time. If there is a risk of the currently updating window drawing on the tracking rectangle, MS OS/2 removes the tracking rectangle while that window and its child windows update, and then replaces it. This is done specifically by the `WinBeginPaint` and `WinEndPaint` functions. If the tracking rectangle overlaps, it is removed by `WinBeginPaint`. With the `WinEndPaint` function, all child windows are updated by using the `WinUpdateWindow` function before the tracking rectangle is redrawn.

`WinTrackRect` has a modal loop within its function. The modal loop has a `HK_MSGFILTER` hook and hook code `MSGF_TRACK`. For an explanation of this hook type, see the `WinSetHook` function.

The rectangle tracked by `WinTrackRect` is guaranteed to be within the specified tracking bounds and dimensions. If the rectangle passed is out of these bounds, or is too large or too small, it is modified to a rectangle that meets those limits.

See Also

`WinBeginPaint`, `WinEndPaint`, `WinSetHook`, `WinUpdateWindow`

■ WinTranslateAccel

```

BOOL WinTranslateAccel(hab, hwnd, haccel, pqmsg)
HAB hab;           /* handle of the anchor block */
HWND hwnd;        /* handle of the window */
HACCEL haccel;    /* handle of the accelerator table */
PQMSG pqmsg;      /* address of structure with message */

```

The `WinTranslateAccel` function translates a `WM_CHAR` message. If it is a `WM_CHAR` message in the specified accelerator table, `WinTranslateAccel` translates the message pointed to by the `pqmsg` parameter. The message is translated into a `WM_COMMAND`, `WM_SYSCOMMAND`, or `WM_HELP` message.

Parameters

hab Identifies the anchor block.

hwnd Identifies the destination window. Normally, this parameter identifies a frame window.

haccel Identifies the accelerator table.

pqmsg Points to a QMSG structure that contains the message to be translated. The QMSG structure has the following form:

```
typedef struct _QMSG {
    HWND    hwnd;
    USHORT  msg;
    MPARAM  mp1;
    MPARAM  mp2;
    ULONG   time;
    POINTL  pt1;
} QMSG;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

The return value is TRUE if the function is successful—that is, if the message matches an accelerator in the table. Otherwise, it is FALSE, indicating that an error occurred.

Comments

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 normally 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. Since the message is physically changed by WinTranslateAccel, applications will not receive the WM_CHAR messages that resulted in WM_COMMAND, WM_SYSCOMMAND, or WM_HELP messages.

See Also

WinGetMsg, WinPeekMsg

■ WinUnionRect

BOOL WinUnionRect(*hab*, *prclDst*, *prclSrc1*, *prclSrc2*)

HAB *hab*; /* handle of the anchor block */

PRECTL *prclDst*; /* address of the destination rectangle structure */

PRECTL *prclSrc1*; /* address of the first rectangle structure */

PRECTL *prclSrc2*; /* address of the second rectangle structure */

The WinUnionRect function calculates a rectangle that bounds the two source rectangles.

Parameters

hab Identifies an anchor block.

prclDst Points to a RECTL structure that will receive a rectangle bounding the rectangles pointed to by the *prclSrc1* and *prclSrc2* parameters. The RECTL structure has the following form:

```
typedef struct _RECTL {
    LONG  xLeft;
    LONG  yBottom;
    LONG  xRight;
    LONG  yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

prclSrc1 Points to a RECTL structure that contains the first source rectangle.

prclSrc2 Points to a RECTL structure that contains the second source rectangle.

Return Value The return value is TRUE if *prclDst* is a nonempty rectangle. Otherwise, it is FALSE, indicating that an error occurred or that the *prclDst* rectangle is empty. If one of the source rectangles is NULL, the other is returned.

See Also WinIntersectRect, WinSubtractRect

■ WinUpdateWindow

BOOL WinUpdateWindow(*hwnd*)

HWND *hwnd*; /* handle of the window */

The WinUpdateWindow function forces a window and its associated child windows to be updated.

Parameters *hwnd* Identifies the window to be updated.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

Comments If *hwnd* is a child window of a parent window that was created without the WS_CLIPCHILDREN style, the update region of *hwnd* is removed from the update region of the parent window. This is so that after the *hwnd* window is redrawn, the parent window will not draw over what was just drawn for *hwnd*.

See Also WinInvalidateRect

■ WinUpper

USHORT WinUpper(*hab, idcp, idcc, psz*)

HAB *hab*; /* handle of the anchor block */

USHORT *idcp*; /* code-page identifier */

USHORT *idcc*; /* country-code identifier */

PSZ *psz*; /* address of the string to convert */

The WinUpper function converts a string to uppercase in place.

Parameters *hab* Identifies the anchor block.

idcp Identifies the code page. If *idcp* is NULL, the current process's code page is used.

idcc Identifies the country code. If *idcc* is NULL, the default country specified in the *config.sys* file is used.

psz Points to the string to be converted.

Return Value The return value is the length of the converted string.

See Also WinUpperChar

■ WinUpperChar

```
USHORT WinUpperChar(hab, idcp, idcc, c)
```

```
HAB hab;          /* handle of the anchor block */
USHORT idcp;     /* code-page identifier */
USHORT idcc;     /* country-code identifier */
USHORT c;        /* character to translate */
```

The `WinUpperChar` function translates a character to uppercase.

Parameters

hab Identifies the anchor block.

idcp Identifies the code page. If *idcp* is NULL, the current process's code page is used.

idcc Identifies the country code. If *idcc* is NULL, the default country specified in the `config.sys` file is used.

c Specifies the character to be translated to uppercase.

Return Value The return value is the converted character if the function is successful. Otherwise, it is zero, indicating that an error occurred.

See Also `WinUpper`

■ WinValidateRect

```
BOOL WinValidateRect(hwnd, prcl, fincludeChildren)
```

```
HWND hwnd;       /* handle of the window */
PRECTL prcl;     /* address of structure with validation rectangle */
BOOL fincludeChildren; /* inclusion flag */
```

The `WinValidateRect` function subtracts a rectangle from the update region of an asynchronous paint window, marking that part of the window as visually valid. This function has no effect on the window if any part of the window has been made invalid since the last call to a `WinBeginPaint`, `WinQueryUpdateRect`, or `WinQueryUpdateRegion` function. This function is not used for `CS_SYNCPAINT` windows.

Parameters

hwnd Identifies the window whose update region is changed. If *hwnd* is `HWND_DESKTOP`, this function applies to the whole screen (or desktop).

prcl Points to a `RECTL` structure that contains the valid rectangle. This rectangle is subtracted from the window's update region. The `RECTL` structure has the following form:

```
typedef struct _RECTL {
    LONG xLeft;
    LONG yBottom;
    LONG xRight;
    LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

fIncludeChildren Specifies the validation scope. If *fIncludeChildren* is `TRUE`, the function includes the descendants of *hwnd* in the valid rectangle. If the *fIncludeChildren* parameter is `FALSE`, the function does not include the descendants of *hwnd* in the valid rectangle.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinBeginPaint, WinQueryUpdateRect, WinQueryUpdateRegion, WinValidateRegion

■ WinValidateRegion

BOOL WinValidateRegion(*hwnd*, *hrgn*, *fIncludeChildren*)

HWND *hwnd*; /* handle of the window */

HRGN *hrgn*; /* handle of the valid region */

BOOL *fIncludeChildren*; /* inclusion flag */

The **WinValidateRegion** function subtracts a region from the update region of an asynchronous paint window, marking that part of the window as visually valid. This function has no effect on the window if any part of the window has been made invalid since the last call to a **WinBeginPaint**, **WinQueryUpdateRect**, or **WinQueryUpdateRegion** function. This function is not used for **CS_SYNCPAINT** windows.

Parameters *hwnd* Identifies the window whose update region is changed. If *hwnd* is **HWND_DESKTOP**, the function applies to the whole screen (or desktop).

hrgn Identifies the region that is subtracted from the window's update region.

fIncludeChildren Specifies the validation scope. If the *fIncludeChildren* parameter is TRUE, the function includes the descendants of *hwnd* in the valid region. If *fIncludeChildren* is FALSE, the function does not include the descendants of *hwnd* in the valid region.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinBeginPaint, WinQueryUpdateRect, WinQueryUpdateRegion, WinValidateRect

■ WinWaitMsg

BOOL WinWaitMsg(*hab*, *msgFirst*, *msgLast*)

HAB *hab*; /* handle of the anchor block */

USHORT *msgFirst*; /* first message */

USHORT *msgLast*; /* last message */

The **WinWaitMsg** function waits for a filtered message. This function waits for the first message that conforms to the filtering criteria to arrive on the message queue since the queue was last inspected by either the **WinGetMsg** or **WinPeekMsg** function. The filtering criteria are specified by the *msgFirst* and *msgLast* parameters.

Parameters *hab* Identifies the anchor block.

msgFirst Specifies the first message.

msgLast Specifies the last message.

Return Value	The return value is TRUE if the function is successful or FALSE if an error occurs.
Comments	<p>Filtering allows an application to process messages in a different order than the one in the queue. Filtering is used so the application can receive messages of a particular type only, rather than receiving other types of messages at an inconvenient point in the logic of the application. For example, when a “mouse button down” message is received, the application can use filtering to wait for the “mouse button up” message without having to process other messages.</p> <p>When using filtering, you must ensure that a message satisfying the specification of the filtering parameters can occur; otherwise, the <code>WinGetMsg</code> function cannot completely execute. For example, calling the <code>WinGetMsg</code> function with the <code>msgFilterFirst</code> and <code>msgFilterLast</code> parameters equal to <code>WM_CHAR</code> and with the <code>hwndFilter</code> parameter set to a window handle that does not have the input focus prevents <code>WinGetMsg</code> from returning.</p> <p>Keystrokes are passed to the <code>WinTranslateAccel</code> function. This means that accelerator keys are translated into <code>WM_COMMAND</code> or <code>WM_SYSCOMMAND</code> messages and are not received as <code>WM_CHAR</code> messages by the application.</p> <p>If both <code>msgFirst</code> and <code>msgLast</code> are zero, then <code>WinWaitMsg</code> will return when any message is placed in the queue. This can be used in conjunction with the <code>WinPeekMsg</code> function to create a <code>WinGetMsg</code> type loop that does not remove messages from the queue.</p> <p>The constants <code>WM_MOUSEFIRST</code> and <code>WM_MOUSELAST</code> can be used for <code>msgFirst</code> and <code>msgLast</code> to filter all but mouse messages.</p> <p>The constants <code>WM_BUTTONCLICKFIRST</code> and <code>WM_BUTTONCLICKLAST</code> can be used for <code>msgFirst</code> and <code>msgLast</code> to filter all but mouse button messages.</p> <p>The constants <code>WM_DDE_FIRST</code> and <code>WM_DDE_LAST</code> can be used for <code>msgFirst</code> and <code>msgLast</code> to filter all but dynamic data exchange messages.</p>
See Also	<code>WinGetMsg</code> , <code>WinPeekMsg</code> , <code>WinTranslateAccel</code>

■ WinWindowFromDC

HWND `WinWindowFromDC` (*hdc*)

HDC *hdc*; /* handle of the device context */

The `WinWindowFromDC` function is used to determine the window associated with a window device context, given a device context handle returned by the `WinOpenWindowDC` function. If the device context handle is not a window device context, this function returns `NULL`.

Parameters *hdc* Identifies the window device context.

Return Value The return value is a window handle if the function is successful. Otherwise, it is `NULL`, indicating that an error occurred.

See Also `WinOpenWindowDC`

■ WinWindowFromID

HWND WinWindowFromID (*hwndParent*, *id*)

HWND *hwndParent*; /* handle of the parent window */
USHORT *id*; /* window identifier */

The **WinWindowFromID** function returns the first child window of *hwndParent* that has the specified identifier.

Parameters

hwndParent Identifies the parent window.
id Identifies the window.

Return Value The return value is a window handle. If no child window exists with identifier *id* the return value is NULL.

Comments To obtain the window handle for an item within a dialog box, the *hwndParent* parameter is set to the dialog-box window's handle and the *id* parameter is set to the identifier of the item in the dialog template.

To obtain the window handle for a frame control, the *hwndParent* parameter is set to the frame window's handle and the *id* parameter is set to one of the FID constants, indicating which frame control you want a handle of. The following list contains the frame control identifiers:

Value	Meaning
FID_CLIENT	Identifies the client window.
FID_HORZSCROLL	Identifies the horizontal scroll bar.
FID_MENU	Identifies the system menu.
FID_MINMAX	Identifies the minimize/maximize box.
FID_SYSMENU	Identifies the system menu.
FID_TITLEBAR	Identifies the title bar.
FID_VERTSCROLL	Identifies the vertical scroll bar.

Example This example calls **WinWindowFromID** to get the window handle of the system menu and calls **WinSendMessage** to send a message to disable the Close menu item.

```
HWND hwndSysMenu;

hwndSysMenu = WinWindowFromID(hwndDlg, FID_SYSMENU);
WinSendMessage(hwndSysMenu, MM_SETITEMATTR,
    MPFROM2SHORT(SC_CLOSE, TRUE),
    MPFROM2SHORT(MIA_DISABLED, MIA_DISABLED));
```

See Also WinMultWindowFromIDs, WinWindowFromPoint

■ WinWindowFromPoint

HWND WinWindowFromPoint(*hwnd*, *pptl*, *fChildren*, *fLock*)

HWND *hwnd*; /* handle of the window */
PPOINTL *pptl*; /* address of structure with the point */
BOOL *fChildren*; /* scope flag */
BOOL *fLock*; /* lock/unlock flag */

The **WinWindowFromPoint** function finds the window that is below a specified point and that is a descendant of a specified window. This function checks only the descendants of the specified window.

Parameters

hwnd Identifies the window whose child windows are tested.

pptl Points to a **POINTL** structure that contains the point to test, specified in window coordinates relative to the *hwnd* parameter. The **POINTL** structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

fChildren Specifies which child windows to test. If *fChildren* is **TRUE**, the function tests all the descendants of *hwnd*, including child windows of child windows. If *fChildren* is **FALSE**, the function tests only the immediate child windows of *hwnd*.

fLock Specifies whether the window is to be locked or unlocked. If **TRUE**, the window is locked. If **FALSE**, it is not.

Return Value

If *fChildren* is **FALSE**, the return value is *hwnd*, a child of *hwnd*, or **NULL**. If *fChildren* is **TRUE**, the return value is the topmost window if that window is *hwnd* or a child of *hwnd*—unless another window of **CS_HITTEST** type is found, in which case the window returned may not be the topmost window.

Comments

If the retrieved window is locked by this function, you must at some point call the **WinLockWindow** function to unlock the window. The reason for locking the window is so that the window cannot be destroyed until you are done using it.

See Also

WinWindowFromID

■ WinWriteProfileData

BOOL WinWriteProfileData(*hab*, *pszAppName*, *pszKeyName*, *pchBinaryData*, *cchData*)

HAB *hab*; /* handle of the anchor block */
PSZ *pszAppName*; /* address of the application name */
PSZ *pszKeyName*; /* address of the keyname */
PVOID *pchBinaryData*; /* address of the data */
USHORT *cchData*; /* length of the data */

The **WinWriteProfileData** function places binary data into the *os2.ini* file. Where the data is placed is determined by an application name and a keyname that are passed to the function. The data can then be retrieved at a later time with the

WinQueryProfileData function, using the same application name (*pszAppName*) and keyname (*pszKeyName*).

Parameters

hab Identifies the anchor block.

pszAppName Points to a null-terminated text string that contains the name of the application. Its length must be less than 1024 bytes, including the null termination character. The application name is always case-dependent. If there is no application field in the *os2.ini* file that matches *pszAppName*, a new application field is created before the keyname entry is made for it.

pszKeyName Points to a null-terminated text string that contains the keyname. Its length must be less than 1024 bytes, including the null termination character. If *pszKeyName* is NULL, all keynames and their data are deleted. The keyname is always case-dependent. If there is no keyname that matches *pszKeyName*, a new keyname field is created. If the keyname already exists, the existing value is overwritten.

pchBinaryData Points to the binary data that is placed into the *os2.ini* file. There is no explicit termination character. If *pchBinaryData* is NULL, the previous value associated with *pszKeyName* is deleted; otherwise, the data string becomes the value even if it has a zero length. The data should not exceed 64K.

cchData Specifies the size of the *pchBinaryData* parameter, in bytes.

Return Value

The return value is TRUE if the function is successful. Otherwise, it is FALSE, indicating that an error occurred. If the *os2.ini* file exists but is in corrupted form, this function returns an error.

Comments

The application must know the size of the stored data when it calls **WinQueryProfileData** to retrieve the data.

See Also

WinQueryProfileData

■ WinWriteProfileString

BOOL WinWriteProfileString(*hab, pszAppName, pszKeyName, pszString*)

HAB *hab*; /* handle of the anchor block */
PSZ *pszAppName*; /* address of the application name */
PSZ *pszKeyName*; /* address of the keyname */
PSZ *pszString*; /* address of the string to write */

The **WinWriteProfileString** function places an ASCII string into the *os2.ini* file. Where the data is placed is determined by an application name and a keyname that are passed to the function. The data can then be retrieved at a later time with the **WinQueryProfileString** function, using the same application name (*pszAppName*) and keyname (*pszKeyName*).

Parameters

hab Identifies the anchor block.

pszAppName Points to a null-terminated text string that contains the name of the application. Its length must be less than 1024 bytes, including the null termination character. The application name is always case-dependent. If there is no application field in the *os2.ini* file that matches *pszAppName*, a new application field is created before the keyname entry is made for it.

pszKeyName Points to a null-terminated text string that contains the keyname. Its length must be less than 1024 bytes, including the null termination character. If *pszKeyName* is NULL, all keynames and their data are deleted. The keyname is always case-dependent. If there is no keyname that matches *pszKeyName*, a new keyname field is created. If the keyname already exists, the existing value is overwritten.

pszString Points to a null-terminated ASCII string that is placed into the *os2.ini* file. If *pszString* is NULL, the previous value associated with *pszKeyName* is deleted; otherwise, the ASCII string becomes the value even if it has a zero length. The string should not exceed 64K.

Return Value The return value is TRUE if the function is successful or FALSE if an error occurs.

See Also WinQueryProfileString



Chapter

3

Messages Directory

3.1	Introduction	393
3.2	Messages	394



3.1 Introduction

This chapter describes MS OS/2 window messages used for windows and predefined control windows. MS OS/2 window messages let you control the operation of the windows you create for your Presentation Manager applications. Most messages apply to windows belonging to any window class, including window classes you register privately. Some messages, however, apply to windows created with the MS OS/2 preregistered, control-window classes.

MS OS/2 window messages represent seven distinct message groups. As described in the following list, programs use these message groups to carry out specific tasks:

Message group	Usage
Buttons	Use the button messages (BM_) to set and query the state of button controls. Button controls represent push buttons, radio buttons, check boxes, and user buttons.
Entry fields	Use the entry-field messages (EM_) to set and retrieve text in entry-field controls. These messages also let you cut, copy, and paste text between the entry-field controls and the clipboard.
List boxes	Use the list-box messages (LM_) to set and retrieve lists in list-box controls. These messages also let you select or delete items in the list, query for the currently selected item or items, and search for items.
Menus	Use the menu messages (MM_) to set and retrieve items in menu controls. These messages let you add or delete items in a menu, retrieve information about menu items, and query for the currently selected item.
Scroll bars	Use the scroll-bar messages (SBM_) to set and retrieve the current position of the scroll-bar slider.
Title bar	Use the title-bar messages (TBM_) to set and retrieve the current text in a title-bar control.
General	Use the general window messages (WM_) to control the operation of windows of any window class. For most general window messages, the system sends the message to the window procedure of the given window. These messages can represent input from the keyboard, mouse, or timer. Some messages are requests from the system to the window procedure for information or they are actions to be taken. Other messages contain information that the window procedure can use or save to process later.

Message group	Usage
	The system uses general window messages when creating, destroying, moving, sizing, and activating windows. It also uses these messages for all input to the window, whether the input is from devices like the keyboard and mouse or through other windows, such as dialogs and menus.

This chapter gives complete syntax, purpose, and parameter descriptions for each message. Types, macros, and structures used by a message are given with the message; these are described more fully in Chapter 4, "Types, Macros, Structures." The numeric values for error values returned by the messages are listed in Appendix A, "Error Values."

Some of the message descriptions in this chapter include examples. The examples show how to use the messages to accomplish simple tasks. In nearly all cases, the examples are code fragments, not complete programs. A code fragment is intended to show the context in which a message can be used, but often assumes that variables, structures, and constants used in the example have been defined and/or initialized. Also, a code fragment may use comments to represent a task instead of giving the actual statements.

Although the examples are not complete, you can still use them in your programs if you take the following steps:

- Include the *os2.h* file in your program.
- Define the appropriate include constants for the functions, structures, and constants used in the example.
- Define and initialize all variables.
- Replace comments that represent tasks with appropriate statements.
- Check return values for errors and take appropriate action.

3.2 Messages

The following is a complete list, in alphabetical order, of the MS OS/2 window messages.

■ BM_CLICK

```
BM_CLICK
mp1 = 0L;    /* not used, must be zero */
mp2 = 0L;    /* not used, must be zero */
```

An application sends a `BM_CLICK` message to simulate the effect of the user clicking a mouse button. The button window responds to this message by taking the same action that would occur if the button was clicked by the user.

Parameters This message does not use any parameters.

Return Value This message does not return a value.

See Also `WM_BUTTON1DOWN`, `WM_BUTTON1UP`, `WM_QUERYDLGCODE`

■ BM_QUERYCHECK

```
BM_QUERYCHECK
mp1 = 0L;    /* not used, must be zero */
mp2 = 0L;    /* not used, must be zero */
```

An application sends a `BM_QUERYCHECK` message to determine the checked state of a button control.

Parameters This message does not use any parameters.

Return Value The return value from a button control created with the `BS_CHECKBOX`, `BS_AUTOCHECKBOX`, `BS_RADIOBUTTON`, `BS_AUTORADIOBUTTON`, `BS_3STATE`, or `BS_AUTO3STATE` style is one of the following values:

Value	Meaning
0	Button state is unchecked.
1	Button state is checked.
2	Button state is indeterminate.

If the button style is any other value, the return value is zero.

See Also `BM_QUERYHILITE`, `BM_SETCHECK`

■ BM_QUERYCHECKINDEX

```
BM_QUERYCHECKINDEX
mp1 = 0L;    /* not used, must be zero */
mp2 = 0L;    /* not used, must be zero */
```

An application sends the `BM_QUERYCHECKINDEX` message to determine the zero-based index of a checked radio button. It can be sent to any radio button or auto-radio button within a group. The button window responds to this message by returning the zero-based index of the checked radio button.

Parameters This message does not use any parameters.

Return Value The return value is the radio-button index if the operation is successful or - 1 if no radio button in the group is checked or if the button control does not have the style BS_RADIOBUTTON or BS_AUTORADIOBUTTON.

See Also BM_QUERYCHECK

■ BM_QUERYHILITE

```
BM_QUERYHILITE
mp1 = OL; /* not used, must be zero */
mp2 = OL; /* not used, must be zero */
```

An application sends a BM_QUERYHILITE message to determine the highlighting state of a button control.

Parameters This message does not use any parameters.

Return Value The return value is TRUE if the button is highlighted, or FALSE if the button is not highlighted or if the button was not created with the BS_PUSHBUTTON style.

See Also BM_QUERYCHECK, BM_SETHILITE

■ BM_SETCHECK

```
BM_SETCHECK
mp1 = MPFROMSHORT((USHORT) fCheck); /* check state */
mp2 = OL; /* not used, must be zero */
```

An application sends a BM_SETCHECK message to set the checked state of a button control.

Parameters *fCheck* Low word of *mp1*. Specifies the check state. This parameter can be one of the following values:

Value	Meaning
0	Set the button state to unchecked.
1	Set the button state to checked.
2	Set the button state to indeterminate. This value can be used only if the button has the BS_3STATE or BS_AUTO3STATE style.

Return Value The return value is the previous check state.

See Also BM_QUERYCHECK, BM_SETDEFAULT, BM_SETHILITE

■ BM_SETDEFAULT

```
BM_SETDEFAULT
mp1 = MPFROMSHORT((USHORT) fDefault); /* default state */
mp2 = OL; /* not used, must be zero */
```

An application sends a BM_SETDEFAULT message to set the default state of a

button control that has the BS_PUSHBUTTON or BS_USERBUTTON style. If the button does not have one of these styles, then the message has no effect.

- Parameters** *fDefault* Low word of *mp1*. Specifies the default state. A value of TRUE sets the default state; a value of FALSE removes the default state.
- Return Value** The return value is TRUE whether or not the default state is changed.
- See Also** BM_SETCHECK, BM_SETHILITE

■ BM_SETHILITE

```
BM_SETHILITE
mp1 = MPFROMSHORT((BOOL) fHighlight); /* highlight state */
mp2 = OL; /* not used, must be zero */
```

An application sends a BM_SETHILITE message to set the highlight state of a button control.

- Parameters** *fHighlight* Low word of *mp1*. Specifies the highlight state. A value of TRUE sets the highlighted state; a value of FALSE removes the highlighted state.
- Return Value** The return value is TRUE if the previous state was highlighted or FALSE if the previous state was not highlighted.
- See Also** BM_QUERYHILITE, BM_SETCHECK, BM_SETDEFAULT

■ EM_CLEAR

```
EM_CLEAR
mp1 = OL; /* not used, must be zero */
mp2 = OL; /* not used, must be zero */
```

An application sends an EM_CLEAR message to delete the current selection in an entry-field control.

- Parameters** This message does not use any parameters.
- Return Value** The return value is TRUE if the operation is successful or FALSE if an error occurs.
- See Also** EM_CUT

■ EM_COPY

```
EM_COPY
mp1 = OL; /* not used, must be zero */
mp2 = OL; /* not used, must be zero */
```

An application sends an EM_COPY message to paste the current selection in an entry-field control to the clipboard in CF_TEXT format. The current selection in the control is not changed.

Parameters	This message does not use any parameters.
Return Value	The return value is TRUE if the operation is successful or FALSE if an error occurs.
See Also	EM_CUT, EM_PASTE

■ EM_CUT

```
EM_CUT
mp1 = OL; /* not used, must be zero */
mp2 = OL; /* not used, must be zero */
```

An application sends an EM_CUT message to delete the current selection in an entry-field control and place the selection into the clipboard in CF_TEXT format.

Parameters	This message does not use any parameters.
Return Value	The return value is TRUE if the operation is successful or FALSE if an error occurs.
See Also	EM_COPY, EM_PASTE

■ EM_PASTE

```
EM_PASTE
mp1 = OL; /* not used, must be zero */
mp2 = OL; /* not used, must be zero */
```

An application sends an EM_PASTE message to replace the current selection in an entry-field control with the contents of the clipboard if the clipboard data is in CF_TEXT format.

Parameters	This message does not use any parameters.
Return Value	The return value is TRUE if the operation is successful or FALSE if an error occurs.
See Also	EM_COPY, EM_CUT

■ EM_QUERYCHANGED

```
EM_QUERYCHANGED
mp1 = OL; /* not used, must be zero */
mp2 = OL; /* not used, must be zero */
```

An application sends an EM_QUERYCHANGED message to determine if the contents of the entry-field control have changed since the last WM_QUERYWINDOWPARAMS or EM_QUERYCHANGED message.

- Parameters** This message does not use any parameters.
- Return Value** The return value is TRUE if the contents have changed since the last query or FALSE if the contents have not changed or if an error occurred.
- See Also** WM_QUERYWINDOWPARAMS

■ EM_QUERYFIRSTCHAR

```
EM_QUERYFIRSTCHAR
mp1 = 0L; /* not used, must be zero */
mp2 = 0L; /* not used, must be zero */
```

An application sends an EM_QUERYFIRSTCHAR message to determine the index of the character displayed in the first position of the edit control.

- Parameters** This message does not use any parameters.
- Return Value** The return value is the zero-based offset to the first character visible at the left edge of an entry-field control.
- See Also** EM_SETFIRSTCHAR

■ EM_QUERYSEL

```
EM_QUERYSEL
mp1 = 0L; /* not used, must be zero */
mp2 = 0L; /* not used, must be zero */
```

An application sends an EM_QUERYSEL message to determine the offsets of the current selection in an entry-field control.

- Parameters** This message does not use any parameters.
- Return Value** The low word of the return value is the byte offset to the first character of the selection. The high word of the return value is the byte offset of the last character of the selection.
- Comments** To determine the text for the current selection, an application would first call the WinQueryWindowText function to get the entire contents, then send a EM_QUERYSEL message to get the offsets to the first and last characters of the selection within the text, and then use these offsets to retrieve the selection text from the entire text.
- See Also** WinQueryWindowText, EM_SETSEL

■ EM_SETFIRSTCHAR

```
EM_SETFIRSTCHAR
mp1 = MPFROMSHORT((SHORT) lchOffset); /* offset to first character */
mp2 = 0L; /* not used, must be zero */
```

An application sends an EM_SETFIRSTCHAR message to set the character to

be displayed as the first character in an entry-field, scrolling the contents as necessary.

- Parameters** *ichOffset* Low word of *mp1*. Specifies the offset to the character to place at the left edge of the control.
- Return Value** The return value is TRUE if the operation is successful or FALSE if an error occurs. This message returns FALSE if the edit control does not have the ES_AUTOSCROLL style or if it is centered or right justified.
- See Also** EM_QUERYFIRSTCHAR

■ EM_SETSEL

```
EM_SETSEL
mp1 = MPFROM2SHORT((USHORT) usFirst, (USHORT) usLast); /* range */
mp2 = 0L; /* not used, must be zero */
```

An application sends an EM_SETSEL message to set the range of a selection in an entry field. If the first character position is zero and the last character position is greater than or equal to the number of characters in the entry field, the entire text is selected.

- Parameters** *usFirst* Low word of *mp1*. Specifies the offset to the first position of the selection.
- usLast* High word of *mp1*. Specifies the offset to the last position of the selection.
- Return Value** The return value is TRUE if the operation is successful or FALSE if an error occurs.
- See Also** EM_QUERYSEL, WM_QUERYDLGCODE

■ EM_SETTEXTLIMIT

```
EM_SETTEXTLIMIT
mp1 = MPFROMSHORT((SHORT) cchMax); /* max. number of characters */
mp2 = 0L; /* not used, must be zero */
```

An application sends an EM_SETTEXTLIMIT message to set the maximum number of characters an entry-field control can hold.

- Parameters** *cchMax* Low word of *mp1*. Specifies the maximum number of characters an entry field can hold.
- Return Value** The return value is TRUE if the operation is successful or FALSE if there is not enough memory to hold the requested number of characters.
- Comments** Sending an EM_SETTEXTLIMIT message causes memory to be allocated from the control heap for the specified maximum number of characters. Failure to allocate sufficient memory results in a WM_CONTROL message with the EM_MEMERROR notification code being sent to the owner window.
- See Also** WM_CONTROL

■ LM_DELETEALL

```
LM_DELETEALL
mp1 = OL; /* not used, must be zero */
mp2 = OL; /* not used, must be zero */
```

An application sends an LM_DELETEALL message to delete all items in a list-box control.

- Parameters** This message does not use any parameters.
- Return Value** The return value is TRUE if the operation is successful or FALSE if an error occurs.
- See Also** LM_DELETEITEM

■ LM_DELETEITEM

```
LM_DELETEITEM
mp1 = MPFROMSHORT((SHORT) iItem); /* item to be deleted */
mp2 = OL; /* not used, must be zero */
```

An application sends an LM_DELETEITEM message to delete an item from a list-box control.

- Parameters** *iItem* Low word of *mp1*. Specifies the index of the item.
- Return Value** The return value is the number of items remaining in the list.
- See Also** LM_DELETEALL, LM_INSERTITEM

■ LM_INSERTITEM

```
LM_INSERTITEM
mp1 = MPFROMSHORT((SHORT) iItem); /* item index */
mp2 = MPFROMP((PSZ) pszText); /* pointer to text to insert */
```

An application sends an LM_INSERTITEM message to insert an item into a list-box control. The actual placement of the item is determined by the *iItem* parameter.

- Parameters** *iItem* Low word of *mp1*. Specifies the index of the item. This parameter can be a zero-based index or one of the following values:

Value	Meaning
LIT_END	The item is to be added to the end of the list.
LIT_SORTASCENDING	The item is to be added to the list sorted in ascending order.
LIT_SORTDESCENDING	The item is to be added to the list sorted in descending order.

pszText Low and high word of *mp2*. Points to the text to insert.

Return Value The return value is the actual position of the item if it was successfully inserted. The return value is LIT_MEMERROR if the list-box control cannot allocate space to insert the item in the specified position. Otherwise, the return value is LIT_ERROR, indicating an error occurred.

See Also LM_DELETEITEM

■ LM_QUERYITEMCOUNT

```
LM_QUERYITEMCOUNT
mp1 = OL;      /* not used, must be zero */
mp2 = OL;      /* not used, must be zero */
```

An application sends an LM_QUERYITEMCOUNT message to determine the number of items in a list-box control.

Parameters This message does not use any parameters.

Return Value The return value is the item count.

■ LM_QUERYITEMHANDLE

```
LM_QUERYITEMHANDLE
mp1 = MPFROMSHORT((SHORT) iItem); /* item index */
mp2 = OL; /* not used, must be zero */
```

An application sends an LM_QUERYITEMHANDLE message to get the handle of the specified item in a list box.

Parameters *iItem* Low word of *mp1*. Specifies the index of the item.

Return Value The return value is the item handle if the operation is successful, or zero if the item does not exist or an error occurred.

See Also LM_SETITEMHANDLE

■ LM_QUERYITEMTEXT

```
LM_QUERYITEMTEXT
mp1 = MPFROM2SHORT((SHORT) iItem, (SHORT) cch); /* item-buffer size */
mp2 = MPFROMP((FAR *) pszText); /* buffer for text */
```

An application sends an LM_QUERYITEMTEXT message to copy the text for a specified list-box item into a buffer provided by the caller. The size of the buffer can be determined by sending an LM_QUERYITEMTEXTLENGTH message for the item.

Parameters	<i>iItem</i> Low word of <i>mp1</i> . Specifies the index of the item. <i>cch</i> High word of <i>mp1</i> . Specifies the maximum number of characters to get. <i>pszText</i> Low and high word of <i>mp2</i> . Points to the buffer to receive the item's text.
Return Value	The return value is the length of the text string copied, not including the null termination character.
See Also	LM_QUERYITEMTEXTLENGTH, LM_SETITEMTEXT, WM_DRAWITEM

■ LM_QUERYITEMTEXTLENGTH

```
LM_QUERYITEMTEXTLENGTH
mp1 = MPFROMSHORT((SHORT) iItem);    /* item index */
mp2 = 0L;                             /* not used, must be zero */
```

An application sends an LM_QUERYITEMTEXTLENGTH message to determine the length of the text in the specified list-box item.

Parameters	<i>iItem</i> Low word of <i>mp1</i> . Specifies the index of the item.
Return Value	The return value is the length (in characters) of the text of the item specified by <i>iItem</i> if the operation is successful, or zero if the specified item does not exist or an error occurred.
See Also	LM_QUERYITEMTEXT

■ LM_QUERYSELECTION

```
LM_QUERYSELECTION
mp1 = MPFROMSHORT((SHORT) iItemPrev); /* previous item */
mp2 = 0L;                             /* not used, must be zero */
```

An application sends an LM_QUERYSELECTION message to enumerate the selected item or items in a list box.

Parameters	<i>iItemPrev</i> Low word of <i>mp1</i> . Specifies the index of the previous item. A value of LIT_FIRST, when used with a multiple-selection list-box control, results in the first selected item being returned.
Return Value	The return value from a single-selection list-box control is the index of the selected item, or LIT_NONE if no item is selected. The return value from a multiple-selection list-box control is the index of the next selected item (starting from the item specified by the <i>iItemPrev</i> parameter) or LIT_NONE if there are no more selected items. The return value is the index of the first selected item if <i>iItemPrev</i> is LIT_FIRST.
See Also	LM_SELECTITEM

■ LM_QUERYTOPINDEX

```
LM_QUERYTOPINDEX
mp1 = OL; /* not used, must be zero */
mp2 = OL; /* not used, must be zero */
```

An application sends an LM_QUERYTOPINDEX to determine the index of the item currently at the top of the list box.

Parameters This message does not use any parameters.

Return Value The return value is the index of the item currently displayed at the top of the list-box window, or LIT_NONE if the list is empty.

See Also LM_SETTOPINDEX

■ LM_SEARCHSTRING

```
LM_SEARCHSTRING
mp1 = MPFROM2SHORT((USHORT) usCmd, (SHORT) iItem); /* cmd and item */
mp2 = MPFROMP((PSZ) pszSearch); /* search string */
```

An application sends an LM_SEARCHSTRING message to search the list for a match to the specified string, returning the first matching item. Match criteria can be set with flags for case-sensitivity and substring matching. All items are searched until a match is found. Searching wraps around at the end of the list, starting again at the first item.

Parameters *usCmd* Low word of *mp1*. Specifies one of the following values that determines how to find a match (these values can be combined by using the OR operator):

Value	Meaning
LSS_CASESENSITIVE	Matching occurs if the item contains the string exactly, as specified by the string in the message.
LSS_PREFIX	Matching occurs if the leading characters of the item match the string specified in the message. If this value is specified, LSS_SUBSTRING should not be specified.
LSS_SUBSTRING	Matching occurs if the item contains a substring that matches the string specified in the message. If this value is specified, LSS_PREFIX should not be specified.

iItem High word of *mp1*. Indicates the index of the item to begin searching. A value of LIT_FIRST causes searching to begin with the first item.

pszSearch Low and high word of *mp2*. Points to the search string.

Return Value The return value is the item index of the next item whose text string matches the string specified by the *pszSearch* parameter, LIT_NONE if no item is found, or LIT_ERROR if an error occurs.

■ LM_SELECTITEM

```
LM_SELECTITEM
mp1 = MPFROMSHORT((SHORT) iItem);    /* item index */
mp2 = MPFROMSHORT((BOOL) fSelect);   /* selection flag */
```

An application sends an LM_SELECTITEM message to set the selection state of an item in a list-box control. If the control is a single-selection list box, the previous selected item is deselected.

- Parameters** *iItem* Low word of *mp1*. Specifies the index of the item to select or deselect.
 fSelect Low word of *mp2*. Indicates if the item should be selected or deselected. A value of TRUE selects the item; a value of FALSE deselects the item.
- Return Value** The return value is TRUE if the operation is successful or FALSE if an error occurs.
- See Also** LM_QUERYSELECTION

■ LM_SETITEMHANDLE

```
LM_SETITEMHANDLE
mp1 = MPFROMSHORT((SHORT) iItem);    /* item index */
mp2 = MPFROMLONG((ULONG) ulHandle);  /* item handle */
```

An application sends an LM_SETITEMHANDLE message to set the handle of an item in a list-box control.

- Parameters** *iItem* Low word of *mp1*. Specifies the index of the item.
ulHandle Low and high word of *mp2*. Specifies the handle of the item.
- Return Value** The return value is TRUE if the specified item exists; otherwise, it is FALSE.
- See Also** LM_QUERYITEMHANDLE

■ LM_SETITEMHEIGHT

```
LM_SETITEMHEIGHT
mp1 = MPFROMSHORT((SHORT) sHeight);  /* item height */
mp2 = 0L;                             /* not used, must be zero */
```

The list-box control responds to an LM_SETITEMHEIGHT message from an application by setting the height of the items in a list box to the height specified by the *sHeight* parameter.

- Parameters** *sHeight* Low word of *mp1*. Specifies the height of each item in the list box.
- Return Value** The return value is TRUE if the operation is successful or FALSE if an error occurs.
- See Also** WM_MEASUREITEM

■ LM_SETITEMTEXT

```
LM_SETITEMTEXT
mp1 = MPFROMSHORT((SHORT) iItem);    /* item index */
mp2 = MPFROM(PSZ) pszText);          /* pointer to text to copy */
```

An application sends an LM_SETITEMTEXT message to copy text from a specified buffer to an item in a list box.

- Parameters** *iItem* Low word of *mp1*. Specifies the index of the item.
 pszText Low and high word of *mp2*. Points to the buffer that contains the text to copy to the item specified by the *iItem* parameter.
- Return Value** The return value is TRUE if the operation is successful or FALSE if an error occurs.
- See Also** LM_QUERYITEMTEXT

■ LM_SETTOPINDEX

```
LM_SETTOPINDEX
mp1 = MPFROMSHORT((SHORT) iItem);    /* item index */
mp2 = OL;                             /* not used, must be zero */
```

An application sends an LM_SETTOPINDEX message to scroll an item to the top of a list box.

- Parameters** *iItem* Low word of *mp1*. Specifies the index of the item to place at the top of the list box.
- Return Value** The return value is TRUE if the operation is successful or FALSE if an error occurs.
- See Also** LM_QUERYTOPINDEX

■ MM_DELETEITEM

```
MM_DELETEITEM
mp1 = MPFROM2SHORT((USHORT) idItem, (BOOL) fIncludeSubMenus);
mp2 = OL;    /* not used, must be zero */
```

An application sends an MM_DELETEITEM message to delete a menu item.

- Parameters** *idItem* Low word of *mp1*. Identifies the item to delete.
 fIncludeSubMenus High word of *mp1*. Specifies whether to include submenus in the search for an item matching the *idItem* parameter. If TRUE, the search includes all child menus. If FALSE, no child menus are searched.
- Return Value** The return value is the count of remaining menu items.
- See Also** MM_INSERTITEM, MM_REMOVEITEM

■ MM_ENDMENU MODE

```
MM_ENDMENU MODE
mp1 = MPFROMSHORT((BOOL) fDismiss); /* dismiss flag */
mp2 = 0L; /* not used, must be zero */
```

An application sends an MM_ENDMENU MODE message to terminate menu selection. If the *fDismiss* parameter is TRUE and a submenu is visible, that window is dismissed (hidden).

Parameters *fDismiss* Low word of *mp1*. Specifies whether a submenu window is to be dismissed. A value of TRUE dismisses the submenu window.

Return Value This message does not return a value.

See Also MM_STARTMENU MODE

■ MM_INSERTITEM

```
MM_INSERTITEM
mp1 = (MPARAM) pmi; /* pointer to MENUITEM structure */
mp2 = MPFROMP((PSZ) pszText); /* pointer to text */
```

An application sends an MM_INSERTITEM message to insert a menu item. The item pointed to by the *pmi* parameter is inserted into the menu list at the position specified by the item index (contained within the MENUITEM structure). If the item index is MIT_END, the item is added to the end of the list. If the style of the item includes MIS_TEXT, the text of the item is pointed to by the *pszText* parameter.

Parameters *pmi* Low and high word of *mp1*. Points to a MENUITEM structure. The MENUITEM structure has the following form:

```
typedef struct _MENUITEM {
    SHORT iPosition;
    USHORT afStyle;
    USHORT afAttribute;
    USHORT id;
    HWND hwndSubMenu;
    ULONG hItem;
} MENUITEM;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

pszText Low and high word of *mp2*. Points to the text for the menu item.

Return Value The return value is the actual position of the item if the item was inserted successfully. The return value is MIT_MEMERROR if the control cannot allocate space to insert the item in the menu. The return value is MIT_ERROR if the *iPosition* field of the MENUITEM structure is invalid.

See Also MENUITEM, MM_DELETEITEM

■ MM_ISITEMVALID

```
MM_ISITEMVALID
mp1 = MPFROM2SHORT((USHORT) idItem, (BOOL) fIncludeSubMenus);
mp2 = MPFROMSHORT((BOOL) fValidIfNotFound);
```

An application sends an MM_ISITEMVALID message to determine if a menu item can be selected. The menu control sends a WM_INITMENU message before checking the state of the menu item and a WM_MENUEND message after checking the state.

- Parameters**
- idItem* Low word of *mp1*. Identifies the menu item.
 - fIncludeSubMenus* High word of *mp1*. Specifies whether to include submenus in the search for an item that matches the *idItem* parameter.
 - fValidIfNotFound* Low word of *mp2*. Specifies the value to return if the item is not found.
- Return Value** The return value is TRUE if the specified menu item can be selected or chosen, or FALSE if the item cannot be selected or does not exist.
- See Also** WM_INITMENU, WM_MENUEND

■ MM_ITEMIDFROMPOSITION

```
MM_ITEMIDFROMPOSITION
mp1 = MPFROMSHORT((SHORT) iItem); /* item index */
mp2 = 0L; /* not used, must be zero */
```

The menu control responds to an MM_ITEMIDFROMPOSITION message by returning the identity of the item whose position is specified by the *iItem* parameter, or MIT_ERROR if *iItem* is invalid.

- Parameters**
- iItem* Low word of *mp1*. Specifies the index of the item in the menu list.
- Return Value** The return value is the identity of the item whose position is specified by *iItem*, or it is MID_ERROR if *iItem* is invalid.
- See Also** MM_ITEMPOSITIONFROMID

■ MM_ITEMPOSITIONFROMID

```
MM_ITEMPOSITIONFROMID
mp1 = MPFROM2SHORT((USHORT) idItem, (BOOL) fIncludeSubMenus);
mp2 = 0L; /* not used, must be zero */
```

An application sends an MM_ITEMPOSITIONFROMID message to determine the position of a menu item in the menu list.

- Parameters**
- idItem* Low word of *mp1*. Identifies the menu item.
 - fIncludeSubMenus* High word of *mp1*. Specifies whether to include submenus in the search for an item that matches the *idItem* parameter.

Return Value The return value is the zero-based index of the item specified in *idItem*, or it is `MIT_NONE` if the item does not exist.

See Also `MM_ITEMIDFROMPOSITION`

■ MM_QUERYITEM

```
MM_QUERYITEM
mp1 = MPFROM2SHORT((USHORT) idItem, (BOOL) fIncludeSubMenus);
mp2 = (MPARAM) pmi; /* pointer to MENUITEM structure */
```

An application sends an `MM_QUERYITEM` message to copy information about the item to a `MENUITEM` structure. This message does not retrieve the text for items that have the style `MIS_TEXT`. The application must use the `MM_QUERYITEMTEXT` message to retrieve these items.

Parameters *idItem* Low word of *mp1*. Identifies the menu item.

fIncludeSubMenus High word of *mp1*. Specifies whether to include submenus in the search for an item that matches the *idItem* parameter.

pmi Low and high word of *mp2*. Points to a `MENUITEM` structure. The `MENUITEM` structure has the following form:

```
typedef struct _MENUITEM {
    SHORT    iPosition;
    USHORT   afStyle;
    USHORT   afAttribute;
    USHORT   id;
    HWND     hwndSubMenu;
    ULONG    hItem;
} MENUITEM;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value The return value is `TRUE` if the operation is successful or `FALSE` if an error occurs.

See Also `MENUITEM`, `MM_QUERYITEMTEXT`, `MM_SETITEM`

■ MM_QUERYITEMATTR

```
MM_QUERYITEMATTR
mp1 = MPFROM2SHORT((USHORT) idItem, (BOOL) fIncludeSubMenus);
mp2 = MPFROMSHORT((USHORT) rgfAttributeMask);
```

An application sends an `MM_QUERYITEMATTR` message to determine the state of the specified attributes of a menu item.

Parameters *idItem* Low word of *mp1*. Identifies the menu item.

fIncludeSubMenus High word of *mp1*. Specifies whether to include submenus in the search for an item that matches the *idItem* parameter.

rgfAttributeMask Low word of *mp2*. Specifies the attributes to get the state of. This parameter can be any combination of the following values:

Value	Meaning
MIA_CHECKED	A check mark appears to the left of the menu item.
MIA_DISABLED	The menu item is disabled.
MIA_FRAMED	The menu item is framed by vertical lines to the left and right.
MIA_HILITED	The menu item is selected (highlighted).
MIA_NODISMISS	Indicates whether the menu is self dismissing.

Return Value The return value is the state of the attributes specified in the *rgfAttributeMask* parameter for the menu item specified by the *idItem* parameter.

Example This example sends an MM_QUERYITEMATTR message to find the state of the *idCase* menu item. It then toggles the state of the item and sends an MM_SETITEMATTR message to set the new state.

```
sState = (SHORT) WinSendMessage(hwndMenu, MM_QUERYITEMATTR,
    MPFROM2SHORT(idCase, TRUE), MPFROMSHORT(MIA_CHECKED));
sState ^= MIA_CHECKED;
WinSendMessage(hwndMenu, MM_SETITEMATTR, MPFROM2SHORT(idCase, TRUE),
    MPFROM2SHORT(MIA_CHECKED, sState));
```

See Also MM_SETITEMATTR

■ MM_QUERYITEMCOUNT

```
MM_QUERYITEMCOUNT
mp1 = 0L; /* not used, must be zero */
mp2 = 0L; /* not used, must be zero */
```

An application sends an MM_QUERYITEMCOUNT message to determine the number of items in a menu. Submenus count as a single item if the message is sent to the action-bar menu window. To count the items in a submenu, the message must be sent to the submenu window.

Parameters This message does not use any parameters.

Return Value The return value is the number of items in the menu.

■ MM_QUERYITEMTEXT

```
MM_QUERYITEMTEXT
mp1 = MPFROM2SHORT((USHORT) idItem, (SHORT) cchMax);
mp2 = MPFROMP((PSZ) pszText);
```

An application sends an MM_QUERYITEMTEXT message to retrieve the text of a menu item. The menu item must have the style MIS_TEXT.

Parameters *idItem* Low word of *mp1*. Identifies the menu item with the text.

cchMax High word of *mp1*. Specifies the maximum number of characters to copy from the menu item to the supplied buffer.

pszText Low and high word of *mp2*. Points to the buffer that receives the text from the menu item.

- Return Value** The return value is the length of the string copied. If no text is copied, the return value is zero (this can result from errors such as an invalid item identifier or an item with no text).
- Comments** An application can determine the size of the text ahead of time by sending an MM_QUERYITEMTEXTLENGTH message.
- See Also** MM_QUERYITEMTEXTLENGTH, MM_SETITEMTEXT

■ MM_QUERYITEMTEXTLENGTH

```
MM_QUERYITEMTEXTLENGTH
mp1 = MPFROMSHORT((USHORT) idItem);    /* item identifier */
mp2 = OL;                               /* not used, must be zero */
```

An application sends an MM_QUERYITEMTEXTLENGTH message to determine the length of a menu item that has the style MIS_TEXT.

- Parameters** *idItem* Low word of *mp1*. Identifies the item.
- Return Value** The return value is the length (in characters) of the text of the item specified by the *idItem* parameter, or zero if an error occurs.
- See Also** MM_QUERYITEMTEXT

■ MM_QUERYSELITEMID

```
MM_QUERYSELITEMID
mp1 = OL;    /* not used, must be zero */
mp2 = OL;    /* not used, must be zero */
```

An application sends an MM_QUERYSELITEMID message to determine the identifier of the selected menu item.

- Parameters** This message does not use any parameters.
- Return Value** The return value is the selected item identifier, MIT_NONE if no item is selected, or MID_ERROR if an error occurs.
- See Also** MM_SELECTITEM

■ MM_REMOVEITEM

```
MM_REMOVEITEM
mp1 = MPFROM2SHORT((USHORT) idItem, (BOOL) fIncludeSubMenus);
mp2 = OL;    /* not used, must be zero */
```

An application sends an MM_REMOVEITEM message to remove a menu item.

- Parameters** *idItem* Low word of *mp1*. Identifies the item to remove.
- fIncludeSubMenus* High word of *mp1*. Specifies whether to include submenus in the search for an item that matches the *idItem* parameter.

- Return Value** The return value is the count of remaining menu items.
- Comments** Unlike the MM_DELETEITEM message, the MM_REMOVEITEM message removes but does not destroy the menu item. For example, after the MM_REMOVEITEM message is sent, the menu item could be inserted into another menu.
- See Also** MM_DELETEITEM

■ MM_SELECTITEM

```
MM_SELECTITEM
mp1 = MPFROM2SHORT((USHORT) idItem, (BOOL) fIncludeSubMenus);
mp2 = MPFROMSHORT((BOOL) fDismiss);
```

An application sends an MM_SELECTITEM message to select or dismiss a menu item. If an item is selected and the *fDismiss* parameter is TRUE, a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message is posted to the owner and the menu is dismissed.

- Parameters**
- idItem* Low word of *mp1*. Identifies the item. If a MID_NONE value is used, the selection is set to none.
- fIncludeSubMenus* High word of *mp1*. Specifies whether to include submenus in the search for an item that matches the *idItem* parameter.
- fDismiss* Low word of *mp2*. Specifies whether the menu is to be dismissed (hidden). A value of TRUE posts a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message before dismissing the item.
- Return Value** The return value is TRUE if the operation is successful or FALSE if an error occurs.
- See Also** MM_QUERYSELEITEMID, WM_COMMAND, WM_HELP, WM_SYSCOMMAND

■ MM_SETITEM

```
MM_SETITEM
mp1 = MPFROM2SHORT(0, (BOOL) fIncludeSubMenus);
mp2 = (MPARAM) pmi; /* pointer to MENUITEM structure */
```

An application sends an MM_SETITEM message to set a menu item. The menu control responds to this message by copying the item definition in the structure pointed to by the *pmi* parameter to the menu item with the same identifier.

If the *fIncludeSubMenus* parameter is TRUE and the menu does not have an item with the specified identifier, the submenus of this menu are searched for an item with a matching identifier. If one is found, the definition is copied to it.

- Parameters**
- fIncludeSubMenus* High word of *mp1*. Specifies whether to include submenus in the search for an item that matches the id field of the MENUITEM structure.
- pmi* Low and high word of *mp2*. Points to a MENUITEM structure. The MENUITEM structure has the following form:

```
typedef struct _MENUITEM {
    SHORT iPosition;
    USHORT afStyle;
    USHORT afAttribute;
    USHORT id;
    HWND hwndSubMenu;
    ULONG hItem;
} MENUITEM;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

- Return Value** The return value is TRUE if the operation is successful or FALSE if an error occurs.
- Comments** The `iPosition` field of the `MENUITEM` structure is ignored. The low word of `mp1` is not used and must be set to zero.
- See Also** `MM_QUERYITEM`, `MM_SETITEMATTR`, `MM_SETITEMHANDLE`, `MM_SETITEMTEXT`

■ MM_SETITEMATTR

```
MM_SETITEMATTR
mp1 = MPFROM2SHORT((USHORT) idItem, (BOOL) fIncludeSubMenus);
mp2 = MPFROM2SHORT((USHORT) rgfMask, (USHORT) rgfData);
```

An application sends an `MM_SETITEMATTR` message to change the state of a menu item’s attributes.

- Parameters**
- idItem* Low word of *mp1*. Identifies the item.
- fIncludeSubMenus* High word of *mp1*. Specifies whether to include submenus in the search for an item that matches the *idItem* parameter.
- rgfMask* Low word of *mp2*. Specifies a mask of the attributes to set. This parameter can be any combination of the following values:

Value	Meaning
<code>MIA_CHECKED</code>	A check mark appears to the left of the menu item.
<code>MIA_DISABLED</code>	The menu item is disabled.
<code>MIA_FRAMED</code>	The menu item is framed by vertical lines to the left and right.
<code>MIA_HILITED</code>	The menu item is selected (highlighted).

rgfData High word of *mp2*. Specifies the new state of the menu item’s attributes.

- Comments** More than one attribute can be set with a single message by combining the attributes to set in the *rgfMask* parameter and their new values in *rgfData*.

- Example** The following example sends an `MM_SETITEMATTR` message to set the `IDM_LARGE` menu item’s state to checked, and then sends another `MM_SETITEMATTR` message to set the `IDM_MEDIUM` menu item’s state to unchecked.

```
WinSendMessage(hwndActionBar, MM_SETITEMATTR,
    MPFROM2SHORT(IDM_LARGE, TRUE),
    MPFROM2SHORT(MIA_CHECKED, MIA_CHECKED));
```

```
WinSendMsg(hwndActionBar, MM_SETITEMATTR,
            MPFROM2SHORT(IDM_MEDIUM, TRUE),
            MPFROM2SHORT(MIA_CHECKED, FALSE));
```

Return Value The return value is TRUE if the operation is successful or FALSE if an error occurs.

See Also MM_QUERYITEMATTR, MM_SETITEM

■ MM_SETITEMHANDLE

```
MM_SETITEMHANDLE
mp1 = MPFROMSHORT((SHORT) idItem);    /* item index */
mp2 = MPFROMLONG((ULONG) ulHandle);  /* item handle */
```

An application sends an MM_SETITEMHANDLE message to set the handle of a menu item. It is used to set the display object, such as a bitmap, for menu items that do not have the style MIS_TEXT.

Parameters *idItem* Low word of *mp1*. Specifies the index of the item.

ulHandle Low and high word of *mp2*. Specifies the handle of the item.

Return Value The return value is TRUE if the operation is successful or FALSE if an error occurs.

■ MM_SETITEMTEXT

```
MM_SETITEMTEXT
mp1 = MPFROMSHORT((SHORT) idItem);    /* item index */
mp2 = MPFROMP((PSZ) pszText);        /* pointer to the text to copy */
```

An application sends an MM_SETITEMTEXT message to copy text from a specified buffer to a menu item that has the style MIS_TEXT.

Parameters *idItem* Low word of *mp1*. Specifies the menu item.

pszText Low and high word of *mp2*. Points to the buffer that contains the text to copy to the menu item specified by the *idItem* parameter.

Return Value The return value is TRUE if the operation is successful or FALSE if an error occurs.

■ MM_STARTMENU MODE

```
MM_STARTMENU MODE
mp1 = MPFROM2SHORT((BOOL) fShowSubmenu, (BOOL) fResumeMenu);
mp2 = 0L;    /* not used, must be zero */
```

An application posts an MM_STARTMENU MODE message to begin menu selection.

Parameters *fShowSubmenu* Low word of *mp1*. Specifies whether to show the submenu of the selected menu. A value of TRUE shows the submenu. This parameter is ignored if the selected menu does not have a submenu.

fResumeMenu High word of *mp1*. Specifies whether the menu is resumed. A value of TRUE causes the menu interaction to resume.

- Return Value** The return value is TRUE if the operation is successful or FALSE if an error occurs.
- Comments** The MM_STARTMENU MODE message must be posted, not sent.
- See Also** MM_ENDMENU MODE

■ SBM_QUERYPOS

```
SBM_QUERYPOS
mp1 = OL; /* not used, must be zero */
mp2 = OL; /* not used, must be zero */
```

An application sends an SBM_QUERYPOS message to get the current position of the slider in a scroll-bar window.

- Parameters** This message does not use any parameters.
- Return Value** The return value is the current position of the slider.
- See Also** SBM_SETPOS, SBM_SETSCROLLBAR

■ SBM_QUERYRANGE

```
SBM_QUERYRANGE
mp1 = OL; /* not used, must be zero */
mp2 = OL; /* not used, must be zero */
```

An application sends an SBM_QUERYRANGE message to get the minimum and maximum values of a scroll bar.

- Parameters** This message does not use any parameters.
- Return Value** The return value is the scroll-bar range. The low word contains the minimum value; high word contains the maximum value.
- See Also** SBM_SETPOS, SBM_SETSCROLLBAR

■ SBM_SETPOS

```
SBM_SETPOS
mp1 = MPFROMSHORT((USHORT) usPos); /* slider position */
mp2 = OL; /* not used, must be zero */
```

An application sends an SBM_SETPOS message to set the slider position in a scroll-bar window. If the position specified is outside the valid range of slider positions, the slider is moved to the nearest valid position.

- Parameters** *usPos* Low word of *mp1*. Specifies the slider position.

Return Value The return value is always TRUE.

See Also SBM_QUERYPOS, SBM_SETSCROLLBAR

■ SBM_SETSCROLLBAR

```
SBM_SETSCROLLBAR
mp1 = MPFROMSHORT((USHORT) usPos);          /* position */
mp2 = MPFROM2SHORT((USHORT) usFirst, (USHORT) usLast); /* range */
```

An application sends an SBM_SETSCROLLBAR message to set the range of a scroll-bar window and the position of the slider within that scroll bar.

Parameters *usPos* Low word of *mp1*. Specifies the slider position.

usFirst Low word of *mp2*. Specifies the first possible position of the slider.

usLast High word of *mp2*. Specifies the last possible position of the slider.

Return Value The return value is always TRUE.

See Also SBM_QUERYPOS, SBM_QUERYRANGE, SBM_SETPOS

■ SM_QUERYHANDLE

```
SM_QUERYHANDLE
mp1 = OL; /* not used, must be zero */
mp2 = OL; /* not used, must be zero */
```

An application sends an SM_QUERYHANDLE message to get the handle to the icon or the bitmap handle of a static control.

Parameters This message does not use any parameters.

Return Value The return value is the handle to the display object of the static control.

See Also SM_SETHANDLE

■ SM_SETHANDLE

```
SM_SETHANDLE
mp1 = MPFROMLONG((ULONG) hObj); /* handle of object */
mp2 = OL; /* not used, must be zero */
```

An application sends an SM_SETHANDLE message to set the icon or the bit-map handle for a static control.

Parameters *hObj* Low and high word of *mp1*. Identifies the object handle.

Return Value The return value is the handle passed in the *hObj* parameter.

See Also SM_QUERYHANDLE

■ TBM_QUERYHILITE

```
TBM_QUERYHILITE
mp1 = OL; /* not used, must be zero */
mp2 = OL; /* not used, must be zero */
```

An application sends a TBM_QUERYHILITE message to get the highlight state of a title-bar control.

Parameters	This message does not use any parameters.
Return Value	The return value is TRUE if the title bar is highlighted or FALSE if it is not.
See Also	TBM_SETHILITE

■ TBM_SETHILITE

```
TBM_SETHILITE
mp1 = MPFROMSHORT((BOOL) fHighlight); /* highlight flag */
mp2 = OL; /* not used */
```

An application sends a TBM_SETHILITE message to set the highlight state of the title-bar control.

Parameters	<i>fHighlight</i> Low word of <i>mp1</i> . Specifies whether to highlight or remove highlighting from the title bar. A value of TRUE highlights the title bar; a value of FALSE removes the highlighting.
Return Value	The return value is always TRUE.
See Also	TBM_QUERYHILITE

■ WM_ACTIVATE

```
WM_ACTIVATE
fActive = (BOOL) SHORT1FROMMP(mp1); /* activation/deactiv. flag */
hwnd = (HWND) HWNDFROMMP(mp2); /* window handle */
```

A WM_ACTIVATE message is sent when a window is being activated or deactivated. This message is sent first to the window procedure of the main window being deactivated and then to the window procedure of the main window being activated.

Parameters	<i>fActive</i> Low word of <i>mp1</i> . Indicates whether the window is being activated or deactivated. A value of TRUE means the window is being activated. A value of FALSE indicates the window is being deactivated. <i>hwnd</i> Low and high word of <i>mp2</i> . Identifies the window being activated or deactivated.
Return Value	An application should return zero if it processes this message.
Comments	When a window gains the focus, it receives a WM_ACTIVATE message, a WM_SETSELECTION message, and a WM_SETFOCUS message (in that

order). When the window loses the focus, it receives a WM_SETFOCUS message, a WM_SETSELECTION message, and a WM_ACTIVATE message (in that order).

See Also WM_FOCUSCHANGE, WM_SETFOCUS, WM_SETSELECTION

■ WM_ADJUSTWINDOWPOS

```
WM_ADJUSTWINDOWPOS
pswp = (PSWP) PVOIDFROMMP(mp1); /* pointer to SWP structure */
```

The WM_ADJUSTWINDOWPOS message is sent when a window is about to be moved or sized. It gives the window an opportunity to adjust the new size and position before the window is actually moved and sized.

Parameters *pswp* Low and high word of *mp1*. Points to an SWP structure that contains the new window size and position information. The SWP structure has the following form:

```
typedef struct _SWP {
    USHORT fs;
    SHORT cy;
    SHORT cx;
    SHORT y;
    SHORT x;
    HWND hwndInsertBehind;
    HWND hwnd;
} SWP;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value An application should return TRUE if it changes the SWP structure or FALSE if it does not.

See Also WM_CALCVALIDRECTS

■ WM_BUTTON1DBLCLK

```
WM_BUTTON1DBLCLK
x = (SHORT) SHORT1FROMMP(mp1); /* horizontal position */
y = (SHORT) SHORT2FROMMP(mp1); /* vertical position */
```

The WM_BUTTON1DBLCLK message is sent when the user double-clicks the left mouse button.

Parameters *x* Low word of *mp1*. Indicates the horizontal position of the mouse pointer (in window coordinates).

y High word of *mp1*. Indicates the vertical position of the mouse pointer (in window coordinates).

Return Value An application should return TRUE if it processes this message.

See Also WM_BUTTON1DOWN, WM_BUTTON1UP, WM_HITTEST

■ WM_BUTTON2DBLCLK

```
WM_BUTTON2DBLCLK
x = (SHORT) SHORT1FROMMP (mp1);    /* horizontal position */
y = (SHORT) SHORT2FROMMP (mp1);    /* vertical position  */
```

The WM_BUTTON2DBLCLK message is sent when the user double-clicks the second mouse button. On a two-button mouse, the second button is the right button.

- Parameters**
- x* Low word of *mp1*. Indicates the horizontal position of the mouse pointer (in window coordinates).
 - y* High word of *mp1*. Indicates the vertical position of the mouse pointer (in window coordinates).
- Return Value** An application should return TRUE if it processes this message.
- See Also** WM_BUTTON2DOWN, WM_BUTTON2UP, WM_HITTEST

■ WM_BUTTON3DBLCLK

```
WM_BUTTON3DBLCLK
x = (SHORT) SHORT1FROMMP (mp1);    /* horizontal position */
y = (SHORT) SHORT2FROMMP (mp1);    /* vertical position  */
```

The WM_BUTTON3DBLCLK message is sent when the user double-clicks the right mouse button of a three-button mouse.

- Parameters**
- x* Low word of *mp1*. Indicates the horizontal position of the mouse pointer (in window coordinates).
 - y* High word of *mp1*. Indicates the vertical position of the mouse pointer (in window coordinates).
- Return Value** An application should return TRUE if it processes this message.
- See Also** WM_BUTTON3DOWN, WM_BUTTON3UP, WM_HITTEST

■ WM_BUTTON1DOWN

```
WM_BUTTON1DOWN
x = (SHORT) SHORT1FROMMP (mp1);    /* horizontal position */
y = (SHORT) SHORT2FROMMP (mp1);    /* vertical position  */
```

The WM_BUTTON1DOWN message is sent when the user clicks the left mouse button.

- Parameters**
- x* Low word of *mp1*. Indicates the horizontal position of the mouse pointer (in window coordinates).
 - y* High word of *mp1*. Indicates the vertical position of the mouse pointer (in window coordinates).
- Return Value** An application should return TRUE if it processes this message.
- See Also** WM_BUTTON1DBLCLK, WM_BUTTON1UP, WM_HITTEST

■ WM_BUTTON1UP

```
WM_BUTTON1UP
x = (SHORT) SHORT1FROMMP (mp1);    /* horizontal position */
y = (SHORT) SHORT2FROMMP (mp1);    /* vertical position   */
```

The WM_BUTTON1UP message is sent when the user releases the left mouse button.

Parameters *x* Low word of *mp1*. Indicates the horizontal position of the mouse pointer (in window coordinates).

y High word of *mp1*. Indicates the vertical position of the mouse pointer (in window coordinates).

Return Value An application should return TRUE if it processes this message.

See Also WM_BUTTON1DBLCLK, WM_BUTTON1DOWN, WM_HITTEST

■ WM_BUTTON2DOWN

```
WM_BUTTON2DOWN
x = (SHORT) SHORT1FROMMP (mp1);    /* horizontal position */
y = (SHORT) SHORT2FROMMP (mp1);    /* vertical position   */
```

The WM_BUTTON2DOWN message is sent when the user clicks the second mouse button. On a two-button mouse, the second button is the right button.

Parameters *x* Low word of *mp1*. Indicates the horizontal position of the mouse pointer (in window coordinates).

y High word of *mp1*. Indicates the vertical position of the mouse pointer (in window coordinates).

Return Value An application should return TRUE if it processes this message.

See Also WM_BUTTON2DBLCLK, WM_BUTTON2UP, WM_HITTEST

■ WM_BUTTON2UP

```
WM_BUTTON2UP
x = (SHORT) SHORT1FROMMP (mp1);    /* horizontal position */
y = (SHORT) SHORT2FROMMP (mp1);    /* vertical position   */
```

The WM_BUTTON2UP message is sent when the user releases the second mouse button. On a two-button mouse, the second button is the right button.

Parameters *x* Low word of *mp1*. Indicates the horizontal position of the mouse pointer (in window coordinates).

y High word of *mp1*. Indicates the vertical position of the mouse pointer (in window coordinates).

Return Value An application should return TRUE if it processes this message.

See Also WM_BUTTON2DBLCLK, WM_BUTTON2DOWN, WM_HITTEST

■ WM_BUTTON3DOWN

```
WM_BUTTON3DOWN
x = (SHORT) SHORT1FROMMP (mp1);    /* horizontal position */
y = (SHORT) SHORT2FROMMP (mp1);    /* vertical position  */
```

The WM_BUTTON3DOWN message is sent when the user clicks the right mouse button of a three-button mouse.

Parameters *x* Low word of *mp1*. Indicates the horizontal position of the mouse pointer (in window coordinates).

y High word of *mp1*. Indicates the vertical position of the mouse pointer (in window coordinates).

Return Value An application should return TRUE if it processes this message.

See Also WM_BUTTON3DBLCLK, WM_BUTTON3UP, WM_HITTEST

■ WM_BUTTON3UP

```
WM_BUTTON3UP
x = (SHORT) SHORT1FROMMP (mp1);    /* horizontal position */
y = (SHORT) SHORT2FROMMP (mp1);    /* vertical position  */
```

The WM_BUTTON3UP message is sent when the user releases the right mouse button of a three-button mouse.

Parameters *x* Low word of *mp1*. Indicates the horizontal position of the mouse pointer (in window coordinates).

y High word of *mp1*. Indicates the vertical position of the mouse pointer (in window coordinates).

Return Value An application should return TRUE if it processes this message.

See Also WM_BUTTON3DBLCLK, WM_BUTTON3DOWN, WM_HITTEST

■ WM_CALCVALIDRECTS

```
WM_CALCVALIDRECTS
prclSrc = (PRECTL) PVOIDFROMMP (mp1);    /* source rectangle  */
prclDest = (PRECTL) PVOIDFROMMP (mp1);    /* destination rectangle */
```

The WM_CALCVALIDRECTS message is sent when a window is about to be resized. This allows the application to specify the coordinates of a rectangle to be preserved and to designate where this rectangle will be in the resized window. Areas outside this rectangle will be redrawn.

Parameters *prclSrc* Low and high word of *mp1*. Points to the rectangle that contains the dimensions of the window prior to resizing. The rectangle's coordinates are relative to the window's parent window.

prclDest Low and high word of *mp2*. Points to the rectangle that contains the dimensions of the window after resizing. The rectangle's coordinates are relative to the window's parent window.

Return Value If an application processes this message, it can return zero to indicate it has changed the rectangles itself, CV_REDRAW if the entire window is to be redrawn, or a combination of the following values:

Value	Meaning
CVR_ALIGNBOTTOM	Align with the bottom edge of the window.
CVR_ALIGNLEFT	Align with the left edge of the window.
CVR_ALIGNRIGHT	Align with the right edge of the window.
CVR_ALIGNTOP	Align with the top edge of the window.

Comments The WM_CALCVALIDRECTS message is not sent if a window has the style CS_SIZEREDRAW because such windows are always completely redrawn when resized.

See Also WM_ADJUSTWINDOWPOS

■ WM_CHAR

```
WM_CHAR
fsKeyFlags = (USHORT) SHORT1FROMMP(mp1); /* key flags */
uchRepeat = (UCHAR) CHAR3FROMMP(mp1); /* repeat count */
uchScanCode = (UCHAR) CHAR4FROMMP(mp1); /* scan code */
usChr = (USHORT) SHORT1FROMMP(mp2); /* character */
usVKey = (USHORT) SHORT2FROMMP(mp2); /* virtual key */
```

The WM_CHAR message is sent whenever the user presses a key. This message is placed in the queue associated with the window that has the focus.

Parameters *fsKeyFlags* Low word of *mp1*. Specifies the keyboard control codes. It can be one or more of the following values:

Value	Meaning
KC_CHAR	The <i>usChr</i> parameter value is valid; otherwise, <i>mp2</i> contains zero.
KC_SCANCODE	The <i>uchScanCode</i> parameter value is valid; otherwise, <i>uchScanCode</i> contains zero.
KC_VIRTUALKEY	The <i>usVKey</i> parameter value is valid; otherwise, <i>usVKey</i> contains zero.
KC_KEYUP	The event was a key-up transition; otherwise, it was a key-down transition.
KC_PREVDOWN	The key was previously down; otherwise, it was previously up.
KC_DEADKEY	The character code is a dead key. The application must display the glyph for the dead key without advancing the cursor.

Value	Meaning
KC_COMPOSITE	The character code was formed by combining the current key with the previous dead key.
KC_INVALIDCOMP	The character code was not a valid combination with the preceding dead key. The application must advance the cursor past the dead-key glyph and then, if the current character is <i>not</i> a space, it must beep the speaker and display the new character code.
KC_LONEKEY	This bit is set if the key was pressed and released without any other keys being pressed or released between the time the key was pressed and released.
KC_SHIFT	The shift state was active when the key is pressed or released.
KC_ALT	The ALT state was active when the key was pressed or released.
KC_CTRL	The CONTROL state was active when the key was pressed or released.

uchRepeat Low byte of high word of *mp1*. Specifies the repeat count of the key.

uchScanCode High byte of high word of *mp1*. Specifies the character scan code of the character.

usChr Low word of *mp2*. Specifies the ASCII character.

usVKey High word of *mp2*. Specifies the virtual-key code.

Comments

Generally, all WM_CHAR messages generated from actual user input have the KC_SCANCODE code set. However, if the message has been generated by an application that has issued the WinSetHook function to filter keystrokes, or if it was posted to the application queue, this code may not be set.

The CHARMSG macro can be used to access the WM_CHAR message parameters. This macro defines a CHARMSG structure pointer that has the following form:

```
struct _CHARMSG {
    USHORT chr;           /* mp2 */
    USHORT vkey;
    USHORT fs;           /* mp1 */
    UCHAR cRepeat;
    UCHAR scancode;
};
```

Example

This example uses the CHARMSG macro to process a WM_CHAR message. It first uses the macro to determine if a key was released. It then uses the macro to generate a switch statement based on the character received.

```
MRESULT CALLBACK GenericWndProc(hwnd, usMessage, mp1, mp2)
HWND hwnd;
USHORT usMessage;
MPARAM mp1;
MPARAM mp2;
{
    switch (usMessage) {
    case WM_CHAR:
        if (CHARMSG(&usMessage)->fs & KC_KEYUP) {
            switch (CHARMSG(&usMessage)->chr) {
```

Return Value An application should return TRUE if it processes the message; otherwise it should return FALSE.

See Also WinSetHook, WM_NULL, WM_TRANSLATEACCEL, WM_VIOCHAR

■ WM_CLOSE

WM_CLOSE

The WM_CLOSE message is sent as a signal that the window or its application should terminate itself. It allows the window to control the termination process. If this message is passed to the WinDefWindowProc function, the function posts a WM_QUIT message.

Parameters This message does not use any parameters

Return Value An application should return zero if it processes this message.

Example In the following example, the *fChanges* variable is checked. If it is TRUE, the user is asked if he or she wants to exit without saving any changes made. If the user responds by choosing the No button, then zero is returned and the application does not exit. If the user responds by choosing the Yes button, then a WM_QUIT message is posted so that the application will terminate.

```
case WM_CLOSE:
    if (fChanges) {
        if (WinMessageBox(HWND_DESKTOP, hwndClient,
            "Do you want to exit without saving your changes?",
            "", 0, MB_NOICON | MB_YESNO) == MBID_NO)
            return (0L);
    }
    WinPostMsg(hwnd, WM_QUIT, 0L, 0L);
    return (0L);
```

See Also WinDefWindowProc, WinMessageBox, WinPostMsg, WM_QUIT

■ WM_COMMAND

```
WM_COMMAND
usCmd = (USHORT) SHORT1FROMMP(mp1); /* command value */
fsSource = (USHORT) SHORT1FROMMP(mp2); /* source type */
fPointer = (BOOL) SHORT2FROMMP(mp2); /* pointer flag */
```

The WM_COMMAND message is sent to a window when it has a command to report or when a keystroke has been translated by an accelerator table into a WM_COMMAND message.

Parameters *usCmd* Low word of *mp1*. Specifies the command value.

fsSource Low word of *mp2*. Specifies the source type. This parameter can be one of the following values:

Value	Meaning
CMDSRC_ACCELERATOR	Posted as the result of an accelerator. The <i>usCmd</i> parameter is the accelerator command value.

Value	Meaning
CMDSRC_MENU	Posted by a menu control. The <i>usCmd</i> parameter is the identifier of the menu item.
CMDSRC_PUSHBUTTON	Posted by a push-button control. The <i>usCmd</i> parameter is the window identifier of the push button.
CMDSRC_OTHER	Other source. The <i>usCmd</i> parameter gives further control-specific information defined for each control type.

fPointer High word of *mp2*. Indicates if the message was posted as a result of a pointing-device (mouse) operation. A value of TRUE indicates a pointing device was used; a value of FALSE indicates a keyboard operation.

Return Value An application should return zero if it processes this message.

See Also WM_CONTROL, WM_HELP, WM_MENUSELECT, WM_TRANSLATEACCEL

■ WM_CONTROL

```
WM_CONTROL
id = (USHORT) SHORT1FROMMP(mp1);          /* window identifier */
usNotifyCode = (USHORT) SHORT2FROMMP(mp1); /* notification code */
usData = (ULONG) LONGFROMMP(mp2);        /* control data */
```

The WM_CONTROL message is sent when a control window has an event to report to its owner.

Parameters *id* Low word of *mp1*. Identifies the control window.

usNotifyCode High word of *mp1*. The meaning of this parameter depends on the control sending the message.

usData Low and high word of *mp2*. Contains control-specific information. The meaning of the notification code and the control-specific information depends on the type of control.

Return Value An application should return zero if it processes this message.

See Also WM_COMMAND

■ WM_CONTROLHEAP

```
WM_CONTROLHEAP
```

The WM_CONTROLHEAP message is sent to a control's owner when the control must get a handle to a heap from which to allocate memory. (For example, entry-field controls allocate memory to hold the text associated with the control.)

Usually, an application can ignore this message, passing it on to the default window procedure, which then returns a handle to a heap maintained by the system for each message queue for this purpose.

Parameters	This message does not use any parameters.
Return Value	An application should return a heap handle if it processes this message.
See Also	WinCreateHeap

■ WM_CONTROLPOINTER

```
WM_CONTROLPOINTER
id = (USHORT) SHORT1FROMMP(mp1);    /* sender ID */
hptr = (HPOINTER) LONGFROMMP(mp2);  /* handle to mouse pointer */
```

The WM_CONTROLPOINTER message is sent to a control's owner when the mouse pointer moves over the control window, allowing the owner to set the mouse pointer to a different shape. The control passes an HPOINTER handle to a mouse pointer as part of this message. An application can alter the default pointer shape by passing back a different HPOINTER handle as the result of this message.

Parameters	<i>id</i> Low word of <i>mp1</i> . Identifies the control window sending the message. <i>hptr</i> Low and high word of <i>mp2</i> . Identifies the mouse pointer that the control is to use.
Return Value	An application that processes this message should return the handle of the mouse pointer to be used while the mouse is positioned over the control window.
See Also	WinCreatePointer

■ WM_CREATE

```
WM_CREATE
pCtlData = (PVOID) PVOIDFROMMP(mp1);    /* pointer to class data */
pcrst = (PCREATESTRUCT) PVOIDFROMMP(mp2); /* pointer to structure */
```

The WM_CREATE message is sent when an application requests that a window be created. The window procedure for the new window receives this message after the window is created but before the window becomes visible.

Parameters	<i>pCtlData</i> Low and high word of <i>mp1</i> . Points to the buffer that has class-specific information. This data is passed to the WinCreateWindow function as a parameter. <i>pcrst</i> Low and high word of <i>mp2</i> . Points to a CREATESTRUCT structure. The CREATESTRUCT structure has the following form:
-------------------	--

```
typedef struct _CREATESTRUCT { /* crst */
    PVOID pPresParams;
    PVOID pCtlData;
    USHORT id;
    HWND hwndInsertBehind;
    HWND hwndOwner;
    SHORT cy;
    SHORT cx;
    SHORT y;
    SHORT x;
    ULONG flStyle;
    PSZ pszText;
    PSZ pszClass;
    HWND hwndParent;
} CREATESTRUCT;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value An application should return FALSE if the system should continue creating the window or TRUE if it should not.

See Also WinCreateWindow, WM_INITDLG, WM_SIZE

■ WM_DDE_ACK

```
WM_DDE_ACK
mp1 = MPFROMHWORD(hwnd); /* sender's window */
mp2 = MPFROM2WORD(pdde); /* pointer to DDESTRUCT structure */
```

The WM_DDE_ACK message notifies an application of the receipt and processing of a WM_DDE_EXECUTE, WM_DDE_DATA, WM_DDE_ADVISE, WM_DDE_UNADVISE, or WM_DDE_POKE message, and, in some cases, of a WM_DDE_REQUEST message. The acknowledging application modifies the fsStatus field of the DDESTRUCT structure to return information about the status of the message received.

Parameters *hwnd* Low and high word of *mp1*. Identifies the sender application’s window.
pdde Low and high word of *mp2*. Points to a DDESTRUCT structure. The DDESTRUCT structure has the following form:

```
typedef struct _DDESTRUCT {
    ULONG cbData;
    USHORT fsStatus;
    USHORT usFormat;
    USHORT offszItemName;
    USHORT offfabData;
} DDESTRUCT;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

See Also WM_DDE_ADVISE, WM_DDE_DATA, WM_DDE_EXECUTE, WM_DDE_POKE, WM_DDE_REQUEST, WM_DDE_UNADVISE

■ WM_DDE_ADVISE

```
WM_DDE_ADVISE
mp1 = MPFROMHWNDR(hwnd); /* sender's window */
mp2 = MPFROMP(pdde); /* pointer to DDESTRUCT structure */
```

The WM_DDE_ADVISE message is sent from a client application to a server application requesting that the server supply an update for a data item whenever it changes. The server application is expected to reply with a positive WM_DDE_ACK message if it can provide the requested data, or with a negative message if it cannot.

Parameters *hwnd* Low and high word of *mp1*. Identifies the sender application's window.
pdde Low and high word of *mp2*. Points to a DDESTRUCT structure. The DDESTRUCT structure has the following form:

```
typedef struct _DDESTRUCT {
    ULONG    cbData;
    USHORT   fsStatus;
    USHORT   usFormat;
    USHORT   offszItemName;
    USHORT   offfabData;
} DDESTRUCT;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

See Also WM_DDE_ACK, WM_DDE_UNADVISE

■ WM_DDE_DATA

```
WM_DDE_DATA
mp1 = MPFROMHWNDR(hwnd); /* sender's window */
mp2 = MPFROMP(pdde); /* pointer to DDESTRUCT structure */
```

The WM_DDE_DATA message is sent from a server application to a client application to notify the client application that the data it requested is available.

Parameters *hwnd* Low and high word of *mp1*. Identifies the sender application's window.
pdde Low and high word of *mp2*. Points to a DDESTRUCT structure. The DDESTRUCT structure has the following form:

```
typedef struct _DDESTRUCT {
    ULONG    cbData;
    USHORT   fsStatus;
    USHORT   usFormat;
    USHORT   offszItemName;
    USHORT   offfabData;
} DDESTRUCT;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

See Also WM_DDE_ACK, WM_DDE_REQUEST

■ WM_DDE_EXECUTE

```
WM_DDE_EXECUTE
mp1 = MPFROMHWORD (hwnd);    /* sender's window */
mp2 = MPFROM2SHORT (pdde);  /* pointer to DDESTRUCT structure */
```

The WM_DDE_EXECUTE message is sent from a client application to a server application. It passes a text string that the server should execute as a series of commands.

Parameters *hwnd* Low and high word of *mp1*. Identifies the sender application's window.
 pdde Low and high word of *mp2*. Points to a DDESTRUCT structure. The DDESTRUCT structure has the following form:

```
typedef struct _DDESTRUCT {
    ULONG    cbData;
    USHORT   fsStatus;
    USHORT   usFormat;
    USHORT   offszItemName;
    USHORT   offabData;
} DDESTRUCT;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

See Also WM_DDE_ACK

■ WM_DDE_INITIATE

```
WM_DDE_INITIATE
mp1 = (MPARAM) ((HWND) hwnd);    /* sender's window */
mp2 = MPFROM2SHORT (pddei);    /* pointer to DDEINIT structure */
```

The WM_DDE_INITIATE message is sent by a client application to exchange data with one or more server applications. This message is often sent to all current applications by calling WinBroadcastMsg.

Parameters *hwnd* Low and high word of *mp1*. Identifies the sender application's window.
 pddei Low and high word of *mp2*. Points to a DDEINIT structure that contains an application name and a topic name. All applications with matching names that support the topic are expected to acknowledge by calling the WinDdeRespond function. The DDEINIT structure has the following form:

```
typedef struct _DDEINIT {
    USHORT   cb;
    PSZ      pszAppName;
    PSZ      pszTopic;
} DDEINIT;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

See Also WinBroadcastMsg, WinDdeRespond, WM_DDE_INITIATEACK,
 WM_DDE_TERMINATE

■ WM_DDE_INITIATEACK

```
WM_DDE_INITIATEACK
mp1 = MPFROMHWND (hwnd);          /* sender's window */
mp2 = MPFROMP (pddei);           /* pointer to DDEINIT structure */
```

The WM_DDE_INITIATEACK message is sent as a positive response to a WM_DDE_INITIATE message for each topic an application supports. An application should use the WinDdeRespond function to send this message.

Parameters

hwnd Low and high word of *mp1*. Identifies the sender application's window.

pddei Low and high word of *mp2*. Points to a DDEINIT structure. The DDEINIT structure has the following form:

```
typedef struct _DDEINIT {
    USHORT  cb;
    PSZ     pszAppName;
    PSZ     pszTopic;
} DDEINIT;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

See Also WinDdeRespond, WM_DDE_INITIATE

■ WM_DDE_POKE

```
WM_DDE_POKE
mp1 = MPFROMHWND (hwnd);          /* sender's window */
mp2 = MPFROMP (pdde);           /* pointer to DDESTRUCT structure */
```

The WM_DDE_POKE message sends an unsolicited data message to the receiving application, which should then reply with a WM_DDE_ACK message indicating whether it accepted the data.

Parameters

hwnd Low and high word of *mp1*. Identifies the sender application's window.

pdde Low and high word of *mp2*. Points to a DDESTRUCT structure. The DDESTRUCT structure has the following form:

```
typedef struct _DDESTRUCT {
    ULONG   cbData;
    USHORT  fsStatus;
    USHORT  usFormat;
    USHORT  offszItemName;
    USHORT  offabData;
} DDESTRUCT;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

See Also WM_DDE_ACK

■ WM_DDE_REQUEST

```
WM_DDE_REQUEST
mp1 = MPEFROMHWND (hwnd);    /* sender's window */
mp2 = MPEFROMP (pdde);      /* pointer to DDESTRUCT structure */
```

The WM_DDE_REQUEST message is posted from a client application to a server application to request that the server provide a data item to the client. The receiving application is expected to respond with a WM_DDE_DATA message that contains the requested data, if possible, or with a negative WM_DDE_ACK message.

Parameters *hwnd* Low and high word of *mp1*. Identifies the sender application's window.
pdde Low and high word of *mp2*. Points to a DDESTRUCT structure. The DDESTRUCT structure has the following form:

```
typedef struct _DDESTRUCT {
    ULONG    cbData;
    USHORT   fsStatus;
    USHORT   usFormat;
    USHORT   offszItemName;
    USHORT   offfabData;
} DDESTRUCT;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

See Also WM_DDE_ACK, WM_DDE_DATA

■ WM_DDE_TERMINATE

```
WM_DDE_TERMINATE
mp1 = (MPARAM) ((HWND) hwnd);    /* sender's window */
mp2 = OL;
```

The WM_DDE_TERMINATE message is sent by a client application or a server application to terminate the exchange. An application is expected to send a WM_DDE_TERMINATE message in response to this message.

Parameters *hwnd* Low and high word of *mp1*. Identifies the sender application's window.

See Also WM_DDE_INITIATE

■ WM_DDE_UNADVISE

```
WM_DDE_UNADVISE
mp1 = MPEFROMHWND (hwnd);    /* sender's window */
mp2 = MPEFROMP (pdde);      /* pointer to DDESTRUCT structure */
```

The WM_DDE_UNADVISE message is sent from a client application to a server application. It indicates that the specified item no longer should be updated and that the server application should remove the link to the data item set up by the WM_DDE_ADVISE message. The receiving application is expected to reply with a positive WM_DDE_ACK message if it can process the request or a negative message if it cannot.

Parameters *hwnd* Low and high word of *mp1*. Identifies the sender application's window.
 pdde Low and high word of *mp2*. Points to a DDESTRUCT structure. The DDESTRUCT structure has the following form:

```
typedef struct _DDESTRUCT {
    ULONG     cbData;
    USHORT    fsStatus;
    USHORT    usFormat;
    USHORT    offszItemName;
    USHORT    offfabData;
} DDESTRUCT;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

See Also WM_DDE_ACK, WM_DDE_ADVISE

■ WM_DESTROY

WM_DESTROY

The WM_DESTROY message is sent when a window is being destroyed. It is sent to the window procedure of the window being destroyed after the window is hidden.

This message is sent first to the window being destroyed, then to the child windows as they are destroyed. During the processing of the WM_DESTROY message, it can be assumed that all child windows still exist.

Parameters The message does not have any parameters.

Return Value An application should return zero if it processes this message.

See Also WinDestroyWindow, WM_CLOSE

■ WM_DESTROYCLIPBOARD

WM_DESTROYCLIPBOARD

The WM_DESTROYCLIPBOARD message is sent when the clipboard is emptied as the result of a call to the WinEmptyClipbrd function.

Parameters This message does not use any parameters.

Return Value An application should return zero if it processes this message.

See Also WinEmptyClipbrd

■ WM_DRAWCLIPBOARD

WM_DRAWCLIPBOARD

The WM_DRAWCLIPBOARD message is sent as a result of a call to the WinCloseClipbrd function if the contents of the clipboard have changed.

Parameters	This message does not use any parameters.
Return Value	An application should return zero if it processes this message.
See Also	WinCloseClipbrd, WM_PAINTCLIPBOARD

■ WM_DRAWITEM

```
WM_DRAWITEM
id = (USHORT) SHORT1FROMMP(mp1);      /* window ID          */
poi = (POWNERITEM) PVOIDFROMMP(mp2); /* pointer to OWNERITEM */
```

The WM_DRAWITEM message is sent to the owner of a list box when an item in an owner-drawn list needs to be drawn or highlighted. The list box must have the LS_OWNERDRAW style.

Parameters	<p><i>id</i> Low word of <i>mp1</i>. Identifies the window of the list-box control sending this message.</p> <p><i>poi</i> Low and high word of <i>mp2</i>. Points to an OWNERITEM structure. The OWNERITEM structure has the following form:</p>
-------------------	---

```
typedef struct _OWNERITEM {
    HWND    hwnd;
    HPS     hps;
    USHORT  fsState;
    USHORT  fsAttribute;
    USHORT  fsStateOld;
    USHORT  fsAttributeOld;
    RECTL   rcItem;
    SHORT   idItem;
    ULONG   hItem;
} OWNERITEM;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

Return Value	The return value is ignored.
Comments	<p>When an item is to be drawn, the <i>fsState</i> field and the <i>fsStateOld</i> field of the OWNERITEM structure will be equal. The application should draw the item and return TRUE, or it should return FALSE to let the list box draw the item. The list box can draw only text items, so the application must handle the drawing of other types of objects.</p> <p>When an item is to be highlighted, the <i>fsState</i> field is TRUE and the <i>fsStateOld</i> field is FALSE. In this case, the application should carry out the highlighting and set <i>fsState</i> and <i>fsStateOld</i> equal to FALSE before returning TRUE, or it should return FALSE so the list box can perform default highlighting of the item.</p>

When highlighting is to be removed from an item, the `fsState` field is `FALSE` and the `fsStateOld` field is `TRUE`. An application can remove the highlighting, set the `fsState` and `fsStateOld` equal to `FALSE` and return `TRUE`, or it can return `FALSE` to let the list box remove the highlighting.

The `WM_DRAWITEM` message is also sent by menu items that have the `MIS_OWNERDRAW` style.

See Also `LM_QUERYITEMTEXT`

■ WM_ENABLE

```
WM_ENABLE
fEnable = SHORT1FROMMP(mp1);    /* enable flag */
```

The `WM_ENABLE` message is sent when an application changes the enabled state of a window. It is sent to the window whose enabled state is changing. This message is always sent before the `WinEnableWindow` function returns, but after the window enabled state (`WS_DISABLE` style bit) has changed.

Parameters *fEnable* Low word of *mp1*. Specifies whether the window is being enabled or disabled. A value of `TRUE` means the window is being enabled; `FALSE` means the window is being disabled.

Return Value An application should return zero if it processes this message.

See Also `WinEnableWindow`

■ WM_ERASEBACKGROUND

```
WM_ERASEBACKGROUND
hps = (HPS) LONGFROMMP(mp1);    /* presentation-space handle */
prcl = (PRECTL) PVIODFROMMP(mp1); /* pointer to RECTL structure */
```

The `WM_ERASEBACKGROUND` message is sent by the frame window to a client window when the background is to be redrawn. Usually, an application ignores this message and erases and redraws the window when the `WM_PAINT` message is received. However, the `WM_ERASEBACKGROUND` message can be valuable in improving the speed of window rearrangement operations by making the window images consistent on the screen as soon as the rearrangement takes place.

Parameters *hps* Low and high word of *mp1*. Identifies a presentation space for the frame window (not the client window).

prcl Low and high word of *mp2*. Points to a **RECTL** structure that contains the rectangle to be painted. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

If an application returns **TRUE**, the frame-window procedure erases the rectangle of the frame window covered by the client window by filling it with the system color **SYSCLR_WINDOW**. An application should return **FALSE** to prevent the frame window from erasing the background.

Comments

The client window may use the presentation-space handle to selectively erase portions of its window, and then return **FALSE** so that the frame window procedure does not erase the background.

Note that the presentation-space handle is to the frame window, not to the client window. The rectangle to be erased may include areas outside the client window.

An application should not rely on the **WM_ERASEBACKGROUND** message for notification as to when to repaint an invalidated client window. The **WM_ERASEBACKGROUND** message is sent by the frame-window procedure as part of processing the **WM_PAINT** message. However, if only the client window has been invalidated, the client window does not receive a **WM_ERASEBACKGROUND** message and the frame window does not receive the **WM_PAINT** message. An application can monitor successive **WM_ERASEBACKGROUND** and **WM_PAINT** messages by using a flag, and then only erase the background when processing **WM_PAINT** messages if there is not an immediately preceding **WM_ERASEBACKGROUND** message.

See Also

WM_PAINT

■ WM_FLASHWINDOW

```
WM_FLASHWINDOW
fFlash = (BOOL) SHORT1FROMMP(mp1); /* flash flag */
```

The **WM_FLASHWINDOW** message is sent to the frame window as a result of a call to the **WinFlashWindow** function.

Parameters

fFlash Low word of *mp1*. Specifies whether the window is to start or stop flashing. A value of **TRUE** starts the flashing; **FALSE** stops the flashing.

Return Value

The frame-window procedure returns **TRUE** if the start/stop command is successful or **FALSE** if an error occurs.

See Also

WinFlashWindow

■ WM_FOCUSCHANGE

```
WM_FOCUSCHANGE
hwnd = (HWND) HWNDFROMMP(mp1);          /* window handle */
fGetFocus = (BOOL) SHORT1FROMMP(mp2);   /* focus flag */
fsFocusChange = (USHORT) SHORT2FROMMP(mp2); /* focus-change flags */
```

The WM_FOCUSCHANGE message is sent when the focus-window changes. It is sent to both the window gaining the focus and the window losing the focus.

Parameters

hwnd Low and high word of *mp1*. Identifies the window gaining or losing the focus.

fGetFocus Low word of *mp2*. Specifies whether the window is gaining or losing the focus. A value of TRUE means the window is gaining the focus; FALSE means the window is losing the focus.

fsFocusChange High word of *mp2*. Specifies flags that modify the focus-change process. This parameter can be any combination of the following values:

Value	Meaning
FC_NOSETFOCUS	Do not send a WM_SETFOCUS message to the window receiving the focus.
FC_NOLOSEFOCUS	Do not send a WM_SETFOCUS message to the window losing the focus.
FC_NOSETACTIVE	Do not send a WM_ACTIVATE message to the window becoming active.
FC_NOLOSEACTIVE	Do not send a WM_ACTIVATE message to the window being deactivated.
FC_NOSETSELECTION	Do not send a WM_SETSELECTION message to the window receiving the selection.
FC_NOLOSESELECTION	Do not send a WM_SETSELECTION message to the window losing the selection.

Return Value An application should return zero if it processes this message.

See Also WinFocusChange, WinSetFocus, WM_ACTIVATE, WM_QUERYFOCUSCHAIN, WM_SETFOCUS, WM_SETSELECTION

■ WM_FORMATFRAME

```
WM_FORMATFRAME
prcl = (PRECTL) PVOIDFROMMP(mp1);      /* pointer to RECTL structure */
pswp = (PSWP) PVOIDFROMMP(mp2);       /* pointer to SWP array */
```

The WM_FORMATFRAME message is sent to a frame window to calculate the sizes and positions of all the frame controls and the client window. The frame-window procedure sends the message to its client window, and if the client window returns TRUE (indicating that it processed the message) no further action is taken. Otherwise, the frame window calls the WinFormatFrame function.

Parameters

prcl Low and high word of *mp2*. Points to a **RECTL** structure that contains the rectangle within which the frame controls are formatted. The **RECTL** structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

pswp Low and high word of *mp1*. Points to an array of **SWP** structures. The array elements are filled in the order of the **FID** values of the frame controls, with the **FID_CLIENT** window always the last element in the array. The **SWP** structure has the following form:

```
typedef struct _SWP {
    USHORT fs;
    SHORT  cy;
    SHORT  cx;
    SHORT  y;
    SHORT  x;
    HWND   hwndInsertBehind;
    HWND   hwnd;
} SWP;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

An application should return **TRUE** if it processes this message.

Comments

Note that the *pswp* parameter points to memory allocated according to the value returned by the **WM_QUERYFRAMECTLCOUNT** message. The application must not write beyond this area.

See Also

WinFormatFrame

■ WM_HELP

```
WM_HELP
usCmd = (USHORT) SHORT1FROMMP(mp1);      /* command value */
fsSource = (USHORT) SHORT1FROMMP(mp2);  /* source type */
fPointer = (BOOL) SHORT2FROMMP(mp2);    /* pointer flag */
```

The **WM_HELP** message is sent when a keystroke is translated by an accelerator table into a **WM_HELP** message. A **WM_HELP** message can also be posted by menu controls and buttons, if they have the appropriate style.

The **WM_HELP** message is identical to the **WM_COMMAND** message, but **WM_HELP** implies that the application should respond by displaying help information.

Parameters

usCmd Low word of *mp1*. Specifies the command value.

fsSource Low word of *mp2*. Specifies the source type. This parameter can be one of the following values:

Value	Meaning
CMDSRC_ACCELERATOR	Posted as the result of an accelerator. The <i>usCmd</i> parameter is the accelerator command value.

Value	Meaning
CMDSRC_MENU	Posted by a menu control. The <i>usCmd</i> parameter is the identifier of the menu item.
CMDSRC_PUSHBUTTON	Posted by a push-button control. The <i>usCmd</i> parameter is the window identifier of the push button.
CMDSRC_OTHER	Other source. The <i>usCmd</i> parameter gives further control-specific information defined for each control type.

fPointer High word of *mp2*. Indicates whether the message was posted as a result of a pointing-device (mouse) operation. A value of TRUE indicates a pointing device was used; FALSE indicates a keyboard operation.

Return Value An application should return zero if it processes this message.

See Also WM_COMMAND, WM_MENUSELECT, WM_TRANSLATEACCEL

■ WM_HITTEST

```
WM_HITTEST
xPos = SHORT1FROMMP(mp1); /* x position */
yPos = SHORT2FROMMP(mp1); /* y position */
```

The WM_HITTEST message is sent when an application requests a message by issuing a WinGetMsg or WinPeekMsg function. If the message that would be retrieved represents a pointer-related event, the WM_HITTEST message is sent to a window to determine whether the message is destined for that window.

Parameters *xPos* Low word of *mp1*. Specifies the horizontal position of the mouse (in window coordinates), relative to the left side of the window.

yPos High word of *mp1*. Specifies the vertical position of the mouse (in window coordinates), relative to the bottom of the window.

Return Value An application that processes this message should return one of the following values:

Value	Meaning
HT_NORMAL	The message (obtained from WinPeekMsg or WinGetMsg) should be processed as normal.
HT_TRANSPARENT	The part of the window under the pointer is transparent; hit testing should continue on windows beneath this window, as if the window did not exist.
HT_DISCARD	The message should be discarded; no message should be posted to the application.
HT_ERROR	Processed like HT_DISCARD. The message should be discarded; no message should be posted to the application except if the message is a button-down message, an alarm sounds.

Comments The handling of this message determines whether a disabled window can process mouse clicks. The default window procedure returns `HT_ERROR` if the window is disabled; otherwise, it returns `HT_NORMAL`.

See Also `WinGetMsg`, `WinPeekMsg`, `WM_MOUSEMOVE`

■ WM_HSCROLL

```
WM_HSCROLL
id = SHORT1FROMMP(mp1);      /* control-window identifier */
sPos = SHORT1FROMMP(mp2);    /* slider position           */
usCmd = SHORT2FROMMP(mp2);   /* command                    */
```

The `WM_HSCROLL` message is posted to the owner of a horizontal scroll-bar window when an event occurs.

Parameters *id* Low word of *mp1*. Identifies the scroll-bar window.

sPos Low word of *mp2*. Specifies the slider position.

usCmd High word of *mp2*. Specifies the type of command. This parameter can be one of the following values:

Value	Meaning
<code>SB_LINELEFT</code>	The user clicked the left scroll-bar arrow or pressed the <code>VK_LEFT</code> key.
<code>SB_LINERIGHT</code>	The user clicked the right scroll-bar arrow or pressed the <code>VK_RIGHT</code> key.
<code>SB_PAGELEFT</code>	The user clicked the area to the left of the slider or pressed the <code>VK_PAGELEFT</code> key.
<code>SB_PAGERIGHT</code>	The user clicked the area to the right of the slider or pressed the <code>VK_PAGERIGHT</code> key.
<code>SB_SLIDERPOSITION</code>	The <i>sPos</i> parameter contains the final position of the slider.
<code>SB_SLIDERTRACK</code>	The user is dragging the slider. This value is sent whenever the slider position changes.
<code>SB_ENDSCROLL</code>	The user has finished scrolling. This value is set only if the user was not doing an absolute slider movement.

Return Value An application should return zero if it processes this message.

See Also `WM_HSCROLLCLIPBOARD`, `WM_VSCROLL`

■ WM_HSCROLLCLIPBOARD

```
WM_HSCROLLCLIPBOARD
hwndClip = HWNDEROMMP(mp1); /* handle of clipboard owner */
sPos = SHORT1FROMMP(mp2);  /* slider position           */
usCmd = SHORT2FROMMP(mp2); /* command                    */
```

The `WM_HSCROLLCLIPBOARD` message is sent by the clipboard viewer to the clipboard owner when the clipboard data has the `CFL_OWNERDRAW`

attribute and there is an event in the clipboard viewer's horizontal scroll bar. The owner should scroll the clipboard image, invalidate the appropriate sections, and update the scroll-bar values.

Parameters

hwndClip Low and high word of *mp1*. Identifies the clipboard viewer.

sPos Low word of *mp2*. Specifies the slider position.

usCmd High word of *mp2*. Specifies the type of command. This parameter can be one of the following values:

Value	Meaning
SB_LINELEFT	The user clicked the left scroll-bar arrow or pressed the VK_LEFT key.
SB_LINERIGHT	The user clicked the right scroll-bar arrow or pressed the VK_RIGHT key.
SB_PAGELEFT	The user clicked the area to the left of the slider or pressed the VK_PAGELEFT key.
SB_PAGERIGHT	The user clicked the area to the right of the slider or pressed the VK_PAGERIGHT key.
SB_SLIDERPOSITION	The <i>sPos</i> parameter contains the final position of the slider.
SB_SLIDERTRACK	The user is dragging the slider. This value is sent every time the slider position changes.
SB_ENDSCROLL	The user has finished scrolling. This value is set only if the user was not doing an absolute slider movement.

Return Value

An application should return zero if it processes this message.

See Also

WM_HSCROLL, WM_VSCROLLCLIPBOARD

■ WM_INITDLG

```
WM_INITDLG
hwnd = (HWND) HWNDEROMMP(mp1);      /* window handle */
pCreateParams = PVOIDFROMMP(mp2);  /* application-specific data */
```

The WM_INITDLG message is sent when a dialog box is being created. This message is sent to the dialog procedure, before the dialog box is displayed.

Parameters

hwnd Low and high word of *mp1*. Identifies the window that receives the focus when FALSE is returned. This value is set to the first tab-stop child window in the dialog window.

pCreateParams Low and high word of *mp2*. Points to application-specific data passed by calls to the WinCreateDlg, WinDlgBox, and WinLoadDlg functions.

Return Value

An application should return TRUE if the dialog procedure alters the window that is to receive the focus by issuing a WinSetFocus function with the handle of another control within the dialog box. Otherwise, it should return FALSE.

See Also

WinCreateDlg, WinDlgBox, WinLoadDlg, WinSetFocus

■ WM_INITMENU

```
WM_INITMENU
id = SHORT1FROMMP(mp1);          /* menu identifier */
hwnd = (HWND) HWNDFROMMP(mp2);  /* menu-window handle */
```

The WM_INITMENU message is sent when a menu is about to become active. This allows the application to modify the menu before it is displayed.

Parameters	<i>id</i> Low word of <i>mp1</i> . Specifies the menu identifier.
	<i>hwnd</i> Low and high word of <i>mp2</i> . Identifies the menu.
Return Value	An application should return zero if it processes this message.
See Also	MM_ISITEMVALID, WM_MENUEND

■ WM_JOURNALNOTIFY

Form 1 (Journaling WinQueryQueueStatus)

```
WM_JOURNALNOTIFY
ulCmd = LONGFROMMP(mp1);          /* calling function */
ulQueStatus = LONGFROMMP(mp2);   /* queue status */
```

Form 2 (Journaling WinGetPhysKeyState)

```
WM_JOURNALNOTIFY
ulCmd = LONGFROMMP(mp1);          /* calling function */
sc = SHORT1FROMMP(mp2);          /* virtual key */
fsPhysKeyState = SHORT2FROMMP(mp2); /* physical-key state */
```

A WM_JOURNALNOTIFY message allows the WinQueryQueueStatus and WinGetPhysKeyState functions to work properly in journaling situations.

Parameters	<p><i>ulCmd</i> Low and high word of <i>mp1</i>. Specifies the function that was called. This value is JRN_QUEUESTATUS for the WinQueryQueueStatus function and JRN_PHYSKEYSTATE for the WinGetPhysKeyState function.</p> <p><i>ulQueStatus</i> Low and high word of <i>mp2</i>. This parameter is used when the <i>ulCmd</i> parameter is JRN_QUEUESTATUS. This parameter contains the queue status returned by the WinQueryQueueStatus function.</p> <p><i>sc</i> Low word of <i>mp2</i>. This parameter is used when the <i>ulCmd</i> parameter is JRN_PHYSKEYSTATE. This parameter specifies the virtual-key value in the low byte and contains zero in the high byte.</p> <p><i>fsPhysKeyState</i> High word of <i>mp2</i>. This parameter is used when the <i>ulCmd</i> parameter is JRN_PHYSKEYSTATE. This parameter specifies the physical-key state, returned by a call to WinGetPhysKeyState. The 0x8000 bit is set (less than zero) if the key is down; it is clear if the key is up. The 0x0001 bit is set if the key has been pressed since the last time WinGetPhysKeyState was called; it is clear if the key has not been pressed. This 0x0001 bit is cleared by a call to WinGetPhysKeyState.</p>
Return Value	An application should return zero if it processes this message.

Comments

Both of these functions depend on scanning a complete message queue, but journal playback effectively uses a queue that is just one message long.

To fix these journal-related problems, calls to **WinQueryQueueStatus** and **WinGetPhysKeyState** must be recorded along with appropriate state information. This is done using WM_JOURNALNOTIFY messages. If the functions have new information to return since the last time they were called, and there is a journal-record hook installed, the system sends a WM_JOURNALNOTIFY message, carrying a function indicator and the new state information. During journal playback, the system interprets the WM_JOURNALNOTIFY message and changes the appropriate physical-key state entry or queue status to reflect the state of the system at the time the message was recorded.

Because the **WinQueryQueueStatus** and **WinGetPhysKeyState** functions can be called by applications other than the one currently processing input, it is possible that the journal-record hook will be called by two threads simultaneously. For this reason, it is important that the journal library use semaphores when accessing global variables.

See Also

WinGetPhysKeyState, **WinQueryQueueStatus**

■ WM_MATCHMNEMONIC

```
WM_MATCHMNEMONIC
usChar = SHORT1FROMMP (mp1);    /* character to match */
```

The WM_MATCHMNEMONIC message is sent by a dialog box to a control window to determine if a typed character matches a mnemonic in the control window's text.

Parameters

usChar Low word of *mp1*. Specifies the character.

Return Value

An application that processes this message should return TRUE if the mnemonic is found or FALSE if it is not found.

■ WM_MEASUREITEM

```
WM_MEASUREITEM
id = SHORT1FROMMP (mp1);        /* list-box identifier */
poi = (POWNERITEM) PVOIDFROMMP (mp2); /* pointer to OWNERITEM */
```

The WM_MEASUREITEM message is sent to calculate the height of each item in a window. It is normally sent to list boxes and menus. All items are the same height in a list box or menu.

Parameters

id Low word of *mp1*. Specifies the window.

poi Low and high word of *mp2*. When this message is sent to a menu window, this parameter points to an OWNERITEM structure. Otherwise, this parameter is not used. The OWNERITEM structure has the following form:

```
typedef struct _OWNERITEM {
    HWND    hwnd;
    HPS     hps;
    USHORT  fsState;
    USHORT  fsAttribute;
    USHORT  fsStateOld;
    USHORT  fsAttributeOld;
    RECTL   rcItem;
    SHORT   idItem;
    ULONG   hItem;
} OWNERITEM;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value An application should return the height of a window item.

See Also LM_SETITEMHEIGHT

■ WM_MENUEND

```
WM_MENUEND
idMenu = SHORT1FROMMP (mp1);    /* menu identifier */
hwndMenu = LONGFROMMP (mp2);    /* menu window */
```

The WM_MENUEND message is sent when a menu is about to terminate. This allows the application to free any resources that were allocated to process the message.

Parameters *idMenu* Low word of *mp1*. Specifies the menu that is terminating.

hwndMenu Low and high word of *mp2*. Identifies the menu window.

Return Value An application should return zero if it processes this message.

See Also MM_ENDMENUMODE, WM_INITMENU

■ WM_MENUSELECT

```
WM_MENUSELECT
idItem = SHORT1FROMMP (mp1);    /* item identifier */
fPost = (BOOL) SHORT2FROMMP (mp1); /* post flag */
hwndMenu = LONGFROMMP (mp2);    /* menu window */
```

The WM_MENUSELECT message is sent to the owner of a menu window when a menu item is selected.

Parameters *idItem* Low word of *mp1*. Specifies the selected menu item.

fPost High word of *mp1*. Indicates whether a WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message is to be posted. A value of TRUE means that a message will be posted; FALSE means that it will not. An application can prevent the posting of a message by returning FALSE after processing this message.

hwndMenu Low and high word of *mp2*. Identifies the menu window.

Return Value If the *fPost* parameter is FALSE, the return value is ignored. If *fPost* is TRUE, an application should return TRUE to post a WM_COMMAND, WM_HELP, or WM_SYSCOMMAND message and dismiss the menu. An application should return FALSE to prevent posting a message and to prevent the menu from being dismissed.

See Also WM_COMMAND, WM_HELP, WM_SYSCOMMAND

■ WM_MINMAXFRAME

```
WM_MINMAXFRAME
pswp = PVOIDFROMMP(mp1);    /* pointer to SWP structure */
```

The WM_MINMAXFRAME message is sent to a frame window when it is about to be minimized, maximized, or restored.

Parameters *pswp* Low and high word of *mp1*. Points to an SWP structure. The *fs* field specifies the type of action that is to take place (minimize, maximize, or restore). The SWP structure has the following form:

```
typedef struct _SWP {
    USHORT fs;
    SHORT cy;
    SHORT cx;
    SHORT y;
    SHORT x;
    HWND hwndInsertBehind;
    HWND hwnd;
} SWP;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value An application should return TRUE if no further processing should occur. Otherwise, it should return FALSE.

■ WM_MOUSEMOVE

```
WM_MOUSEMOVE
x = SHORT1FROMMP(mp1);      /* horizontal position */
y = SHORT2FROMMP(mp1);      /* vertical position */
usHit = SHORT1FROMMP(mp2);  /* hit-test result */
```

The WM_MOUSEMOVE message is sent to a window when the mouse pointer moves. If the mouse is not captured, the message goes to the window beneath the mouse pointer. Otherwise, the message goes to the mouse-capture window.

Parameters *x* Low word of *mp1*. Specifies the horizontal mouse coordinate, relative to the window's lower-left corner.

y High word of *mp1*. Specifies the vertical mouse coordinate, relative to the window's lower-left corner.

usHit Low word of *mp2*. Specifies the result of a WM_HITTEST message or is zero if a mouse-capture operation is in process.

Return Value An application should return TRUE if it processes this message or FALSE if it does not.

Comments An application that processes the WM_MOUSEMOVE message and does not call the `WinDefWindowProc` function as part of that processing should call the `WinSetPointer` function to set the mouse pointer.

Note that windows not registered with the style `CS_HITTEST` do not receive this message.

See Also `WinDefWindowProc`, `WinSetPointer`, `WM_HITTEST`

■ WM_MOVE

WM_MOVE

The WM_MOVE message is sent when a window with `CS_MOVENOTIFY` style changes its absolute position or when a parent window of that window is moved. The window's new position can be obtained by calling the `WinQueryWindowRect` function.

Parameters This message does not use any parameters.

Return Value An application should return zero if it processes this message.

See Also `WinQueryWindowRect`

■ WM_NEXTMENU

```
WM_NEXTMENU
hwnd = HWNDFROMMP(mp1);          /* window handle */
fPrev = (BOOL) SHORT1FROMMP(mp2); /* previous flag */
```

The WM_NEXTMENU message is sent to the owner of a menu window to obtain the next or previous menu window. For example, this message is sent when either the beginning or the end of a menu has been reached when enumerating menus with the direction keys.

The WM_NEXTMENU message is processed by frame windows by toggling between the frame's main action-bar menu and the system menu. To implement a multiple-document interface, change this frame-window action to include the active child window's system menu in the menu enumeration set.

Parameters *hwnd* Low and high word of *mp1*. Identifies the current menu.

fPrev Low word of *mp2*. Specifies whether to go to the next or previous menu. When this parameter is TRUE, go to the previous menu. When it is FALSE, go to the next menu.

Return Value An application should return a handle to the next or previous menu. It should return NULL if a valid handle cannot be obtained.

■ WM_NULL

WM_NULL

The WM_NULL message is sent to activate message queues or modal loops. This message should be ignored.

- Parameters** This message does not use any parameters.
- Return Value** An application should return zero if it processes this message.
- See Also** WinTranslateAccel, WM_CHAR, WM_TRANSLATEACCEL

■ WM_OTHERWINDOWDESTROYED

```
WM_OTHERWINDOWDESTROYED
hwnd = HWNDFROMMP (mp1); /* destroyed window handle */
```

The WM_OTHERWINDOWDESTROYED message is sent to all child windows of the desktop when a window registered by the WinRegisterWindowDestroy function is being destroyed.

- Parameters** *hwnd* Low and high word of *mp1*. Identifies the window being destroyed.
- Return Value** An application should return zero if it processes this message.
- See Also** WinRegisterWindowDestroy, WM_DESTROY

■ WM_PAINT

WM_PAINT

The WM_PAINT message is sent when a window is to be repainted. An application can get a presentation space for drawing by calling WinBeginPaint. The presentation space will be clipped to the area of the window that is to be painted.

- Parameters** This message does not use any parameters.
- Return Value** An application should return zero if it processes this message.
- Example** This example shows how an application gets a presentation space for drawing by calling the WinBeginPaint function. When drawing is complete, the WinEndPaint function is called to release the presentation space.

```
case WM_PAINT:
    hps = WinBeginPaint(hwnd, NULL, &rcl);
        /* drawing routines would go here */

    WinEndPaint(hps);
    return (0L);
```

- See Also** WinBeginPaint, WinEndPaint, WM_ERASEBACKGROUND

■ WM_PAINTCLIPBOARD

```
WM_PAINTCLIPBOARD
hwndClip = HWNDFROMMP(mp1);    /* handle of clipboard viewer */
```

The WM_PAINTCLIPBOARD message is sent by an application to the current clipboard viewer when the clipboard's client area is to be repainted.

- Parameters** *hwndClip* Low and high word of *mp1*. Identifies the clipboard-viewer window.
- Return Value** An application should return zero if it processes this message.
- See Also** WM_DRAWCLIPBOARD

■ WM_QUERYACCELTABLE

```
WM_QUERYACCELTABLE
```

The WM_QUERYACCELTABLE message is sent to a frame window to get the handle of the accelerator table.

- Parameters** This message does not use any parameters.
- Return Value** An application should return the accelerator-table handle associated with the window. If no handle is available, the application should return NULL.
- See Also** WM_SETACCELTABLE

■ WM_QUERYBORDERSIZE

```
WM_QUERYBORDERSIZE
pptl = PVOIDFROMMP(mp1);    /* pointer to WPOINTL structure */
```

The WM_QUERYBORDERSIZE message is sent to a window to determine the size of its border. Typically, this message is sent to the frame window.

- Parameters** *pptl* Low and high word of *mp1*. Points to a POINTL structure that will contain the window border's width and height. The POINTL structure has the following form:

```
typedef struct _POINTL {
    LONG x;
    LONG y;
} POINTL;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

- Return Value** An application should place the size of the window borders into the POINTL structure pointed to by *pptl* and return TRUE for success or FALSE for failure.
- See Also** WM_SETBORDERSIZE

■ WM_QUERYCONVERTPOS

```
WM_QUERYCONVERTPOS
prclCurPos = (PRECTL) PVOIDFROMMP(mp1); /* pointer to RECTL */
```

The WM_QUERYCONVERTPOS message is sent by the Kanji conversion window in order to determine whether to begin conversion and where to position the conversion window.

Parameters

prclCurPos Low and high word of *mp1*. Points to a RECTL structure in which to place the cursor position. The RECTL structure has the following form:

```
typedef struct _RECTL {
    LONG xLeft;
    LONG yBottom;
    LONG xRight;
    LONG yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value

An application should return a conversion code. This code can be one of the following values:

Value	Meaning
QCP_CONVERT	Conversion may be performed for the window with the input focus. The RECTL structure pointed to by the <i>prclCurPos</i> parameter has been filled with a rectangle that describes where the text cursor is located and that will be used as a guide for positioning the conversion window. The rectangle must be given in screen coordinates.
QCP_NOCONVERT	No conversion should be performed. The window with the input focus cannot deal with DBCS characters. The RECTL structure pointed to by the <i>prclCurPos</i> parameter has not changed.

■ WM_QUERYDLGCODE

```
WM_QUERYDLGCODE
```

The WM_QUERYDLGCODE message is sent to a control window in a dialog box to determine the capabilities of the control.

Parameters

This message does not use any parameters.

Return Value

An application should return one or more of the following values combined into a single result by using the OR operator:

Value	Meaning
DLGC_BUTTON	Button item; processes the BM_CLICK message.
DLGC_CHECKBOX	Check-box button control.
DLGC_DEFAULT	Default push button.

Value	Meaning
DLGC_ENTRYFIELD	Entry-field item; processes the EM_SETSEL message.
DLGC_MENU	Menu.
DLGC_PUSHBUTTON	Normal (non-default) push button.
DLGC_RADIOBUTTON	Radio button.
DLGC_SCROLLBAR	Scroll bar.
DLGC_STATIC	Static item.
DLGC_TABONCLICK	The window should not get the focus. Instead, the focus should be passed on to the next-on-tab control.

See Also BM_CLICK, EM_SETSEL

■ WM_QUERYFOCUSCHAIN

```
WM_QUERYFOCUSCHAIN
fCmd = (BOOL) SHORT1FROMMP (mp1); /* command flag */
hwndFocus = HWNDFROMMP (mp2); /* focus-window ID */
```

The WM_QUERYFOCUSCHAIN message is sent to query the next window in the focus chain, to query the active window if the supplied window was given focus, or to query which window should be activated if the supplied window was selected (from Task Manager, or by pressing ALT+ESC).

Parameters

fCmd Low word of *mp1*. Specifies the action to be performed. This parameter can be one of the following values:

Value	Meaning
QFC_ACTIVE	Return the frame window that would be activated if that window were to be given the input focus (or conversely, the window that would be deactivated if the focus were taken away).
QFC_FRAME	Return the first frame window associated with the window.
QFC_NEXTINCHAIN	Return the next window in the focus chain.
QFC_SELECTACTIVE	Return the window that should be activated if the window receiving the message were selected from Task Manager or by ALT+ESC processing. For example, a disabled frame window that has an owned dialog box would return the window handle of the dialog box. This option is used by Task Manager and in similar situations to activate an application or owner group rather than a specific window. By default, the top-most enabled and visible window within the owner group should be returned.

hwndFocus Low and high word of *mp2*. Identifies the focus window.

Return Value An application should return a window handle passed on the type of action requested by the *fCmd* parameter.

See Also WM_FOCUSCHAIN

■ WM_QUERYFRAMECTLCOUNT

WM_QUERYFRAMECTLCOUNT

The WM_QUERYFRAMECTLCOUNT message is sent to determine the maximum number of frame controls that can exist in the frame window.

Parameters This message does not use any parameters.

Return Value An application should return the number of frame controls a frame window can have. This is usually (FID_CLIENT - FID_SYSMENU + 1).

Comments There is no limit to the number of controls that can be supported. Because it is used for memory allocation, it is critical that this message return a number that is no less than the actual number of controls in a frame.

See Also WM_FORMATFRAME

■ WM_QUERYFRAMEINFO

WM_QUERYFRAMEINFO

The WM_QUERYFRAMEINFO message is sent to a window to determine the following things about the window:

- Whether the window is a frame window.
- Whether the window should be hidden or shown as a result of its owner window being hidden, shown, minimized, or maximized.
- Whether the window can be activated (used by ALT+ESC enumeration code).
- Whether the window should move as a result of its owner being moved.

Parameters This message does not use any parameters.

Return Value An application should return a long word that contains frame-information flag bits that can be one or more of the following values:

Value	Meaning
FL_ACTIVATEOK	The window may be activated if it isn't disabled.
FL_FRAME	The window is a frame window.
FL_NOMOVEWITHOWNER	The window should not move as a result of its owner being moved.
FL_OWNERHIDE	The window should be hidden or shown as a result of its owner window being hidden, shown, minimized, or maximized.

See Also WM_QUERYWINDOWPARAMS

■ WM_QUERYICON

WM_QUERYICON

The WM_QUERYICON message is sent to a frame window to get a handle to the icon it uses to represent itself when minimized.

Parameters	This message does not use any parameters.
Return Value	An application should return an icon handle, or NULL if a handle is not available.
See Also	WM_SETICON

■ WM_QUERYTRACKINFO

```
WM_QUERYTRACKINFO
fTrack = (BOOL) SHORT1FROMMP(mp1); /* tracking flags */
pti = (PTRACKINFO) PVOIDFROMMP(mp2); /* pointer to TRACKINFO */
```

The WM_QUERYTRACKINFO message is sent to the window procedure of the owner of a title-bar control window at the start of track-move processing.

Parameters	<i>fTrack</i> Low word of <i>mp1</i> . Specifies tracking flags. This parameter can be one or more of the following flags:
-------------------	--

Value	Meaning
TF_LEFT	Track the left side of the rectangle.
TF_TOP	Track the top side of the rectangle.
TF_RIGHT	Track the right side of the rectangle.
TF_BOTTOM	Track the bottom side of the rectangle.
TF_MOVE	Track all sides of the rectangle.
TF_SETPOINTERPOS	Reposition the pointer according to the other options specified.
TF_FIXLEFT	Vertically center the pointer at the left of the tracking rectangle.
TF_FIXTOP	Horizontally center the pointer at the top of the tracking rectangle.
TF_FIXRIGHT	Vertically center the pointer at the right of the tracking rectangle.
TF_ALLINBOUNDARY	Perform tracking so that no part of the tracking rectangle ever falls outside the bounding rectangle.
TF_FIXBOTTOM	Horizontally center the pointer at the bottom of the tracking rectangle.
TF_GRID	Restrict tracking to the grid defined by the <i>cxGrid</i> and <i>cyGrid</i> fields.

Value	Meaning
TF_PARTINBOUNDARY	Track so that the tracking rectangle never falls outside the bounding rectangle.
TF_STANDARD	The width, height, grid width and grid height are all multiples of border width and border height.
TF_VALIDATETRACKRECT	Check the tracking rectangle against size and boundary limits and modify it to fit if necessary. No actual tracking takes place; return after validating.

pti Low and high word of *mp2*. Points to a TRACKINFO structure. The TRACKINFO structure has the following form:

```
typedef struct _TRACKINFO {
    SHORT  cxBorder;
    SHORT  cyBorder;
    SHORT  cxGrid;
    SHORT  cyGrid;
    SHORT  cxKeyboard;
    SHORT  cyKeyboard;
    RECTL  rclTrack;
    RECTL  rclBoundary;
    POINTL ptlMinTrackSize;
    POINTL ptlMaxTrackSize;
    USHORT fs;
    USHORT cxLeft;
    USHORT cyBottom;
    USHORT cxRight;
    USHORT cyTop;
} TRACKINFO;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

Return Value An application should return TRUE to continue sizing or moving. It should return FALSE to terminate sizing or moving.

See Also WM_QUERYTRACKINFO

■ WM_QUERYWINDOWPARAMS

```
WM_QUERYWINDOWPARAMS
pwprm = (WNDPARAMS) PVOIDFROMMP(mp1); /* pointer to WNDPARAMS */
```

The WM_QUERYWINDOWPARAMS message is sent to get various window parameters.

Parameters *pwprm* Low and high word of *mp1*. Points to a WNDPARAMS structure that defines the data to be returned. The window text, window-text length, control data, and control-data length are selectively returned according to the status flags set in WNDPARAMS. The WNDPARAMS structure has the following form:

```
typedef struct _WNDPARAMS {
    USHORT fsStatus;
    USHORT cchText;
    PSZ    pszText;
    USHORT cbPresParams;
    PVOID  pPresParams;
    USHORT cbCtlData;
    PVOID  pCtlData;
} WNDPARAMS;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value	An application should return TRUE if the operation is successful. Otherwise, it should return FALSE.
See Also	WM_QUERYFRAMEINFO, WM_SETWINDOWPARAMS

■ WM_QUIT

WM_QUIT

The WM_QUIT message is posted to terminate an application. The WinGetMsg function returns FALSE when it receives this message. A message-processing loop should terminate when WinGetMsg returns FALSE. For more information, see the description of the WinGetMsg function.

Parameters	This message does not have any parameters.
Return Value	This message does not have a return value because it causes the message loop to terminate before the message is sent to the application’s window procedure.

Example In this example, a WM_CLOSE message is received. If the *fChanges* flag is set, the application calls a function to determine if the user wants to save the changes before exiting. This function (called QuerySaveFile in this example) would ask the user if he or she wants to save the changes. If the user selects OK, it would save the changes. If the user selects Cancel, the function returns this value and the application continues normal execution. Otherwise, it posts a WM_QUIT message to terminate the application.

```
case WM_CLOSE:
    if (fChanges) {
        if (QuerySaveFile(hwnd) == MB_CANCEL) {
            return (OL); /* do not exit after all */
        }
    }
    WinPostMsg(hwnd, WM_QUIT, OL, OL);
    return (OL);
```

See Also	WinGetMsg, WinPostMsg
-----------------	-----------------------

■ WM_RENDERALLFMTS

WM_RENDERALLFMTS

The WM_RENDERALLFMTS message is sent to the clipboard owner when the owner application is being destroyed. The owner should render all formats that it is capable of generating and pass a handle or selector for each format to the clipboard by calling the WinSetClipboardData function. This ensures that the clipboard contains valid data even though the application that rendered the data is destroyed.

Parameters	This message does not have any parameters.
-------------------	--

Return Value An application should return zero if it processes this message.

See Also WinSetClipbrdData, WM_RENDERFMT

■ WM_RENDERFMT

```
WM_RENDERFMT
usFormat = SHORT1FROMMP(mp1); /* format of data */
```

The WM_RENDERFMT message is sent to the clipboard owner when a particular format with delayed rendering needs to be rendered. The receiver should render the data in that format and pass it to the clipboard by calling the WinSetClipbrdData function.

Parameters *usFormat* Low word of *mp1*. Specifies the format of the data to be rendered. It can be an application-specific format, or one of the following standard formats:

Value	Meaning
CF_BITMAP	Specifies a bitmap.
CF_DSPBITMAP	Specifies a bitmap representation of a private data format.
CF_DSPMETAFILE	Specifies a metafile representation of a private data format.
CF_DSPTEXT	Specifies a textual representation of a private data format.
CF_METAFILE	Specifies a metafile.
CF_TEXT	Specifies an array of text characters.

Return Value An application should return zero if it processes this message.

See Also WinSetClipbrdData, WM_RENDERALLFMTS

■ WM_SEM1

```
WM_SEM1
flFlags = LONGFROMMP(mp1); /* application-defined flags */
```

The WM_SEM1 message is one of four WM_SEM messages that an application can use to send signals within and between threads. WM_SEM messages can be used much like an MS OS/2 semaphore. Unlike an MS OS/2 semaphore, however, a thread waiting for a WM_SEM message can continue to process other messages instead of blocking until the WM_SEM message is received.

A WM_SEM1 message should be posted, not sent, by using the WinPostMsg function.

Parameters *flFlags* Specifies a 32-bit field that is combined with any previous WM_SEM1 messages that have not been retrieved from the message queue by using the OR operator. The application determines how this parameter is to be used.

- Return Value** An application should return zero if it processes this message.
- Comments** A WM_SEM1 message is quite fast, having a higher priority than other messages. The following list shows the message priority of the WM_SEM1 message in relation to other messages (from highest to lowest):
- ```
WM_SEM1
Any message posted using the WinPostMsg function or any input message
not listed here.
WM_SEM2
WM_TIMER
WM_SEM3
WM_PAINT
WM_SEM4
```
- WM\_SEM1 messages are handled differently by the system than other messages. They can be used even if a message queue is full. When a WM\_SEM1 message is posted, the system combines each message queue's WM\_SEM1 messages into one message by combining the *flFlags* parameter with the message queue's previous WM\_SEM1 *flFlags* parameter. The *flFlags* parameter is cleared whenever the WM\_SEM1 message is retrieved.
- The application must determine how the *flFlags* field is to be used. An application might set flag bits to indicate certain actions to be taken by the receiver of the WM\_SEM1 message, or it might set a bit to indicate who is actually posting the WM\_SEM1 message.
- Example** In this example, a thread notifies the client window that the thread is about to terminate. It sends the constant THREAD3 as the *flFlags* parameter so that when the client window receives the message, it can tell which thread terminated.
- ```
#define THREAD1 1      /* bit #1 */
#define THREAD2 2      /* bit #2 */
#define THREAD3 4      /* bit #3 */
VOID FAR Thread3() {
    .
    .
    WinPostMsg(hwndClient, WM_SEM1, (MPARAM) THREAD3, 0);
    DosExit(EXIT_THREAD, 0);
}
```
- See Also** WinPostMsg

■ WM_SEM2

```
WM_SEM2
flFlags = LONGFROMMP(mp1);      /* application-defined flags */
```

The WM_SEM2 message is one of four WM_SEM messages. It is identical to the WM_SEM1 message except in priority. For more information, see the description of the WM_SEM1 message.

■ WM_SEM3

```
WM_SEM3
flFlags = LONGFROMMP(mp1);    /* application-defined flags */
```

The WM_SEM3 message is one of four WM_SEM messages. It is identical to the WM_SEM1 message except in priority. For more information, see the description of the WM_SEM1 message.

■ WM_SEM4

```
WM_SEM4
flFlags = LONGFROMMP(mp1);    /* application-defined flags */
```

The WM_SEM4 message is one of four WM_SEM messages. It is identical to the WM_SEM1 message except in priority. For more information, see the description of the WM_SEM1 message.

■ WM_SETACCELTABLE

```
WM_SETACCELTABLE
haccel = (HACCEL) HWNDFROMMP(mp1);    /* handle of accelerator table */
```

The WM_SETACCELTABLE message is sent to a frame window to set the handle of the accelerator table.

Parameters *haccel* Low and high word of *mp1*. Identifies the accelerator table.

Return Value An application should return zero if it processes this message.

See Also WM_QUERYACCELTABLE

■ WM_SETBORDERSIZE

```
WM_SETBORDERSIZE
cx = SHORT1FROMMP(mp1);    /* width */
cy = SHORT1FROMMP(mp2);    /* height */
```

The WM_SETBORDERSIZE message is sent to a frame window to change its size-control's border width and height.

Parameters *cx* Low word of *mp1*. Specifies the width of the size control.

cy Low word of *mp2*. Specifies the height of the size control.

Return Value An application should return TRUE if the size control is set. Otherwise, it should return FALSE.

See Also WM_QUERYBORDERSIZE

■ WM_SETFOCUS

```
WM_SETFOCUS
hwnd = HWNDFROMMP (mp1);          /* window ID */
fFocus = (BOOL) SHORT1FROMMP (mp2); /* focus flag */
```

The WM_SETFOCUS message is sent when a window is to receive or lose the input focus.

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

The default window procedure takes no action on this message.

Parameters

hwnd Low and high word of *mp1*. Identifies the window gaining or losing the focus. This parameter is NULL if no window previously had the focus.

fFocus Low word of *mp2*. Specifies whether the window is receiving or losing the focus. If this parameter is TRUE, the window is receiving the focus. If it is FALSE, the window is losing the focus.

Return Value

An application should return zero if it processes this message.

See Also

WM_ACTIVATE, WM_FOCUSCHANGE

■ WM_SETICON

```
WM_SETICON
hptrIcon = (HPOINTER) LONGFROMMP (mp1); /* handle to icon */
```

The WM_SETICON message is sent to a frame window to set the icon it uses to represent itself when minimized.

Parameters

hptrIcon Low and high word of *mp1*. Identifies an icon.

Return Value

An application should return TRUE if the associated icon is set or FALSE if it is not.

See Also

WM_QUERYICON

■ WM_SETSELECTION

```
WM_SETSELECTION
fSelect = (BOOL) SHORT1FROMMP (mp1); /* TRUE for selection */
```

The WM_SETSELECTION message is sent to a window when it is selected or deselected.

Parameters

fSelect Low word of *mp1*. Specifies whether the window is being selected or deselected. If this parameter is TRUE, the window is being selected. If it is FALSE, the window is being deselected.

Return Value An application should return zero if it processes this message.

See Also WM_ACTIVATE, WM_FOCUSCHANGE, WM_SETFOCUS

■ WM_SETWINDOWPARAMS

```
WM_SETWINDOWPARAMS
pwprm = (PWNDPARAMS) PVOIDFROMMP (mp1);    /* pointer to WNDPARAMS */
```

The WM_SETWINDOWPARAMS message is sent when an application sets or changes the window parameters.

If this message is sent to a window of another process, the WNDPARAMS structure pointed to by *pwprm* must be in memory shared by both processes.

Parameters *pwprm* Low and high word of *mp1*. Points to a WNDPARAMS structure that defines the data to be set. The window text and control data are selectively set according to the status flags set in WNDPARAMS. The WNDPARAMS structure has the following form:

```
typedef struct _WNDPARAMS {
    USHORT fsStatus;
    USHORT cchText;
    PSZ    pszText;
    USHORT cbPresParams;
    PVOID  pPresParams;
    USHORT cbCtlData;
    PVOID  pCtlData;
} WNDPARAMS;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

Return Value An application should return TRUE if the operation is successful. Otherwise, it should return FALSE.

See Also WM_QUERYWINDOWPARAMS

■ WM_SHOW

```
WM_SHOW
fShow = (BOOL) SHORT1FROMMP (mp1);    /* show flag */
```

The WM_SHOW message is sent when the visible state of a window changes (controlled by the WS_VISIBLE style bit).

The visible state of a window can be changed by calling the WinShowWindow function. This state is not affected by the movement of other windows which may change the visible region of a window.

Parameters *fShow* Low word of *mp1*. Specifies whether the visible state of the window is shown or hidden. If this parameter is TRUE, the window is being shown. If it is FALSE, the window is being hidden.

Return Value An application should return zero if it processes this message.

See Also WinShowWindow

■ WM_SIZE

```
WM_SIZE
cxOld = SHORT1FROMMP (mp1); /* old width */
cyOld = SHORT2FROMMP (mp1); /* old height */
cxNew = SHORT1FROMMP (mp2); /* new width */
cyNew = SHORT2FROMMP (mp2); /* new height */
```

The WM_SIZE message is sent when a window changes its size. It is sent after the window has been sized, but before any repainting has been performed. Any resizing or repositioning of child windows that may be necessary as a result of the size change is usually performed during the processing of this message. The application should not send any output to the window during the processing of the WM_SIZE message because the area drawn into may be drawn a second time after the WM_SIZE processing is complete.

This message is not sent when the window is created.

The processing of this message for a window displaying an advanced video-input-and-output (AVIO) presentation space must be carried out by the default AVIO window procedure.

- Parameters**
- cxOld* Low word of *mp1*. Specifies the old width.
 - cyOld* High word of *mp1*. Specifies the old height.
 - cxNew* Low word of *mp2*. Specifies the new width.
 - cyNew* High word of *mp2*. Specifies the new height.
- Return Value** An application should return zero if it processes this message.
- See Also** WinCreateWindow, WinSetWindowPos, WM_CREATE

■ WM_SIZECLIPBOARD

```
WM_SIZECLIPBOARD
hwnd = HWNDFROMMP (mp1); /* clipboard-viewer handle */
prcl = (PRECTL) PVOIDFROMMP (mp2); /* pointer to RECTL structure */
```

The WM_SIZECLIPBOARD message is sent by the clipboard viewer to the clipboard owner when the clipboard contains data with the attribute CFL_OWNERDISPLAY attribute and the clipboard-viewer window has changed size. When the clipboard viewer is being destroyed or made iconic, this message is sent with the rectangle size equal to (0,0,0,0), which permits the owner to free its display resources.

- Parameters**
- hwnd* Low and high word of *mp1*. Identifies the clipboard viewer.
 - prcl* Low and high word of *mp2*. Points to a RECTL structure that contains the rectangle of the area that is to be repainted. The RECTL structure has the following form:

```
typedef struct _RECTL {
    LONG   xLeft;
    LONG   yBottom;
    LONG   xRight;
    LONG   yTop;
} RECTL;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

Return Value An application should return zero if it processes this message.

■ WM_SUBSTITUTESTRING

```
WM_SUBSTITUTESTRING
index = SHORT1FROMMP(mp1); /* index of substitution string */
```

The WM_SUBSTITUTESTRING message is sent as part of the processing of the WinSubstituteStrings function. It allows an application to substitute phrases within a string.

Parameters *index* Low word of *mp1*. Specifies a value that is equal to the numeric value in the substitution string. It can be any value between 0 and 9.

Return Value An application should return a far pointer to a substitution string or NULL to indicate no string.

See Also WinSubstituteStrings

■ WM_SYSCOLORCHANGE

```
WM_SYSCOLORCHANGE
```

The WM_SYSCOLORCHANGE message is sent to all child windows of the desktop when a change is made to the system colors by the WinSetSysColors function. When the WM_SYSCOLORCHANGE message is received, applications that depend on the system colors can query the new color values using the WinQuerySysColors function.

After the WM_SYSCOLORCHANGE messages are sent, all windows in the system are invalidated so that they will be redrawn with the new system colors.

The default window procedure takes no action on this message.

Parameters *mp1* Specifies a reserved value. It must be NULL.

mp2 Specifies a reserved value. It must be NULL.

Return Value The return value is NULL.

See Also WinQuerySysColors, WinSetSysColors, WM_SYSVALUECHANGED

■ WM_SYSCOMMAND

```
WM_SYSCOMMAND
usCmd = (USHORT) SHORT1FROMMP (mp1);    /* command value */
fsSource = (USHORT) SHORT1FROMMP (mp2); /* source type */
fPointer = (BOOL) SHORT2FROMMP (mp2);   /* pointer flag */
```

The WM_SYSCOMMAND message is sent when a control window has a command to report to its owner or when a keystroke has been translated by an accelerator table into a WM_SYSCOMMAND message. Typically, a WM_SYSCOMMAND message is sent when the user selects an item from the system menu.

Parameters

usCmd Low word of *mp1*. Specifies the command value.

fsSource Low word of *mp2*. Specifies the source type. This parameter can be one of the following values:

Value	Meaning
CMDSRC_ACCELERATOR	Posted as the result of an accelerator. The <i>usCmd</i> parameter is the accelerator command value.
CMDSRC_MENU	Posted by a menu control. The <i>usCmd</i> parameter is the identifier of the menu item.
CMDSRC_PUSHBUTTON	Posted by a push-button control. The <i>usCmd</i> parameter is the window identifier of the push button.
CMDSRC_OTHER	Other source. The <i>usCmd</i> parameter gives further control-specific information defined for each control type.

fPointer High word of *mp2*. Specifies whether the message was posted as a result of a pointing-device (mouse) operation. A value of TRUE indicates a pointing device was used; FALSE indicates a keyboard operation.

Return Value

An application should return zero if it processes this message.

See Also

WM_COMMAND, WM_MENUSELECT, WM_TRANSLATEACCEL

■ WM_SYSVALUECHANGED

```
WM_SYSVALUECHANGED
iFirst = SHORT1FROMMP (mp1);    /* first value that changed */
iLast = SHORT1FROMMP (mp2);    /* last value that changed */
```

The WM_SYSVALUECHANGED message is sent to all child windows of the desktop when a change is made to a system value. The application should post this message whenever it changes the system values to notify other windows of the change.

Parameters

iFirst Low word of *mp1*. Specifies the first of a contiguous set of changed system values.

iLast Low word of *mp2*. Specifies the last of a contiguous set of changed system values.

Return Value An application should return zero if it processes this message.

See Also WM_SYSCOLORCHANGE

■ WM_TIMER

```
WM_TIMER
idTimer = SHORT1FROMMP(mp1);    /* timer ID */
```

The WM_TIMER message is sent after each interval specified in the WinStartTimer function that was used to start a timer.

Parameters *idTimer* Low word of *mp1*. Specifies the timer.

Return Value An application should return zero if it processes this message.

See Also WinStartTimer

■ WM_TRACKFRAME

```
WM_TRACKFRAME
fsTrackFlags = SHORT1FROMMP(mp1);    /* tracking flags */
```

The WM_TRACKFRAME message is sent to start the tracking operation for a frame window.

Parameters *fsTrackFlags* Low word of *mp1*. Specifies tracking flags. This parameter can be one or more of the following flags:

Value	Meaning
TF_LEFT	Track the left side of the rectangle.
TF_TOP	Track the top side of the rectangle.
TF_RIGHT	Track the right side of the rectangle.
TF_BOTTOM	Track the bottom side of the rectangle.
TF_MOVE	Track all sides of the rectangle.
TF_SETPOINTERPOS	Reposition the pointer according to the other options specified.
TF_FIXLEFT	Vertically center the pointer at the left of the tracking rectangle.
TF_FIXTOP	Horizontally center the pointer at the top of the tracking rectangle.
TF_FIXRIGHT	Vertically center the pointer at the right of the tracking rectangle.
TF_ALLINBOUNDARY	Perform tracking so that no part of the tracking rectangle ever falls outside the bounding rectangle.
TF_FIXBOTTOM	Horizontally center the pointer at the bottom of the tracking rectangle.

Value	Meaning
TF_GRID	Restrict tracking to the grid defined by the <code>cxGrid</code> and <code>cyGrid</code> fields.
TF_PARTINBOUNDARY	Perform tracking so that all the tracking rectangle never falls outside the bounding rectangle.
TF_STANDARD	The width, height, grid width and grid height are all multiples of border width and border height.
TF_VALIDATETRACKRECT	Check the tracking rectangle against size and boundary limits and modify it to fit if necessary. No actual tracking takes place; return after validating.

Return Value An application should return TRUE if the tracking operation was successful, or FALSE if it was not.

See Also WM_QUERYTRACKINFO

■ WM_TRANSLATEACCEL

```
WM_TRANSLATEACCEL
pqmsg = (PQMSG) PVOIDFROMMP(mp1); /* pointer to QMSG structure */
```

The WM_TRANSLATEACCEL message is sent to the focus window whenever a WM_CHAR message is obtained, to allow for any accelerator translation of the WM_CHAR message. The default window procedure handles this message by calling the WinTranslateAccel function.

Parameters *pqmsg* Low and high word of *mp1*. Points to a QMSG structure that contains a queue message. The QMSG structure has the following form:

```
typedef struct _QMSG {
    HWND    hwnd;
    USHORT  msg;
    MPARAM  mp1;
    MPARAM  mp2;
    ULONG   time;
    POINTL  pt1;
} QMSG;
```

For a full description, see Chapter 4, "Types, Macros, Structures."

Return Value An application should return TRUE if the queue message has been translated. Otherwise, it should return FALSE.

See Also WinTranslateAccel, WM_CHAR, WM_COMMAND, WM_HELP, WM_NULL, WM_SYSCOMMAND

■ WM_UPDATEFRAME

```
WM_UPDATEFRAME
fsStyle = SHORT1FROMMP(mp1); /* style bits */
```

The WM_UPDATEFRAME message is sent after frame controls have been added or removed from the window frame. It tells the frame window to reformat and update the appearance of the window. An application should send this message to its frame window whenever it adds or removes frame controls.

Parameters

fsStyle Low word of *mp1*. Specifies frame-style bits that indicate which frame controls were added or removed. This parameter can be any one of the following values:

Value	Meaning
FCF_TITLEBAR	Title bar added or removed.
FCF_SYSMENU	System menu added or removed.
FCF_MENU	Menu added or removed.
FCF_SIZEBORDER	Sizing border added or removed.
FCF_MINBUTTON	Minimize button added or removed.
FCF_MAXBUTTON	Maximize button added or removed.
FCF_MINMAX	Minimize/maximize button added or removed.
FCF_VERTSCROLL	Vertical scroll bar added or removed.
FCF_HORZSCROLL	Horizontal scroll bar added or removed.

Return Value

An application should return TRUE if it processes this message.

Comments

Because this message causes any redrawing that is necessary, the application should ensure that no drawing takes place when adding or removing a frame control, to prevent unnecessary redrawing. If using the WinSetParent function, this is done by setting the *fRedraw* parameter to FALSE.

See Also

WinSetParent, WM_FORMATFRAME

■ WM_VIOCHAR

```
WM_VIOCHAR
fsKeyFlags = (USHORT) SHORT1FROMMP(mp1); /* key flags */
uchRepeat = (UCHAR) CHAR3FROMMP(mp1); /* repeat count */
uchScanCode = (UCHAR) CHAR4FROMMP(mp1); /* scan code */
uchChr = (UCHAR) CHAR1FROMMP(mp2); /* character */
uchKbdScan = (UCHAR) CHAR2FROMMP(mp2); /* virtual key */
```

The WM_VIOCHAR message is sent whenever the user presses a key. This message is placed in the queue associated with the window that has the focus.

Parameters

fsKeyFlags Low word of *mp1*. Specifies the keyboard control codes. This parameter may be one or more of the following values:

Value	Meaning
KC_CHAR	Indicates that the <i>uchChr</i> parameter contains a valid character. This bit will be set only on a down stroke. KC_CHAR is not set if either the KC_ALT or the KC_CONTROL flag is set, even if <i>uchChr</i> contains a valid character.
KC_SCANCODE	Indicates the <i>uchScanCode</i> value is valid; otherwise, <i>uchScanCode</i> contains zero. Generally, all WM_CHAR messages generated from actual user input have the KC_SCANCODE flag set. However, if the message has been generated by an application that has issued the <i>WinSetHook</i> function to filter keystrokes, or been posted to the application queue, this code may not be set.
KC_VIRTUALKEY	Indicates the <i>usVKey</i> value is valid; otherwise, <i>usVKey</i> contains zero.
KC_KEYUP	The event was a key-up transition; otherwise, it was a key-down transition.
KC_PREVDOWN	The key was previously down; otherwise, it was previously up.
KC_DEADKEY	The character code is a dead key. The application displays the glyph for the dead key without advancing the cursor.
KC_COMPOSITE	The character code was formed by combining the current key with the previous dead key.
KC_INVALIDCOMP	The character code was not a valid combination with the preceding dead key. The application advances the cursor past the dead-key glyph and then, if the current character is <i>not</i> a space, it beeps the speaker and displays the new character code.
KC_LONEKEY	This bit is set if the key was pressed and released without any other keys being pressed or released between the time the key went down and went up.
KC_SHIFT	The SHIFT state was active when the key was pressed or released.
KC_ALT	The ALT state was active when the key was pressed or released.
KC_CTRL	The CONTROL state was active when the key was pressed or released.

uchRepeat Low byte of the high word of *mp1*. Specifies the repeat count of the key.

uchScanCode High byte of the high word of *mp1*. Specifies the scan code of the character.

uchChr Low word of *mp2*. Specifies the ASCII character.

uchKbdScan High byte of low word of *mp2*. Specifies the keyboard scan code.

Return Value The application should return TRUE if it processes the message. Otherwise, it should return FALSE.

See Also WinSetHook, CHARMSG, WM_CHAR

■ WM_VSCROLL

```
WM_VSCROLL
id = SHORT1FROMMP(mp1);      /* control-window ID */
sPos = SHORT1FROMMP(mp2);    /* slider position */
usCmd = SHORT2FROMMP(mp2);   /* command */
```

The WM_VSCROLL message is posted to the owner of a vertical scroll-bar window when an event occurs.

Parameters

id Low word of *mp1*. Identifies the scroll-bar window.

sPos Low word of *mp2*. When *usCmd* is SB_SLIDERPOSITION or SB_SLIDERTRACK, it specifies the slider position. If *usCmd* is SB_ENDSCROLL, *sPos* is TRUE if the pointer was inside the scroll bar or FALSE if the pointer was outside. The *sPos* parameter is zero for all other *usCmd* values.

usCmd High word of *mp2*. Specifies the type of command. This parameter can be one of the following values:

Value	Meaning
SB_LINEUP	The user clicked the scroll-bar up arrow or pressed the VK_UP key.
SB_LINEDOWN	The user clicked the scroll-bar down arrow or pressed the VK_DOWN key.
SB_PAGEUP	The user clicked the area above the slider or pressed the VK_PAGEUP key.
SB_PAGEDOWN	The user clicked the area below the slider or pressed the VK_PAGEDOWN key.
SB_SLIDERPOSITION	The <i>sPos</i> parameter contains the final position of the slider.
SB_SLIDERTRACK	The user is dragging the slider. This value is sent whenever the slider position changes.
SB_ENDSCROLL	The user has finished scrolling. This value is set only if the user was not doing an absolute slider movement.

Return Value An application should return zero if it processes this message.

See Also WM_HSCROLL, WM_VSCROLLCLIPBOARD

■ WM_VSCROLLCLIPBOARD

```
WM_VSCROLLCLIPBOARD
hwndClip = HWNDEROMMP (mp1);    /* handle of clipboard owner */
sPos = SHORT1FROMMP (mp2);      /* slider position */
usCmd = SHORT2FROMMP (mp2);     /* command */
```

The WM_VSCROLLCLIPBOARD message is sent by the clipboard viewer to the clipboard owner when the clipboard data has the CF_OWNERDRAW attribute and there is an event in the clipboard viewer's vertical scroll bar. The owner should scroll the clipboard image, invalidate the appropriate sections, and update the scroll bar values.

Parameters

hwndClip Low and high word of *mp1*. Identifies the clipboard viewer.

sPos Low word of *mp2*. Specifies the slider position.

usCmd High word of *mp2*. Specifies the type of command. This parameter can be one of the following values:

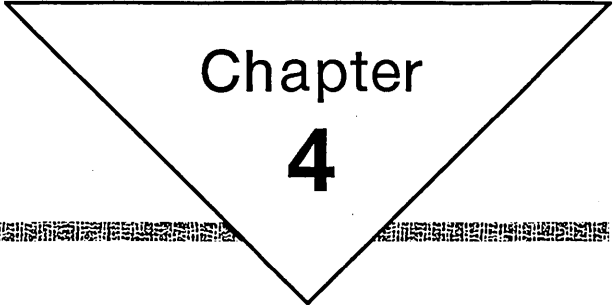
Value	Meaning
SB_LINEUP	The user clicked the scroll-bar up arrow or pressed the VK_UP key.
SB_LINEDOWN	The user clicked the scroll-bar down arrow or pressed the VK_DOWN key.
SB_PAGEUP	The user clicked the area above the slider or pressed the VK_PAGEUP key.
SB_PAGEDOWN	The user clicked the area below the slider or pressed the VK_PAGEDOWN key.
SB_SLIDERPOSITION	The <i>sPos</i> parameter contains the final position of the slider.
SB_SLIDERTRACK	The user is dragging the slider. This value is sent whenever the slider position changes.
SB_ENDSCROLL	The user has finished scrolling. This value is set only if the user was not doing an absolute slider movement.

Return Value

An application should return zero if it processes this message.

See Also

WM_HSCROLLCLIPBOARD, WM_VSCROLL



Chapter
4

Types, Macros, Structures

4.1	Introduction	471
4.2	Types	472
4.3	Macros	473
4.4	Structures	485

4.1 Introduction

This chapter describes the types, macros, and structures used with MS OS/2 Dev, Gpi, and Win functions. MS OS/2 functions use many types, macros, and structures that are not part of the standard C language. These types, macros, and structures have been defined to make the task of creating MS OS/2 programs easier and to make program sources clearer and easier to understand.

All types, macros, and structures in this manual are defined in the MS OS/2 C-language include files. Programmers may also wish to use these when developing MS OS/2 programs in other computer languages, such as Pascal or assembly language. If include files for a given language are not available, a programmer can translate the definitions given in this chapter by following these guidelines:

- Numbers must be integers or fixed-point real numbers. MS OS/2 functions do not support floating-point numbers. An MS OS/2 program can use floating-point numbers if an appropriate run-time library or coprocessor is supplied and if floating-point numbers are not used as parameters to the MS OS/2 functions.
- Structures must be packed. Some compilers align each new field in a structure on word or double-word boundaries. This may leave unused bytes in a structure if a given field is smaller than the width between boundaries. MS OS/2 functions require that unused bytes be removed from structures.
- Reserved fields in structures should be set to zero. Unless otherwise specified, MS OS/2 functions require that reserved fields be set to zero to avoid compatibility problems with future releases of MS OS/2.
- Variable-length structures must be supported. Several MS OS/2 functions use variable-length structures to receive and/or return information. In a variable-length structure, the number of fields varies depending on when the structure is used. In the C language, programs typically support variable-length structures by allocating enough memory for the current number of fields and accessing those fields by using a pointer to the structure. Programs in other languages may use this method or devise their own method for supporting variable-length structures.
- All 16-bit pointers must be relative to an explicitly defined segment register. Some compilers assume that the `ds` and `ss` registers contain the same value and implicitly use one segment for both. MS OS/2 does not guarantee that the `ds` and `ss` registers will be equal. This is especially true in dynamic-link libraries and programs that use callback functions (for example, window procedures).
- All 32-bit pointers must consist of a *selector:offset* pair. A physical address, that is, an address that represents a 32-bit offset from the beginning of physical memory, cannot be used by MS OS/2 functions.

4.2 Types

The following is a complete list, in alphabetical order, of the types that have been defined for the functions described in this manual. Many of these types begin with a letter that identifies what the type is used for—for example, **H** identifies a handle, **P** identifies a far pointer.

Type	Meaning
ATOM	16-bit value used as an atom handle.
COLOR	32-bit signed value used as a color value.
FFDESCS	Two-dimensional array used for font information.
FIXED	32-bit fixed-point real number.
HAB	32-bit value used as an anchor-block handle.
HACCEL	32-bit value used as an accelerator-table handle.
HATOMTBL	32-bit value used as an atom-table handle.
HBITMAP	32-bit value used as a bitmap handle.
HDC	32-bit value used as a device-context handle.
HENUM	32-bit value used as enumeration-list handle.
HHEAP	32-bit value used as a heap handle.
HMF	32-bit value used as a metafile handle.
HMQ	32-bit value used as a message-queue handle.
HPOINTER	32-bit value used as a pointer handle.
HPROGRAM	32-bit value used as a program or group handle.
HPS	32-bit value used as a presentation-space handle.
HRGN	32-bit value used as a region handle.
HSWITCH	32-bit value used as a switch-entry handle.
HVPS	16-bit value used as an advanced video-input-and-output (AVIO) presentation-space handle.
HWND	32-bit value used as a window handle.
MPARAM	32-bit pointer to an unspecified data type.
MRESULT	32-bit pointer to an unspecified data type.
PCOLOR	32-bit pointer to a color value.
PFDESCS	32-bit pointer to an array of font information.
PFIXED	32-bit pointer to a fixed-point real number.
PHAB	32-bit pointer to an anchor-block handle.
PHBITMAP	32-bit pointer to a bitmap handle.
PHDC	32-bit pointer to a device-context handle.

Type	Meaning
PHMF	32-bit pointer to a metafile handle.
PHPROGRAM	32-bit pointer to a program or group handle.
PHPS	32-bit pointer to a presentation-space handle.
PHRGN	32-bit pointer to a region handle.
PHSWITCH	32-bit pointer to a switch-entry handle.
PHVPS	32-bit pointer to an advanced video-input-and-output (AVIO) presentation-space handle.
PMPARAM	32-bit pointer to a message-result pointer.
PMRESULT	32-bit pointer to a message-result pointer.
PROGCATEGORY	8-bit value used as a program category.
PSTR16	32-bit pointer to a 16-character array.
PSTR32	32-bit pointer to a 32-character array.
PSTR64	32-bit pointer to a 64-character array.
STR16	16-character array.
STR32	32-character array.
STR64	64-character array.

4.3 Macros

The following is a complete list, in alphabetical order, of the macros that can be used with the functions described in this manual.

■ CHAR1FROMMP

CHAR1FROMMP(*mp*)

The **CHAR1FROMMP** macro retrieves the character in the low byte of a message parameter.

Parameters *mp* Specifies the message parameter.

See Also CHAR2FROMMP, CHAR3FROMMP, CHAR4FROMMP, CHARMSG

■ CHAR2FROMMP

CHAR2FROMMP(*mp*)

The **CHAR2FROMMP** macro retrieves the character in the high byte of the low word of a message parameter.

Parameters *mp* Specifies the message parameter.

See Also CHAR1FROMMP, CHAR3FROMMP, CHAR4FROMMP, CHARMSG

■ CHAR3FROMMP

CHAR3FROMMP(*mp*)

The **CHAR3FROMMP** macro retrieves the character in the low byte of the high word of a message parameter.

Parameters *mp* Specifies the message parameter.

See Also CHAR1FROMMP, CHAR2FROMMP, CHAR4FROMMP, CHARMSG

■ CHAR4FROMMP

CHAR4FROMMP(*mp*)

The **CHAR4FROMMP** macro retrieves the character in the high byte of the high word of a message parameter.

Parameters *mp* Specifies the message parameter.

See Also CHAR1FROMMP, CHAR2FROMMP, CHAR3FROMMP, CHARMSG

■ CHARMSG

CHARMSG(*pmsg*)

The **CHARMSG** macro is used to access the **WM_CHAR** message parameters. This macro defines a **CHARMSG** structure pointer, which has the following form:

```
struct _CHARMSG {
    USHORT chr;           /* mp2 */
    USHORT vkey;
    USHORT fs;           /* mp1 */
    UCHAR cRepeat;
    UCHAR scancode;
};
```

Parameters *pmsg* Points to the message received by the application's window procedure.

Example This example uses the **CHARMSG** macro to process a **WM_CHAR** message. First, it uses the macro to determine whether a key was released, then it uses the macro to generate a **switch** statement based on the character received.

```
MRESULT CALLBACK GenericWndProc(hwnd, usMessage, mp1, mp2)
HWND hwnd;
USHORT usMessage;
MPARAM mp1;
MPARAM mp2;
{
    switch (usMessage) {
    case WM_CHAR:
        if (CHARMSG(&usMessage)->fs & KC_KEYUP) {
            switch (CHARMSG(&usMessage)->chr) {
```

See Also **CHAR1FROMMP, CHAR2FROMMP, CHAR3FROMMP, CHAR4FROMMP**

■ COMMANDMSG

COMMANDMSG(*pmsg*)

The **COMMANDMSG** macro retrieves information that is passed in the parameters of the **WM_COMMAND**, **WM_HELP**, and **WM_SYSCOMMAND** messages. This macro defines a **COMMANDMSG** structure pointer, which has the following form:

```
struct _COMMANDMSG {
    USHORT source;       /* mp2 */
    BOOL fMouse;
    USHORT cmd;         /* mp1 */
    USHORT unused;
};
```

Parameters *pmsg* Points to the message received by the application's window procedure.

See Also **WM_COMMAND, WM_HELP, WM_SYSCOMMAND**

■ DDES_PABDATA

DDES_PABDATA(*pdde*)

The DDES_PABDATA macro retrieves a far (32-bit) pointer to the data area that follows a DDESTRUCT structure. The following messages pass a DDESTRUCT structure as the second parameter:

WM_DDE_REQUEST
WM_DDE_ACK
WM_DDE_DATA
WM_DDE_ADVISE
WM_DDE_UNADVISE
WM_DDE_POKE
WM_DDE_EXECUTE

Parameters *pdde* Points to the DDESTRUCT structure that precedes the data area.

See Also DDES_PSZITEMNAME

■ DDES_PSZITEMNAME

DDES_PSZITEMNAME(*pdde*)

The DDES_PSZITEMNAME macro retrieves a far (32-bit) pointer to the item name contained within a DDESTRUCT structure. The following messages pass a DDESTRUCT structure as the second parameter:

WM_DDE_REQUEST
WM_DDE_ACK
WM_DDE_DATA
WM_DDE_ADVISE
WM_DDE_UNADVISE
WM_DDE_POKE
WM_DDE_EXECUTE

Parameters *pdde* Points to the DDESTRUCT structure that contains the item name.

See Also DDES_PABDATA

■ ERRORIDERROR

ERRORIDERROR(*errid*)

The ERRORIDERROR macro retrieves the error number from an ERRORID value. An ERRORID value is returned by the WinGetLastError function.

Parameters *errid* Specifies the error identifier.

See Also ERRORIDSEV, MAKEERRORID

■ ERRORIDSEV

ERRORIDSEV(*errid*)

The **ERRORIDSEV** macro retrieves the severity value from an **ERRORID** value. An **ERRORID** value is returned by the **WinGetLastError** function. The severity value may be one of the following:

SEVERITY_NOERROR
 SEVERITY_WARNING
 SEVERITY_ERROR
 SEVERITY_SEVERE
 SEVERITY_UNRECOVERABLE

Parameters *errid* Specifies the error identifier.

See Also **ERRORIDERROR, MAKEERRORID**

■ FIXEDFRAC

FIXEDFRAC(*fx*)

The **FIXEDFRAC** macro retrieves the fractional part of a **FIXED** value.

Parameters *fx* Specifies the **FIXED** value.

See Also **FIXEDINT, MAKEFIXED**

■ FIXEDINT

FIXEDINT(*fx*)

The **FIXEDINT** macro retrieves the integer part of a **FIXED** value.

Parameters *fx* Specifies the **FIXED** value.

See Also **FIXEDFRAC, MAKEFIXED**

■ HWNDFROMMP

HWNDFROMMP(*mp*)

The **HWNDFROMMP** macros casts a message parameter to an **HWND** type.

Parameter *mp* Specifies the message parameter.

See Also **MPFROMHWND**

LONGFROMMP

LONGFROMMP(*mp*)

The **LONGFROMMP** macro casts a message parameter to a **LONG** type.

Parameter *mp* Specifies the message parameter.

See Also **MPFROMLONG**

LONGFROMMR

LONGFROMMR(*mr*)

The **LONGFROMMR** macro casts a message result to a **LONG** type.

Parameter *mp* Specifies the message result. The following functions return this parameter:

WinDdeRespond
WinDefAVioWindowProc
WinDefWindowProc
WinSendMessage
WinSendDlgItemMsg

See Also **MRFROMLONG**

MAKEERRORID

MAKEERRORID(*sev, error*)

The **MAKEERRORID** macro creates an error identifier that consists of a severity level and an error number.

Parameters *sev* Specifies a severity level, which can be any value in the range 0 through 65,535. You may use the following constants:

SEVERITY_NOERROR
SEVERITY_WARNING
SEVERITY_ERROR
SEVERITY_SEVERE
SEVERITY_UNRECOVERABLE

error Specifies an error value, which can be any value in the range 0 through 65,535.

See Also **ERRORIDERROR, ERRORIDSEV**

■ MAKEFIXED

MAKEXED(*intpart*, *fractpart*)

The **MAKEXED** macro creates a **FIXED** value from an integer and a fraction.

Parameters *intpart* Specifies the integer part.
 fractpart Specifies the fractional part.

See Also **FIXEDINT**, **FIXEDFRAC**

■ MAKEINTATOM

MAKEINTATOM(*a*)

The **MAKEINTATOM** macro creates an integer atom from an atom.

Parameters *a* Specifies the atom.

■ MOUSEMSG

MOUSEMSG(*pmsg*)

The **MOUSEMSG** macro is used to access the **WM_MOUSEMOVE** and **WM_BUTTON** message parameters. This macro defines a **MOUSEMSG** structure pointer, which has the following form:

```
struct _MOUSEMSG {  
    USHORT codeHitTest;       /* mp2 */  
    USHORT unused;  
    SHORT x;                 /* mp1 */  
    SHORT y;  
};
```

Parameters *pmsg* Points to the message received by the application's window procedure.

See Also **WM_BUTTON**, **WM_MOUSEMOVE**

■ MPFROM2SHORT

MPFROM2SHORT(*s1*, *s2*)

The **MPFROM2SHORT** macro creates a message parameter from two **SHORT** values.

Parameters *s1* Specifies the first **SHORT** value.
 s2 Specifies the second **SHORT** value.

See Also **MRFROM2SHORT**, **SHORT1FROMMP**, **SHORT2FROMMP**

■ MPFROMCHAR

MPFROMCHAR(*ch*)

The **MPFROMCHAR** macro creates a message parameter from a character.

Parameters *ch* Specifies the character.

See Also **CHAR1FROMMP**

■ MPFROMHWN

MPFROMHWN(*hwnd*)

The **MPFROMHWN** macro creates a message parameter from a window handle (**HWN**).

Parameters *hwnd* Specifies the window handle.

See Also **HWNDFROMMP**

■ MPFROMLONG

MPFROMLONG(*l*)

The **MPFROMLONG** macro creates a message parameter from a **LONG** value.

Parameters *l* Specifies the **LONG** value.

See Also **LONGFROMMP, MRFROMLONG**

■ MPFROMP

MPFROMP(*p*)

The **MPFROMP** macro creates a message parameter from a pointer.

Parameters *p* Specifies the pointer.

See Also **MRFROMP, PVOIDFROMMP**

■ MPFROMSH2CH

MPFROMSH2CH(*s, uch1, uch2*)

The **MPFROMSH2CH** macro creates a message parameter from a **SHORT** value and two unsigned characters.

Parameters *s* Specifies the **SHORT** value.
 uch1 Specifies the first unsigned character.
 uch2 Specifies the second unsigned character.

See Also **CHAR3FROMMP, CHAR4FROMMP, SHORT1FROMMP**

■ MPFROMSHORT

MPFROMSHORT(*s*)

The **MPFROMSHORT** macro creates a message parameter from a **SHORT** value.

Parameters *s* Specifies the **SHORT** value.

See Also **MRFROMSHORT, SHORT1FROMMP**

■ MRFROM2SHORT

MRFROM2SHORT(*s1, s2*)

The **MRFROM2SHORT** macro creates a message result from two **SHORT** values.

Parameters *s1* Specifies the first **SHORT** value.
 s2 Specifies the second **SHORT** value.

See Also **MPFROM2SHORT, SHORT1FROMMR, SHORT2FROMMR**

■ MRFROMLONG

MRFROMLONG(*l*)

The **MRFROMLONG** macro creates a message result from a **LONG** value.

Parameters *l* Specifies the **LONG** value.

See Also **LONGFROMMR, MPFROMLONG**

■ MRFROMP

MRFROMP(*p*)

The **MRFROMP** macro creates a message result from a pointer.

Parameters *p* Specifies the pointer.

See Also **MPFROMP, PVOIDFROMMR**

■ MRFROMSHORT

MRFROMSHORT(*s*)

The MRFROMSHORT macro creates a message result from a SHORT value.

Parameters *s* Specifies the SHORT value.

See Also MPFROMSHORT, SHORT1FROMMR

■ PDDEITOSEL

PDDEITOSEL(*pddei*)

The PDDEITOSEL macro creates a selector from a pointer to a DDEINIT structure. This is necessary in order to use the **DosFreeSeg** function to free the memory that the DDEINIT structure uses.

Parameter *pddei* Points to a DDEINIT structure.

See Also PDDESTOSEL

■ PDDESTOSEL

PDDESTOSEL(*pdde*)

The PDDESTOSEL macro creates a selector from a pointer to a DDESTRUCT structure. This is necessary in order to use the **DosFreeSeg** function to free the memory that the DDESTRUCT structure uses.

Parameter *pdde* Points to a DDESTRUCT structure.

See Also PDDEITOSEL, SELTOPDDES

■ PVOIDFROMMP

PVOIDFROMMP(*mp*)

The PVOIDFROMMP macro creates a pointer from a message parameter.

Parameters *mp* Specifies the message parameter.

See Also MPFROMMP, PVOIDFROMMR

■ PVOIDFROMMR

PVOIDFROMMR(*mr*)

The PVOIDFROMMR macro creates a pointer from a message result.

Parameters *mr* Specifies the message result.

See Also MRFROMP, PVOIDFROMMP

■ SELTOPDDES

SELTOPDDES(*sel*)

The SELTOPDDES macro creates a pointer from a selector; this pointer is to a DDESTRUCT structure.

Parameters *sel* Specifies the selector.

See Also PDDESTOSEL

■ SETMEMBACKPTR

SETMEMBACKPTR(*npb*)

The SETMEMBACKPTR macro creates a back pointer from the near pointer returned by the WinAllocMem and WinReallocMem functions.

Parameters *npb* Specifies the near pointer.

See Also WinAllocMem, WinReallocMem

■ SHORT1FROMMP

SHORT1FROMMP(*mp*)

The SHORT1FROMMP macro creates a SHORT value from the low word of a message parameter.

Parameters *mp* Specifies the message parameter.

See Also MPFROMSHORT, SHORT1FROMMR

■ SHORT1FROMMR

SHORT1FROMMR(*mr*)

The **SHORT1FROMMR** macro creates a **SHORT** value from the low word of a message result.

Parameters *mr* Specifies the message result.

See Also MRFROMSHORT, SHORT1FROMMP

■ SHORT2FROMMP

SHORT2FROMMP(*mp*)

The **SHORT2FROMMP** macro creates a **SHORT** value from the high word of a message parameter.

Parameters *mp* Specifies the message parameter.

See Also MPFROM2SHORT, SHORT2FROMMR

■ SHORT2FROMMR

SHORT2FROMMR(*mr*)

The **SHORT2FROMMR** macro creates a **SHORT** value from the high word of a message result.

Parameters *mr* Specifies the message result.

See Also MRFROM2SHORT, SHORT2FROMMP

4.4 Structures

The following is a complete list, in alphabetical order, of the structures used by the functions described in this manual.

■ ACCEL

```
typedef struct _ACCEL {    /* acc */
    USHORT cmd;
    USHORT fs;
    USHORT key;
} ACCEL;
```

The **ACCEL** structure contains an accelerator key used in the **ACCELTABLE** structure.

Fields

cmd Specifies the value to be placed in the *usCmd* parameter of the **WM_HELP**, **WM_COMMAND**, or **WM_SYSCOMMAND** message.

fs Specifies the style of the accelerator. This field can be one of the following values:

Value	Meaning
AF_ALT	The ALT key must be held down when the accelerator key is pressed.
AF_CHAR	The keystroke is a translated character, using the code page for the accelerator table. This is the default style.
AF_CONTROL	The CONTROL key must be held down when the accelerator key is pressed.
AF_HELP	The accelerator key generates a WM_HELP message instead of a WM_COMMAND message.
AF_LONEKEY	No other key was pressed while the accelerator key was down. This style typically is used with the ALT key to specify that simply pressing and releasing the ALT key triggers the accelerator.
AF_SCANCODE	The keystroke is an untranslated scan code from the keyboard.
AF_SHIFT	The SHIFT key must be held down when the accelerator key is pressed.
AF_SYSCOMMAND	When pressed, the accelerator key generates a WM_SYSCOMMAND message instead of a WM_COMMAND message.
AF_VIRTUALKEY	The keystroke is a virtual key—for example, F1.

key Specifies the accelerator key.

See Also

ACCELTABLE, **WM_COMMAND**, **WM_HELP**, **WM_SYSCOMMAND**

■ ACCELTABLE

```
typedef struct _ACCELTABLE {    /* acct */
    USHORT cAccel;
    USHORT codepage;
    ACCEL aaccel[1];
} ACCELTABLE;
```

The **ACCELTABLE** structure contains an accelerator table.

- Fields**
- cAccel** Specifies the number of accelerator entries. The default is 28.
 - codepage** Specifies the code page for the accelerator entries in the accelerator table of the default queue.
 - aaccel[1]** Specifies the accelerator entries. The actual dimension of this array of ACCEL structures is dependent on the number of accelerator entries.
- See Also** WinCreateAccelTable, WinCopyAccelTable, ACCEL

■ ARCPARAMS

```
typedef struct _ARCPARAMS {    /* arcp */
    LONG  IP;
    LONG  IQ;
    LONG  IR;
    LONG  IS;
} ARCPARAMS;
```

The ARCPARAMS structure contains arc parameters that define the shape and orientation of an ellipse that is used for subsequent GpiFullArc, GpiPartialArc and GpiPointArc functions.

- Fields**
- IP** Specifies the length of the horizontal-scaling vector.
 - IQ** Specifies the length of the vertical-scaling vector.
 - IR** Specifies the position of the horizontal-shear vector.
 - IS** Specifies the position of the vertical-shear vector.

Comments Most arcs and ellipses are drawn without shear. This means that the IR and IS fields are usually set to zero.

See Also GpiFullArc, GpiPartialArc, GpiPointArc, GpiQueryArcParams, GpiSetArcParams

■ AREABUNDLE

```
typedef struct _AREABUNDLE {    /* pbnd */
    LONG  IColor;
    LONG  lBackColor;
    USHORT usMixMode;
    USHORT usBackMixMode;
    USHORT usSet;
    USHORT usSymbol;
    POINTL ptlRefPoint;
} AREABUNDLE;
```

The AREABUNDLE structure contains fields that describe the current fill pattern of the presentation space. MS OS/2 uses this fill pattern when the application constructs areas or paths.

- Fields**
- IColor** Specifies the foreground color of the area fill pattern.
 - lBackColor** Specifies the background color of the area fill pattern.

usMixMode Specifies the foreground mix mode the system uses when it combines the foreground fill-pattern color with the current drawing-surface color.

usBackMixMode Specifies the background mix mode the system uses when it combines the background fill-pattern color with the current drawing surface color.

usSet Specifies the local identifier value for the fill-pattern bitmap or font.

usSymbol Specifies the character or symbol code-point. This field is useful only if the fill pattern is generated from a character or symbol in an image font.

ptlRefPoint Points to the **POINTL** structure that contains the coordinates of the pattern reference point.

See Also

GpiQueryAttrs, **GpiQueryPattern**, **GpiQueryPatternRefPoint**, **GpiQueryPatternSet**, **GpiSetAttrs**, **GpiSetPattern**, **GpiSetPatternRefPoint**, **GpiSetPatternSet**, **POINTL**

■ BITMAPINFO

```
typedef struct _BITMAPINFO { /* bmi */
    ULONG   cbFix;
    USHORT  cx;
    USHORT  cy;
    USHORT  cPlanes;
    USHORT  cBitCount;
    RGB     argbColor[1];
} BITMAPINFO;
```

The **BITMAPINFO** structure contains a bitmap information table.

Fields

cbFix Specifies the length of the fixed portion of the structure. This value must be 12.

cx Specifies the width of the bitmap (in pels).

cy Specifies the height of the bitmap (in pels).

cPlanes Specifies the number of bit planes.

cBitCount Specifies the number of bits per pel within a plane.

argbColor[1] Specifies a packed array of 24-bit RGB colors. If there are n bits per pel, the array contains 2^n RGB colors, unless n equals 24. The standard-format bitmap, with 24 bits per pel, is assumed to contain RGB colors and does not use the colors array.

See Also

GpiQueryBitmapBits, **GpiSetBitmapBits**

■ BITMAPINFOHEADER

```
typedef struct _BITMAPINFOHEADER {    /* bmp */
    ULONG cbFix;
    USHORT cx;
    USHORT cy;
    USHORT cPlanes;
    USHORT cBitCount;
} BITMAPINFOHEADER;
```

The **BITMAPINFOHEADER** structure contains a bitmap header that defines the structure of a bitmap.

- Fields**
- cbFix** Specifies the size of the structure. For MS OS/2 version 1.1, this value must be 12.
 - cx** Specifies the bitmap width (in pels).
 - cy** Specifies the bitmap height (in pels).
 - cPlanes** Specifies the number of bit planes.
 - cBitCount** Specifies the number of bits per pel within a plane.

See Also GpiCreateBitMap, GpiQueryBitmapParameters

■ BTNCDATA

```
typedef struct _BTNCDATA {    /* btncd */
    USHORT cb;
    USHORT fsCheckState;
    USHORT fsHiliteState;
} BTNCDATA;
```

The **BTNCDATA** structure contains information about a button control.

- Fields**
- cb** Specifies the length of the **BTNCDATA** structure. For MS OS/2 version 1.1, the length is 6.
 - fsCheckState** Specifies the check state of the button.
 - fsHiliteState** Specifies whether the button is highlighted.

■ CATCHBUF

```
typedef struct _CATCHBUF {    /* ctchbf */
    ULONG reserved[4];
} CATCHBUF;
```

The **CATCHBUF** structure is used by the **WinCatch** and **WinThrow** functions to save and restore the execution environment.

- Fields**
- reserved[4]** The contents of this field are reserved for use by MS OS/2.

See Also WinCatch, WinThrow

■ CHARBUNDLE

```
typedef struct _CHARBUNDLE {    /* cbnd */
    LONG    lColor;
    LONG    lBackColor;
    USHORT  usMixMode;
    USHORT  usBackMixMode;
    USHORT  usSet;
    USHORT  usPrecision;
    SIZEF   sizfxCell;
    POINTL  ptlAngle;
    POINTL  ptlShear;
    USHORT  usDirection;
} CHARBUNDLE;
```

The **CHARBUNDLE** structure contains fields that describe the current character attributes in the application's presentation space. MS OS/2 uses these attributes whenever the application draws text using one of the **Gpi** functions.

Fields

lColor Specifies the character foreground color.

lBackColor Specifies the character background color.

usMixMode Specifies the foreground mix mode. MS OS/2 uses this mix mode when it combines the character foreground color and the current drawing-surface color.

usBackMixMode Specifies the background mix mode. MS OS/2 uses this mix mode when it combines the character background color and the current drawing-surface color.

usSet Specifies the character set. This value is the local identifier for the current logical font. It can be any value from 1 through 254.

usPrecision Specifies the current character mode. There are three possible modes: mode 1, mode 2, and mode 3. If mode 1 is set and the current font is an image font, MS OS/2 ignores the current shear, angle, and box attributes. If mode 2 is set and the current font is an image font, MS OS/2 uses the current shear, angle, and box attributes. If mode 3 is set and the current font is an image font, MS OS/2 issues an error message. If the current font is a vector font, MS OS/2 always uses the current shear, angle, and box attributes (regardless of the mode).

sizfxCell Specifies the character-cell size (in world units). This **SIZEF** structure contains two fixed values.

ptlAngle Points to the **POINTL** structure that contains the coordinates of the endpoint of the character-angle vector. The baseline of vector characters is drawn parallel to the character-angle vector.

ptlShear Points to the **POINTL** structure that contains the coordinates of the endpoint of the character-shear vector. The vertical strokes in vector characters are drawn parallel to the character-shear vector.

usDirection Specifies the character direction. For MS OS/2 version 1.1, this field must be set to 1.

See Also

GpiQueryAttrs, **GpiQueryCharAngle**, **GpiQueryCharBox**, **GpiQueryCharSet**, **GpiQueryCp**, **GpiSetAttrs**, **GpiSetCharAngle**, **GpiSetCharBox**, **GpiSetCharSet**, **GpiSetCp**, **POINTL**, **SIZEF**

■ CHARMSG

```

struct _CHARMSG {
    USHORT chr;           /* mp2 */
    USHORT vkey;
    USHORT fs;           /* mp1 */
    UCHAR cRepeat;
    UCHAR scancode;
};

```

The **CHARMSG** structure is used by the **CHARMSG** macro to access key information that is passed in the parameters of the **WM_CHAR** message. Unlike other MS OS/2 structures, this structure is not defined as a type.

Fields

chr Specifies the ASCII character.

vkey Specifies the virtual-key code.

fs Specifies the keyboard-control codes. This field can be one or more of the following values:

Value	Meaning
KC_CHAR	The <i>usChr</i> value is valid; otherwise, <i>mp2</i> is zero.
KC_SCANCODE	The <i>uchScanCode</i> value is valid; otherwise, <i>uchScanCode</i> contains zero.
KC_VIRTUALKEY	The <i>usVKey</i> value is valid; otherwise, <i>usVKey</i> is zero.
KC_KEYUP	The event was a key-up transition; otherwise, it was a key-down transition.
KC_PREVDOWN	The key was previously down; otherwise, it was previously up.
KC_DEADKEY	The character key is a dead key. The application must display the glyph for the dead key without advancing the cursor.
KC_COMPOSITE	The character code is formed by combining the current key with the previous dead key.
KC_INVALIDCOMP	The character code is not valid with the preceding dead key. The application advances the cursor past the dead-key glyph and, if the character is <i>not</i> a space, beeps and displays the new character code.
KC_LONEKEY	This bit is set if the key was pressed and released with no other keys being pressed or released between the time the key was pressed and the time it was released.
KC_SHIFT	The shift state was active when the key was pressed or released.
KC_ALT	The ALT state was active when the key was pressed or released.
KC_CTRL	The CONTROL state was active when the key was pressed or released.

cRepeat Specifies the key-repeat count.

scancode Specifies the character scan code.

See Also

WM_CHAR

■ CLASSINFO

```
typedef struct _CLASSINFO { /* clsi */
    ULONG   fClassStyle;
    PFNWP   pfnWindowProc;
    USHORT  cbWindowData;
} CLASSINFO;
```

The **CLASSINFO** structure contains information about a window class.

Fields

fClassStyle Specifies the class-style flags.

pfnWindowProc Points to the window procedure of the class.

cbWindowData Specifies the number of words stored as part of the class.

See Also

WinQueryClassInfo

■ COMMANDMSG

```
struct _COMMANDMSG {
    USHORT source; /* mp2 */
    BOOL   fMouse;
    USHORT cmd; /* mp1 */
    USHORT unused;
};
```

The **COMMANDMSG** structure is used by the **COMMANDMSG** macro to access information passed in the parameters of the **WM_COMMAND**, **WM_HELP**, and **WM_SYSCOMMAND** messages. Unlike other MS OS/2 structures, this structure is not defined as a type.

Fields

source Specifies the source type. It can be one of the following values:

Value	Meaning
CMDSRC_PUSHBUTTON	Posted by a push-button control. The <i>usCmd</i> parameter is the window identifier of the push button.
CMDSRC_MENU	Posted by a menu control. The <i>usCmd</i> parameter is the identifier of the menu item.
CMDSRC_ACCELERATOR	Posted as the result of an accelerator. The <i>usCmd</i> parameter is the accelerator command value.
CMDSRC_OTHER	Other source. The <i>usCmd</i> parameter gives control-specific information defined for each control type.

fMouse Specifies whether the message was posted as a result of a pointing-device operation. A value of **TRUE** indicates a pointing device (mouse) was used. A value of **FALSE** indicates a keyboard operation.

cmd Specifies the command.

unused This field is not used.

See Also

COMMANDMSG, **WM_COMMAND**, **WM_HELP**, **WM_SYSCOMMAND**

■ CREATESTRUCT

```
typedef struct _CREATESTRUCT { /* crst */
    PVOID pPresParams;
    PVOID pCtlData;
    USHORT id;
    HWND hwndInsertBehind;
    HWND hwndOwner;
    SHORT cy;
    SHORT cx;
    SHORT y;
    SHORT x;
    ULONG flStyle;
    PSZ pszText;
    PSZ pszClass;
    HWND hwndParent;
} CREATESTRUCT;
```

The **CREATESTRUCT** structure contains information about a created window. It is passed as the second parameter of the **WM_CREATE** message. Note that the fields are identical to the parameters in the **WinCreateWindow** function.

Fields

pPresParams Points to the presentation parameters. This field is currently reserved.

pCtlData Points to the buffer that contains class-specific information.

id Specifies the window identifier, the value given by the application allowing a specific child window to be identified. For example, the controls of a dialog box have unique identifiers so that an owner window can distinguish which control has notified it. Window identifiers are also used for frame windows.

hwndInsertBehind Identifies the sibling window behind which the specified window is placed. If this value is **HWND_TOP**, the specified window is placed on top of all its sibling windows. If this value is **HWND_BOTTOM**, the specified window is placed behind all its sibling windows. If this value is not **HWND_TOP** or **HWND_BOTTOM**, or if it is a child window of the desktop window identified by the *ofRom* field, then **NULL** is returned.

hwndOwner Identifies the owner window.

cy Specifies the window depth (in pels).

cx Specifies the window width (in pels).

y Specifies the vertical position of the window relative to the origin of the parent window.

x Specifies the horizontal position of the window relative to the origin of the parent window.

flStyle Specifies the window style. This field can be a combination of one or more of the following values:

Value	Meaning
WS_CLIPCHILDREN	Prevents a window from painting over its child windows.
WS_CLIPSIBLINGS	Prevents a window from painting over its sibling windows.
WS_DISABLED	Disables mouse and keyboard input to the window. This style is used to temporarily prevent the user from using the window.

Value	Meaning
WS_MAXIMIZED	Enlarges the window to the maximum size.
WS_MINIMIZED	Reduces the window to the minimum size.
WS_PARENTCLIP	Prevents a window from painting over its parent window.
WS_SAVEBITS	Saves the image under the window as a bitmap. When the window is moved or hidden, the system restores the image by copying the bits.
WS_SYNCPAINT	Causes the window to receive WM_PAINT messages immediately after a part of the window becomes invalid. Without this style, the window receives WM_PAINT messages only if no other messages are waiting to be processed.
WS_VISIBLE	Makes the window visible. This window will be drawn on the screen unless overlapping windows completely obscure it. Windows without this style are hidden.

pszText Points to window text or other class-specific data. The actual structure of the data is class-specific. This data is usually a zero-terminated string and is often displayed in the window.

pszClass Points to the registered class name. This field is an application-specified name (defined by the `WinRegisterClass` function), the name of a preregistered window-control class, or a window-control (WC) constant.

hwndParent Identifies the parent window of the new window. This field can be any valid window handle.

See Also

`WinCreateWindow`, `WM_CREATE`

■ CURSORINFO

```
typedef struct _CURSORINFO { /* csri */
    HWND    hwnd;
    SHORT   x;
    SHORT   y;
    SHORT   cx;
    SHORT   cy;
    USHORT  fs;
    RECT    rcClip;
} CURSORINFO;
```

The `CURSORINFO` structure contains information about the cursor.

Fields

hwnd Identifies the window in which the cursor is displayed.

x Specifies the horizontal position of the cursor.

y Specifies the vertical position of the cursor.

cx Specifies the horizontal size of the cursor. If this field is zero, the system nominal border width (`SV_CXBORDER`) is used.

cy Specifies the vertical size of cursor. If this field is zero, the system nominal border height (`SV_CYBORDER`) is used.

fs Specifies the appearance of the cursor. This field can be one of the following values:

Value	Meaning
CURSOR_FLASH	The cursor is flashing.
CURSOR_FRAME	The cursor is a frame.
CURSOR_HALFTONE	The cursor is halftone.
CURSOR_SOLID	The cursor is solid.

rcfClip Contains the **RECTL** structure that contains the rectangle within which the cursor is visible. If the cursor moves outside this rectangle, it is clipped and becomes invisible. The rectangle is specified in window coordinates. If this field is **NULL**, the cursor is clipped to the window rectangle identified by the **hwnd** field.

Comments The fields of the **CURSORINFO** structure are identical to the parameters of the **WinCreateCursor** function.

See Also **WinCreateCursor**, **WinQueryCursorInfo**, **RECTL**

■ DDEINIT

```
typedef struct _DDEINIT { /* ddei */
    USHORT cb;
    PSZ pszAppName;
    PSZ pszTopic;
} DDEINIT;
```

The **DDEINIT** structure is passed by the **WM_DDE_INITIATEACK** and **WM_DDE_INITIATE** messages. The fields are normally set by the system.

Fields

- cb** Specifies the length of the object.
- pszAppName** Points to the application name.
- pszTopic** Points to the topic name.

See Also **DDESTRUCT**, **WM_DDE_INITIATE**, **WM_DDE_INITIATEACK**

■ DDESTRUCT

```
typedef struct _DDESTRUCT { /* dde */
    ULONG cbData;
    USHORT fsStatus;
    USHORT usFormat;
    USHORT offszItemName;
    USHORT offabData;
} DDESTRUCT;
```

The **DDESTRUCT** structure is passed as the second parameter of all DDE messages except **WM_DDE_INITIATE** and **WM_DDE_INITIATEACK**.

Fields

- cbData** Specifies the length of the data (in bytes).

fsStatus Specifies various status flags. This field can be one or more of the following values:

Value	Meaning
DDE_FACK	Set for a positive acknowledge message.
DDE_FBUSY	Set if the application is busy.
DDE_FNODATA	Set if there is no data transfer for an advise message.
DDE_FACKREQ	Set for acknowledgment of DDE messages.
DDE_FRESPONSE	Set if there is a response to a request message.
DDE_NOTPROCESSED	Set if the message is not supported.
DDE_FRESERVED	Reserved; must be zero.
DDE_FAPPSTATUS	Upper eight bits of the status word are reserved for application-specific data.

usFormat Specifies the format of the data.

offsItemName Specifies the offset of the item name.

offabData Specifies the offset of the data.

See Also

DDEINIT

■ DEVOPENSTRUC

```
typedef struct _DEVOPENSTRUC { /* dop */
    PSZ      pszLogAddress;
    PSZ      pszDriverName;
    PDRIVDATA pdrv;
    PSZ      pszDataType;
    PSZ      pszComment;
    PSZ      pszQueueProcName;
    PSZ      pszQueueProcParams;
    PSZ      pszSpoolerParams;
    PSZ      pszNetworkParams;
} DEVOPENSTRUC;
```

The **DEVOPENSTRUC** structure describes an output device. A copy of this structure is passed to the **DevOpenDC** function when a device context is opened.

Fields

pszLogAddress Points to the logical-device address (for example, lpt1).

pszDriverName Points to the device-driver name (for example, PSCRIPT).

pdrv Points to the **DRIVDATA** structure that contains device-driver information. This structure identifies the device-driver version number and the device name. It can also contain additional device-driver data.

pszDataType Points to the device-driver type (for example, PM_Q_STD).

pszComment Points to additional information used by queued devices.

pszQueueProcName Points to additional information used by queued devices.

pszQueueProcParams Points to additional information used by queued devices.

pszSpoolerParams Points to additional information used by queued devices.

pszNetworkParams Points to additional information used by queued devices.

See Also

DevOpenDC

■ DLGTEMPLATE

```
typedef struct _DLGTEMPLATE { /* dlg_t */
    USHORT    cbTemplate;
    USHORT    type;
    USHORT    codepage;
    USHORT    offadlgti;
    USHORT    fsTemplateStatus;
    USHORT    iItemFocus;
    USHORT    coffPresParams;
    DLGTITEM adlgti[1];
} DLGTEMPLATE;
```

The **DLGTEMPLATE** structure contains header information and an array of dialog items. It is used by the **WinCreateDlg** function to create a dialog window instead of loading it from the resource file.

Fields

cbTemplate Specifies the length of the structure (in bytes).

type Specifies the type of the dialog window. This field is currently unused.

codepage Specifies the code page for the dialog window. This field can be one of the following values:

Number	Code page
437	United States
850	Multilingual
860	Portuguese
863	French-Canadian
865	Nordic

offadlgti Specifies the offset from the beginning of the dialog to the array of dialog-item structures. For MS OS/2 version 1.1, this value is 14.

fsTemplateStatus This field is currently unused. It must be set to 1.

iItemFocus Specifies the index of the item that has the focus.

coffPresParams This field is currently unused. It must be set to zero.

adlgti[1] Specifies an array of **DLGTITEM** structures that contain information about each dialog item.

See Also

WinCreateDlg, DLGTITEM

DLGTITEM

```
typedef struct _DLGTITEM { /* dlgti */
    USHORT    fsItemStatus;
    USHORT    cChildren;
    USHORT    cchClassName;
    USHORT    offClassName;
    USHORT    cchText;
    USHORT    offText;
    ULONG    flStyle;
    SHORT    x;
    SHORT    y;
    SHORT    cx;
    SHORT    cy;
    USHORT    id;
    USHORT    offPresParams;
    USHORT    offCtlData;
} DLGTITEM;
```

The **DLGTITEM** structure contains information about a dialog item.

Fields

fsItemStatus This field is currently unused. It must be set to zero.

cChildren Specifies the number of child windows of the dialog window.

cchClassName Specifies the length of the class name.

offClassName Specifies the offset to the class name.

cchText Specifies the length of the dialog-item text.

offText Specifies the offset to the dialog-item text.

flStyle Specifies the dialog-item window style. The high word contains the standard window-style bits. The low word is available for class-specific use. The high word is **WS_GROUP** if the dialog item begins a group. It is **WS_TABSTOP** if the dialog item can receive the focus when the user presses the **TAB** key.

x Specifies the left origin of the dialog-item window.

y Specifies the bottom origin of the dialog-item window.

cx Specifies the width of the dialog item.

cy Specifies the height of the dialog item.

id Identifies the dialog item.

offPresParams Specifies the offset to presentation parameters. This field is currently reserved.

offCtlData Specifies the offset to any control data.

See Also

DLGTEMPLATE

DRIVDATA

```
typedef struct _DRIVDATA { /* driv */
    LONG    cb;
    LONG    lVersion;
    CHAR    szDeviceName[32];
    CHAR    abGeneralData[1];
} DRIVDATA;
```

The **DRIVDATA** structure contains information about a device driver returned by the **DevPostDeviceModes** function.

- Fields**
- cb** Specifies the length of the structure (in bytes).
 - IVersion** Specifies the version number of the data. Version numbers are defined by particular MS OS/2 device drivers.
 - szDeviceName[32]** Specifies a 32-byte character string that identifies the particular device (for example, model number). Valid values are defined by MS OS/2 device drivers.
 - abGeneralData[1]** Specifies an array of bytes containing general data as defined by the MS OS/2 device driver. The number of bytes is defined by the particular device driver. This array should not contain pointers, because these are not necessarily valid when passed to the device driver.

See Also [DevPostDeviceModes](#)

■ ERRINFO

```
typedef struct _ERRINFO { /* erri */
    USHORT    cbFixedErrInfo;
    ERRORID   idError;
    USHORT    cDetailLevel;
    USHORT    offaoffsMsg;
    USHORT    offBinaryData;
} ERRINFO;
```

The **ERRINFO** structure contains the previous error for the current thread.

- Fields**
- cbFixedErrInfo** Specifies the length of the structure.
 - idError** Identifies the error. This field is identical to the value returned by the [WinGetLastError](#) function.
 - cDetailLevel** Specifies the number of levels of detail.
 - offaoffsMsg** Specifies an offset to an array of offsets to messages.
 - offBinaryData** Specifies an offset to the binary data.

See Also [WinFreeErrorInfo](#), [WinGetErrorInfo](#), [WinGetLastError](#)

■ FATTRS

```
typedef struct _FATTRS { /* fat */
    USHORT    usRecordLength;
    USHORT    fsSelection;
    LONG      lMatch;
    CHAR      szFacename[ACESIZE];
    USHORT    idRegistry;
    USHORT    usCodePage;
    LONG      lMaxBaselineExt;
    LONG      lAveCharWidth;
    USHORT    fsType;
    USHORT    fsFontUse;
} FATTRS;
```

The **FATTRS** structure specifies the attributes of the logical font to be created by the [VioCreateLogFont](#) or [GpiCreateLogFont](#) function.

Fields

usRecordLength Specifies the length of the structure (in bytes).

fsSelection Specifies one or more character attributes. This field can be any combination of the following values:

Value	Meaning
FATTR_SEL_HOLLOW	Requests hollow characters.
FATTR_SEL_ITALIC	Requests italic characters.
FATTR_SEL_NEGATIVE	Requests negative-appearance characters.
FATTR_SEL_STRIKEOUT	Requests strikeout characters.
FATTR_SEL_UNDERSCORE	Requests underscored characters.

IMatch Specifies the match number for a specific font. The **VioQueryFonts** and **GpiQueryFonts** functions return a unique match number for each font. When this number is specified in the **IMatch** field, the specified font is used. If the **IMatch** field is zero, the system determines which font gives the best mapping to the required attributes.

szFaceName[FACE SIZE] Specifies the typeface name of the font.

idRegistry Specifies the registry number of the font.

usCodePage Specifies the code-page identifier of the font.

IMaxBaselineExt Specifies the sum of the maximum ascender and descender values for a font.

IAveCharWidth Specifies the average width of a character in a font. This value is obtained by multiplying the width of each lowercase letter by a weighted factor, adding the results for all of the letters in the alphabet, and dividing by 1000. The factor corresponds to the frequency of use for a particular letter. For example, the letter *e* appears frequently in text while the letter *q* does not; therefore, the factor assigned to *e* would be greater than the factor assigned to *q*.

fsType Specifies whether the font uses kerning or proportional spacing. This field can be one of the following values:

Value	Meaning
FATTR_TYPE_FIXED	Reserved; must be zero.
FATTR_TYPE_KERNING	If this flag is set, MS OS/2 attempts to return a kerned font when the Gpi-CreateLogFont function is called.

fsFontUse Specifies how the font is related to the character attributes. This field can be any combination of the following values:

Value	Meaning
FATTR_FONTUSE_NOMIX	The application will not mix text and graphics.
FATTR_FONTUSE_OUTLINE	Requests an outline font.
FATTR_FONTUSE_TRANSFORMABLE	Requests a transformable font.

See Also

GpiCreateLogFont, **GpiQueryFonts**, **VioCreateLogFont**, **VioQueryFonts**

■ FONTMETRICS

```
typedef struct _FONTMETRICS { /* fm */
    CHAR    szFamilyname[FACESIZE];
    CHAR    szFacename[FACESIZE];
    USHORT  idRegistry;
    USHORT  usCodePage;
    LONG    lEmHeight;
    LONG    lXHeight;
    LONG    lMaxAscender;
    LONG    lMaxDescender;
    LONG    lLowerCaseAscent;
    LONG    lLowerCaseDescent;
    LONG    lInternalLeading;
    LONG    lExternalLeading;
    LONG    lAveCharWidth;
    LONG    lMaxCharInc;
    LONG    lEmInc;
    LONG    lMaxBaselineExt;
    SHORT   sCharSlope;
    SHORT   sInlineDir;
    SHORT   sCharRot;
    USHORT  usWeightClass;
    USHORT  usWidthClass;
    SHORT   sXDeviceRes;
    SHORT   sYDeviceRes;
    SHORT   sFirstChar;
    SHORT   sLastChar;
    SHORT   sDefaultChar;
    SHORT   sBreakChar;
    SHORT   sNominalPointSize;
    SHORT   sMinimumPointSize;
    SHORT   sMaximumPointSize;
    USHORT  fsType;
    USHORT  fsDefn;
    USHORT  fsSelection;
    USHORT  fsCapabilities;
    LONG    lSubscriptXSize;
    LONG    lSubscriptYSize;
    LONG    lSubscriptXOffset;
    LONG    lSubscriptYOffset;
    LONG    lSuperscriptXSize;
    LONG    lSuperscriptYSize;
    LONG    lSuperscriptXOffset;
    LONG    lSuperscriptYOffset;
    LONG    lUnderscoreSize;
    LONG    lUnderscorePosition;
    LONG    lStrikeoutSize;
    LONG    lStrikeoutPosition;
    SHORT   sKerningPairs;
    SHORT   sReserved;
    LONG    lMatch;
} FONTMETRICS;
```

The FONTMETRICS structure contains information about fonts.

Fields

szFamilyname[FACESIZE] Specifies the family name of the font. Examples of common family names in MS OS/2 version 1.1 are Courier and Helvetica.

szFacename[FACESIZE] Specifies the typeface name of the font. Examples of common typeface names in MS OS/2 version 1.1 are Courier and Helvetica.

idRegistry Specifies the registry number of the font. For MS OS/2 version 1.1, this value must be 0.

usCodePage Identifies the code page that an application should use with the particular font. For MS OS/2 version 1.1, this value must be 850.

lEmHeight Specifies the average height of uppercase characters. The height is measured in world coordinates from the baseline to the top of the character.

lXHeight Specifies the average height of lowercase characters. The height is measured in world coordinates from the baseline to the top of the character.

lMaxAscender Specifies the maximum height of any character in the font. The height is measured in world coordinates from the baseline to the top of the character.

lMaxDescender Specifies the maximum depth of any character in the font. The depth is measured in world coordinates from the baseline to the bottom of the lowest character.

lLowerCaseAscent Specifies the maximum height of any lowercase character in the font. The height is measured in world coordinates from the baseline to the top of the ascender of the tallest lowercase character.

lLowerCaseDescent Specifies the maximum depth of any lowercase character in the font. The depth is measured in world coordinates from the baseline to the bottom of the descender of the lowest lowercase character.

lInternalLeading Specifies the amount of space reserved in the top of each character cell for accent marks. This metric is always given in world coordinates.

lExternalLeading Specifies the amount of space that should appear between adjacent rows of text. This metric is always given in world coordinates.

lAveCharWidth Specifies the average character width for characters in the font. The average character width is determined by multiplying the width of each lowercase character by a predetermined constant, adding the results, and then dividing by 1000. Letters and their predetermined constants are listed as follows:

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

lMaxCharInc Specifies the maximum increment between characters in the font.

lEMInc Specifies the width of an uppercase *M* in the font.

lMaxBaselineExt Specifies the sum of the maximum ascender and maximum descender values.

sCharSlope Specifies the angle (in degrees and minutes) between a vertical line and the upright strokes in characters in the font. The first nine bits of this value contain the degrees, the next six bits contain the minutes, and the last bit is reserved. The slope of characters in a normal font is zero; the slope of italic characters is nonzero.

sInlineDir Specifies an angle (in degrees and minutes, increasing clockwise) from the *x*-axis that the system uses when it draws a text string. The system draws each consecutive character from the text string in the inline direction. The inline-direction angle for a Swiss font is zero; the inline direction for a Hebrew font is 180.

sCharRot Specifies the angle (in degrees and minutes) between the baseline of characters in the font and the *x*-axis. This is the angle assigned by the font designer.

usWeightClass Specifies the thickness of the strokes that form the characters in the font. This field can be one of the following values:

Value	Meaning
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

usWidthClass Specifies the relative-aspect ratio of characters in the font in relation to the normal-aspect ratio for a font of this type. The possible values are listed here:

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

sXDeviceRes Specifies the horizontal resolution of the target device for which the font was originally designed. This value is given in pels per inch.

sYDeviceRes Specifies the vertical resolution of the target device for which the font was originally designed. This value is given in pels per inch.

sFirstChar Specifies the code point for the first character in the font.

sLastChar Specifies the code point for the last character in the font. This code point is an offset from the **sFirstChar** value.

sDefaultChar Specifies the code point for the default character in the font. This code point is an offset from the **sFirstChar** value. The default character is the character the system uses when an application specifies a code point that is out of the range of a font's code page.

sBreakChar Specifies the code point for the space character in the font. This code point is an offset from the **sFirstChar** value.

sNominalPointSize Specifies the height of the font in decipoints (1/720 inch). The nominal point size is the point size for which the font was designed.

sMinimumPointSize Specifies the minimum height of the font (in deci-points). A font should not be reduced to a size smaller than this value.

sMaximumPointSize Specifies the maximum height of the font (in deci-points). A font should not be increased to a size larger than this value.

fsType Specifies whether the font is proportional or fixed, whether it is licensed or not licensed, and whether it consumes more or less than 64K of memory. The following list shows the significance of the bits in this field:

Bit	Description
0	Font is fixed if this bit is set; otherwise it is proportional.
1	Font is licensed if this bit is set; otherwise it is not licensed.
2-14	These bits are reserved and must be set to zero.
15	Font requires more than 64K of memory if this bit is set; otherwise, the font requires 64K or less.

fsDefn Specifies whether the font is an image or vector font. If bit zero is set, the font is a vector font; otherwise it is an image font.

fsSelection Specifies whether the font is normal or italic, whether it is underscored, whether it uses positive or negative image characters, whether it uses solid or outlined characters, and whether it uses overstruck characters. The following list indicates the purpose of each bit in the **fsSelection** field:

Bit	Description
0	Characters are italic if this bit is set; otherwise, they are normal.
1	Characters are underscored if this bit is set; otherwise, they are not underscored.
2	Characters are drawn using negative images if this bit is set; otherwise they are drawn using positive images.
3	Characters are outlined if this bit is set; otherwise, they are solid.
4	Characters are overstruck if this bit is set; otherwise, they are not overstruck.
5-15	These bits are reserved and must be set to zero.

fsCapabilities Specifies whether the characters in this font can be mixed with graphics. If bit 0 of this field is set, the characters cannot be mixed with graphics; otherwise, they can be mixed with graphics. Bits 1-15 of this field are reserved and must be set to zero.

lSubscriptXSize Specifies the horizontal size (in world coordinates) for subscripts in the font.

lSubscriptYSize Specifies the vertical size (in world coordinates) for subscripts in the font.

lSubscriptXOffset Specifies the horizontal offset from the left edge of the character cell.

lSubscriptYOffset Specifies the vertical offset from the character-cell baseline.

- ISuperscriptXSize** Specifies the horizontal size (in world coordinates) for superscripts in the font.
- ISuperscriptYSize** Specifies the vertical size (in world coordinates) for superscripts in the font.
- ISuperscriptXOffset** Specifies the horizontal offset from the left edge of the character cell.
- ISuperscriptYOffset** Specifies the vertical offset from the character-cell baseline.
- IUnderscoreSize** Specifies the width of the underscore (in world coordinates).
- IUnderscorePosition** Specifies the distance from the baseline to the underscore line (in world coordinates).
- IStrikeoutSize** Specifies the width of the overstrike (in world coordinates).
- IStrikeoutPosition** Specifies the position of the overstrike in relation to the baseline.
- sKerningPairs** Specifies the number of kerning pairs in the kerning-pair table for the font.
- sReserved** This field is reserved.
- lMatch** Specifies a long integer value that should be copied to the **FATTRS** structure when calling the **GpiCreateLogFont** function. (When this value is passed, the system must select a font that contains the metrics associated with the **lMatch** field.)

See Also

GpiCreateLogFont, **GpiQueryFontMetrics**, **GpiQueryFonts**, **VioQueryFonts**, **FATTRS**

■ FRAMECDATA

```
typedef struct _FRAMECDATA { /* fcddata */
    USHORT cb;
    ULONG flCreateFlags;
    HMODULE hmodResources;
    USHORT idResources;
} FRAMECDATA;
```

The **FRAMECDATA** structure contains information about the frame controls that are created by using the **WinCreateFrameControls** function.

Fields

- cb** Specifies the size of the structure (in bytes).
- flCreateFlags** Specifies the frame controls to be created.
- hmodResources** Identifies the resource file loaded if the **FCF_MENU** option is specified in the **flCreateFlags** field.
- idResources** Identifies the menu loaded from the resource file identified by **hmodResources** if the **FCF_MENU** option is specified in the **flCreateFlags** field.

See Also

WinCreateFrameControls

■ GRADIENL

```
typedef struct _GRADIENL { /* gradl */
    LONG x;
    LONG y;
} GRADIENL;
```

The **GRADIENL** structure specifies the endpoint of a special angle vector. The baseline of a character drawn with a **Gpi** text-output function is aligned with this vector.

- Fields**
- x** Specifies the *x*-coordinate of the endpoint of the angle vector.
 - y** Specifies the *y*-coordinate of the endpoint of the angle vector.
- Comments** The angle vector is drawn from the origin of a Cartesian coordinate system to the endpoint specified by the **x** and **y** fields.
- See Also** **GpiQueryCharAngle**, **GpiSetCharAngle**

■ HCINFO

```
typedef struct _HCINFO { /* hci */
    CHAR szFormname[32];
    LONG cx;
    LONG cy;
    LONG xLeftClip;
    LONG yBottomClip;
    LONG xRightClip;
    LONG yTopClip;
    LONG xPels;
    LONG yPels;
    LONG flAttributes;
} HCINFO;
```

The **HCINFO** structure contains information about the hard copy capabilities of a device.

- Fields**
- szFormname[32]** Specifies the form name.
 - cx** Specifies the form width (in millimeters).
 - cy** Specifies the form height (in millimeters).
 - xLeftClip** Specifies the left clip limit (in millimeters).
 - yBottomClip** Specifies the bottom clip limit (in millimeters).
 - xRightClip** Specifies the right clip limit (in millimeters).
 - yTopClip** Specifies the top clip limit (in millimeters).
 - xPels** Specifies the number of pels between the left and right clip limits.
 - yPels** Specifies the number of pels between the top and bottom clip limits.
 - flAttributes** Specifies the attributes of the form identifier.
- See Also** **DevQueryHardcopyCaps**

■ HPROGARRAY

```
typedef struct _HPROGRAM { /* hpga */
    HPROGRAM ahprog[1];
} HPROGRAM;
```

The **HPROGRAM** structure contains an array of program handles returned by the **WinQueryProgramHandle** function.

Fields **ahprog[1]** Identifies the program.

See Also **WinQueryProgramHandle**

■ IMAGEBUNDLE

```
typedef struct _IMAGEBUNDLE { /* ibmd */
    LONG    lColor;
    LONG    lBackColor;
    USHORT  usMixMode;
    USHORT  usBackMixMode;
} IMAGEBUNDLE;
```

The **IMAGEBUNDLE** structure contains the current image colors and mix modes in the application's presentation space. MS OS/2 uses these fields when an application draws an image using the **GpiImage** function. The system combines the image foreground and background colors with the current drawing-surface colors by using the foreground- and background-image mix modes.

Fields **lColor** Specifies the foreground-image color.

lBackColor Specifies the background-image color.

usMixMode Specifies the foreground mix mode.

usBackMixMode Specifies the background mix mode.

See Also **GpiImage, GpiQueryAttrs, GpiSetAttrs**

■ KERNINGPAIRS

```
typedef struct _KERNINGPAIRS { /* krnpr */
    SHORT  sFirstChar;
    SHORT  sSecondChar;
    SHORT  sKerningAmount;
} KERNINGPAIRS;
```

The **KERNINGPAIRS** structure contains kerning-pair information for a logical font.

Fields **sFirstChar** Specifies the code point for the first character in the kerning pair.

sSecondChar Specifies the code point for the second character in the kerning pair.

sKerningAmount Specifies the amount of kerning between the pair of characters. This amount is always specified in world coordinates.

See Also **GpiQueryKerningPairs**

■ LINEBUNDLE

```
typedef struct _LINEBUNDLE {    /* lbnd */
    LONG    lColor;
    LONG    lReserved;
    USHORT  usMixMode;
    USHORT  usReserved;
    FIXED   fxWidth;
    LONG    lGeomWidth;
    USHORT  usType;
    USHORT  usEnd;
    USHORT  usJoin;
} LINEBUNDLE;
```

The **LINEBUNDLE** structure contains the current line attributes in the application's presentation space. When the application draws a line or an arc primitive, MS OS/2 uses these attributes to achieve the correct line color, width, and style.

Fields

IColor Specifies the line color.

lReserved This field is reserved. It must be set to zero.

usMixMode Specifies the mix mode that MS OS/2 uses when it combines the line color with the drawing-surface color.

usReserved This field is reserved. It must be set to zero.

fxWidth Specifies the width of cosmetic lines. For MS OS/2 version 1.1, this field must be set to 1.

lGeomWidth Specifies the width of a geometric line (in pels).

usType Specifies the cosmetic line type.

usEnd Specifies the end-style for geometric lines.

usJoin Specifies the join-style for geometric lines.

See Also

GpiQueryAttrs, **GpiQueryLineType**, **GpiQueryLineWidth** **GpiSetAttrs**, **GpiSetLineType**, **GpiSetLineWidth**

■ MARKERBUNDLE

```
typedef struct _MARKERBUNDLE {    /* mbnd */
    LONG    lColor;
    LONG    lBackColor;
    USHORT  usMixMode;
    USHORT  usBackMixMode;
    USHORT  usSet;
    USHORT  usSymbol;
    SIZEF   sizfxCell;
} MARKERBUNDLE;
```

The **MARKERBUNDLE** structure contains information on the current marker in the application's presentation space. When the application draws a marker using with the **GpiMarker** or **GpiPolyMarker** function, MS OS/2 uses the color, mix mode, character set, character symbol, and cell values found in this structure.

Fields

IColor Specifies the foreground-marker color.

lBackColor Specifies the background-marker color.

usMixMode Specifies the foreground mix mode.

usBackMixMode Specifies the background mix mode.

usSet Specifies the local identifier for the font. This field should be set only if the application requires a custom marker using a symbol or character from the specified font.

usSymbol Specifies the code-point for a character or symbol in the font. This field should be set only if the application requires a custom marker using the specified character or symbol from the specified font.

sizfxCell Specifies the marker-box dimensions (in world coordinates).

See Also

GpiMarker, GpiPolyMarker, GpiQueryAttrs, GpiQueryMarker, GpiQueryMarkerBox, GpiQueryMarkerSet, GpiSetAttrs, GpiSetMarker, GpiSetMarkerBox, GpiSetMarkerSet, SIZEF

■ **MATRIXLF**

```
typedef struct _MATRIXLF { /* matlf */
    FIXED fxM11;
    FIXED fxM12;
    LONG lM13;
    FIXED fxM21;
    FIXED fxM22;
    LONG lM23;
    LONG lM31;
    LONG lM32;
    LONG lM33;
} MATRIXLF;
```

The **MATRIXLF** structure contains the scaling, translation, rotation, shear, and reflection transformation values that MS OS/2 uses when the application calls one of the transformation functions.

If the matrix contains scaling transformation values, the following fields are set:

Field	Description
fxM11	Specifies the horizontal scaling value.
fxM12	Specifies the vertical scaling value.

If the matrix contains translation transformation values, the following fields are set:

Field	Description
lM31	Specifies the horizontal translation value.
lM32	Specifies the vertical translation value.

If the matrix contains rotation transformation values, the following fields are set:

Field	Description
fxM11	Specifies the cosine of the angle of rotation.
fxM12	Specifies the negative sine of the angle of rotation.
fxM21	Specifies the sine of the angle of rotation.
fxM22	Specifies the cosine of the angle of rotation.

If the matrix contains vertical-shear transformation values, the following fields are set:

Field	Description
fxM21	Specifies the horizontal-shear value.
fxM22	Specifies the vertical-shear value.

If the matrix contains horizontal-shear transformation values, the following fields are set:

Field	Description
fxM11	Specifies the horizontal-shear value.
fxM12	Specifies the vertical-shear value.

If the matrix contains reflection values, the following fields are set:

Field	Description
fxM11	Specifies the vertical-reflection value. (This value is always negative. It causes reflection about the <i>x</i> -axis.)
fxM22	Specifies the horizontal-reflection value. (This value is always negative. It causes reflection about the <i>y</i> -axis.)

See Also

GpiQueryDefaultViewMatrix, **GpiQueryModelTransformMatrix**, **GpiQuerySegmentTransformMatrix**, **GpiQueryViewingTransformMatrix**, **GpiSetDefaultViewMatrix**, **GpiSetModelTransformMatrix**, **GpiSetSegmentTransformMatrix**, **GpiSetViewingTransformMatrix**

■ MENUITEM

```
typedef struct _MENUITEM {    /* mi */
    SHORT    iPosition;
    USHORT   afStyle;
    USHORT   afAttribute;
    USHORT   id;
    HWND     hwndSubMenu;
    ULONG    hItem;
} MENUITEM;
```

The **MENUITEM** structure contains information about a menu item.

Fields

iPosition Specifies the ordinal position of the item within its menu window. If the menu item is part of a submenu, **iPosition** gives its relative top-to-bottom and left-to-right position, with zero being the upper-left item.

afStyle Specifies the style bits of the item.

afAttribute Specifies the attribute bits of the item.

id Specifies the menu identifier.

hwndSubMenu Identifies the window of the menu window if the item is a submenu item. Command items contain **NULL** in this field.

hItem Identifies the display object for the item. If the item has the **MIS_TEXT** style bit set, this field is **NULL**.

See Also

WinLoadMenu, **MM_INSERTITEM**, **MM_QUERYITEM**, **MM_SETITEM**

■ MOUSEMSG

```

struct _MOUSEMSG {
    USHORT codeHitTest;    /* mp2 */
    USHORT unused;
    SHORT  x;             /* mp1 */
    SHORT  y;
};

```

The **MOUSEMSG** structure contains the message parameters passed with the **WM_MOUSEMOVE** and **WM_BUTTON** messages. Unlike other MS OS/2 structures, this structure is not defined as a type.

- Fields**
- codeHitTest** Specifies the result of a **WM_HITTEST** message, or zero if a mouse capture is in progress.
 - unused** This field is not used.
 - x** Specifies the horizontal mouse coordinate relative to the window's lower-left corner.
 - y** Specifies the vertical mouse coordinate relative to the window's lower-left corner.

See Also **MOUSEMSG**, **WM_BUTTON**, **WM_MOUSEMOVE**

■ MOVBLOCKHDR

```

typedef struct _MOVBLOCKHDR {    /* mbhdr */
    NPBYTE *ppmem;
    USHORT cb;
} MOVBLOCKHDR;

```

The **MOVBLOCKHDR** structure is used at the head of an allocated memory block from a moveable heap.

- Fields**
- ppmem** Specifies the address of the pointer in global memory to be changed when the heap moves.
 - cb** Specifies the size of the memory block.

See Also **WinAllocMem**, **WinCreateHeap**

■ MQINFO

```

typedef struct _MQINFO {    /* mqi */
    USHORT cb;
    PID    pid;
    TID    tid;
    USHORT cmsgs;
    PVOID  pReserved;
} MQINFO;

```

The **MQINFO** structure contains information about a message queue.

- Fields**
- cb** Specifies the length of the structure (in bytes).
 - pid** Specifies the process identifier of the process that created the message queue.
 - tid** Specifies the thread identifier of the thread that created the message queue.
 - cmsgs** Specifies the maximum number of messages that can be held in the queue.
 - pReserved** Specifies a reserved value.

See Also WinQueryQueueInfo

■ OWNERITEM

```
typedef struct _OWNERITEM {    /* oi */
    HWND    hwnd;
    HPS     hps;
    USHORT  fsState;
    USHORT  fsAttribute;
    USHORT  fsStateOld;
    USHORT  fsAttributeOld;
    RECT    rcItem;
    SHORT   idItem;
    ULONG   hItem;
} OWNERITEM;
```

The OWNERITEM structure contains information about an item, usually a list-box or a menu item.

- Fields**
- hwnd** Identifies the handle of the item.
 - hps** Identifies the presentation space.
 - fsState** Specifies highlighting flags.
 - fsAttribute** Specifies attribute flags.
 - fsStateOld** Specifies previous highlighting flags.
 - fsAttributeOld** Specifies previous attribute flags.
 - rcItem** Specifies the RECTL structure that contains the coordinates of the rectangle that bounds the item.
 - idItem** Identifies the item.
 - hItem** Specifies an application-defined item handle.

See Also WM_DRAWITEM, WM_MEASUREITEM

■ PIBSTRUCT

```
typedef struct _PIBSTRUCT { /* pib */
    PROGTTYPE   progt;
    CHAR        szTitle[MAXNAMEL+1];
    CHAR        szIconFileName[MAXPATHL+1];
    CHAR        szExecutable[MAXPATHL+1];
    CHAR        szStartupDir [MAXPATHL+1];
    XYWINSIZE   xywinInitial;
    USHORT      res1;
    LHANDLE     res2;
    USHORT      cchEnvironmentVars;
    PCH         pchEnvironmentVars;
    USHORT      cchProgramParameter;
    PCH         pchProgramParameter;
} PIBSTRUCT;
```

The **PIBSTRUCT** structure contains information about a program within a group. This list is displayed by the Presentation Manager Start Programs window.

Fields

progt Specifies the program type.

szTitle[MAXNAMEL+1] Specifies the program title.

szIconFileName[MAXPATHL+1] Specifies the title to use when the program is minimized.

szExecutable[MAXPATHL+1] Specifies the path of the executable file.

szStartupDir[MAXPATHL+1] Specifies the default drive and directory.

xywinInitial Specifies the initial window position.

res1 This field is reserved.

res2 This field is reserved.

cchEnvironmentVars Specifies the length of the environment variables.

pchEnvironmentVars Points to the environment variables.

cchProgramParameter Specifies the length of the program parameters.

pchProgramParameter Points to the program parameters.

See Also

WinAddProgram, WinChangeProgram, WinQueryDefinition

■ POINTERINFO

```
typedef struct _POINTERINFO { /* ptri */
    BOOL        fPointer;
    SHORT       xHotspot;
    SHORT       yHotspot;
    HBITMAP     hbmPointer;
} POINTERINFO;
```

The **POINTERINFO** structure contains information about the mouse pointer.

Fields

fPointer Specifies whether the pointer is an icon-sized or pointer-sized bitmap. If this value is **TRUE**, the pointer is a pointer-sized bitmap. If **FALSE**, the pointer is an icon-sized bitmap.

xHotspot Specifies the horizontal position of the hot spot.
yHotspot Specifies the vertical position of the hot spot.
hbmPointer Identifies the bitmap used to draw the pointer.

Comments The **xHotspot** and **yHotspot** values are in units relative to the size of the system pointer or the system icon, depending on the **fPointer** field.

See Also **WinQueryPointerInfo**

■ POINTFX

```
typedef struct _POINTFX {    /* ptfx */
    FIXED x;
    FIXED y;
} POINTFX;
```

The **POINTFX** structure contains the coordinates of a point using **FIXED** coordinates instead of **LONG** coordinates used in the **POINTL** structure.

Fields **x** Specifies the horizontal coordinate of the point.
y Specifies the vertical coordinate of the point.

See Also **POINTL**

■ POINTL

```
typedef struct _POINTL {    /* ptl */
    LONG x;
    LONG y;
} POINTL;
```

The **POINTL** structure contains the coordinates of a point.

Fields **x** Specifies the horizontal coordinate of the point.
y Specifies the vertical coordinate of the point.

See Also **GpiSetCurrentPosition**, **GpiMove**, **POINTFX**, **RECTL**

■ PROGRAMENTRY

```
typedef struct _PROGRAMENTRY {    /* proge */
    HPROGRAM hprog;
    PROGTYPE progt;
    CHAR    szTitle[MAXNAMEL+1];
} PROGRAMENTRY;
```

The **PROGRAMENTRY** structure contains information about the programs in a specified group, as returned by the **WinQueryProgramTitles** function.

Fields

hprog Identifies the program.

progt Specifies the type of program.

szTitle[MAXNAMEL+ 1] Specifies the title of the program.

See Also WinQueryProgramTitles

■ PROGTYPE

```
typedef struct _PROGTYPE { /* progt */
    PROGCATEGORY progc;
    BYTE fbVisible;
} PROGTYPE;
```

The **PROGTYPE** structure is used in the **PIBSTRUCT** structure to specify a program type.

Fields

progc Specifies the program category. This field can be one of the following values:

Value	Meaning
PROG_DEFAULT	Default category.
PROG_FULLSCREEN	Full-screen program.
PROG_WINDOWABLEVIO	Program runs in a window.
PROG_PM	Program is a Presentation Manager application.
PROG_GROUP	Handle is to a group.
PROG_REAL	Program is a real-mode application (DOS).

fbVisible Specifies the visibility attribute of the program. It is **SHE_VISIBLE** if the program is visible or **SHE_INVISIBLE** if the program is invisible.

See Also PIBSTRUCT

■ QMSG

```
typedef struct _QMSG { /* qmsg */
    HWND hwnd;
    USHORT msg;
    MPARAM mp1;
    MPARAM mp2;
    ULONG time;
    POINTL pt1;
} QMSG;
```

The **QMSG** structure contains information about a message.

Fields

hwnd Identifies the window handle.

msg Specifies the message.

- mp1** Specifies the first message parameter.
- mp2** Specifies the second message parameter.
- time** Specifies the time the message was generated.
- ptl** Specifies the pointer position when the message was generated.

See Also WinCallMsgFilter, WinDispatchMsg, WinGetMsg, WinPeekMsg

■ RECTFX

```
typedef struct _RECTFX { /* rcfx */
    POINTFX ptfx1;
    POINTFX ptfx2;
} RECTFX;
```

The **RECTFX** structure specifies the coordinates of a rectangle using **FIXED** coordinates instead of **LONG** coordinates used in the **RECTL** structure.

- Fields**
- ptfx1** Specifies the lower-left corner.
 - ptfx2** Specifies the upper-right corner.

See Also RECTL

■ RECTL

```
typedef struct _RECTL { /* rcl */
    LONG xLeft;
    LONG yBottom;
    LONG xRight;
    LONG yTop;
} RECTL;
```

The **RECTL** structure contains the coordinates of a rectangle.

- Fields**
- xLeft** Specifies the left side of the rectangle.
 - yBottom** Specifies the bottom of the rectangle.
 - xRight** Specifies the right side of the rectangle.
 - yTop** Specifies the top of the rectangle.

Comments If the rectangle is drawn in world space, model space, or page space, MS OS/2 includes the bottom and far-right edges. However, if the rectangle is drawn in device space (that is, the application passes the rectangle to the **GpiCreateRegion**, **GpiCombineRegion**, or **GpiOffsetRegion** function) MS OS/2 excludes the bottom and far-right edges of the rectangle.

See Also POINTL, RECTFX

■ RGB

```
typedef struct _RGB { /* rgb */
    BYTE bBlue;
    BYTE bGreen;
    BYTE bRed;
} RGB;
```

The RGB structure contains a color definition. This structure is used by the BITMAPINFO structure.

Fields

- bBlue** Specifies the blue component of the color definition.
- bGreen** Specifies the green component of the color definition.
- bRed** Specifies the red component of the color definition.

See Also BITMAPINFO

■ RGNRECT

```
typedef struct _RGNRECT { /* rgnrc */
    USHORT ircStart;
    USHORT crc;
    USHORT crcReturned;
    USHORT usDirection;
} RGNRECT;
```

The RGNRECT structure contains information that controls the processing of the GpiQueryRegionRects function.

Fields

- ircStart** Specifies the rectangle from which to start enumeration.
- crc** Specifies the number of rectangles that can be returned in the *parcl* parameter of GpiQueryRegionRects. This field must be at least 1.
- crcReturned** Specifies the number of rectangles actually written into the *parcl* parameter of GpiQueryRegionRects. A value below that specified by the *ircStart* field indicates that there are no more rectangles to enumerate.
- usDirection** Specifies the direction in which the (leading edge of the) rectangles are to be returned. This field can be one of the following values:

Value	Meaning
RECTDIR_LFRT_TOPBOT	Left to right, top to bottom.
RECTDIR_RTFL_TOPBOT	Right to left, top to bottom.
RECTDIR_LFRT_BOTTOP	Left to right, bottom to top.
RECTDIR_RTFL_BOTTOP	Right to left, bottom to top.

See Also GpiQueryRegionRects

■ SBCDATA

```
typedef struct _SBCDATA {    /* sbcd */
    USHORT cb;
    USHORT sHilite;
    SHORT posFirst;
    SHORT posLast;
    SHORT posThumb;
} SBCDATA;
```

The **SBCDDATA** structure contains information about a scroll-bar window.

Fields

cb Specifies the length of the structure. For MS OS/2 version 1.1, this field must be 10.

sHilite This field is reserved. It must be set to zero.

posFirst Specifies the first possible position of the slider in the scroll bar.

posLast Specifies the last possible position of the slider in the scroll bar.

posThumb Specifies the current position of the slider in the scroll bar.

■ SIZEF

```
typedef struct _SIZEF {    /* sizfx */
    FIXED cx;
    FIXED cy;
} SIZEF;
```

The **SIZEF** structure specifies the width and height of a rectangle. This structure is used to define the dimensions of a character and marker box.

Fields

cx Specifies the rectangle width (in world coordinates). This is a fixed value.

cy Specifies the rectangle height (in world coordinates). This is a fixed value.

Comments

A fixed value is a binary representation of a floating-point number. A fixed value has two parts: the high-order 16 bits and the low-order 16 bits. The high-order 16 bits contain a signed integer in the range -32,768 through 32,767; the low-order 16 bits contain the numerator of a fraction, in the range 0 through 65,536 (the denominator of this fraction is always 65,536).

See Also

GpiQueryAttrs, **GpiQueryCharBox**, **GpiQueryMarkerBox**, **GpiSetAttrs**, **GpiSetCharBox**, **GpiSetMarkerBox**, **CHARBUNDLE**, **MARKERBUNDLE**, **SIZEL**

■ SIZEL

```
typedef struct _SIZEL {    /* sizl */
    LONG cx;
    LONG cy;
} SIZEL;
```

The **SIZEL** structure specifies the width and the height of a rectangle.

Fields

cx Specifies the rectangle width.

cy Specifies the rectangle height.

See Also GpiCreatePS, GpiImage, GpiQueryBitmapDimensions, GpiQueryDefCharBox, GpiQueryPickApertureSize, GpiQueryPS, GpiSetBitmapDimensions, GpiSetDefCharBox, GpiSetPickApertureSize, GpiSetPS, SIZEF

■ SMHSTRUCT

```
typedef struct _SMHSTRUCT {    /* smhs */
    MPARAM mp2;
    MPARAM mp1;
    USHORT msg;
    HWND hwnd;
} SMHSTRUCT;
```

The **SMHSTRUCT** structure contains information about a message that is used in a send-message hook.

Fields

mp2 Specifies the second message parameter.

mp1 Specifies the first message parameter.

msg Specifies the message.

hwnd Identifies the window sending the message.

■ SWCNTRL

```
typedef struct _SWCNTRL {    /* swctl */
    HWND hwnd;
    HWND hwndIcon;
    HPROGRAM hprog;
    USHORT idProcess;
    USHORT idSession;
    UCHAR uchVisibility;
    UCHAR fbJump;
    CHAR szSwttitle[MAXNAMEL+1];
    BYTE fReserved;
} SWCNTRL;
```

The **SWCNTRL** structure is used when adding or changing a title to the Task Manager switch list.

Fields

hwnd Identifies the window handle.

hwndIcon Identifies the icon handle.

hprog Identifies the program handle.

idProcess Specifies the identifier of the process.

idSession Specifies the identifier of the session.

uchVisibility Specifies the visibility. This field can be one of the following values:

Value	Meaning
SWL_GRAYED	Program cannot be switched to.
SWL_INVISIBLE	Title is invisible in the switch list.
SWL_VISIBLE	Title is visible in the switch list.

fbJump Specifies a jump flag. If this field is SWL_JUMPABLE, the title participates in the jump sequence. If this field is SWL_NOTJUMPABLE, the title does not participate in the jump sequence.

szSwtitle[MAXNAMEL+ 1] Specifies the title of the program for the switch list. If the first character is zero, the program name will be used for the title.

fReserved This field is reserved.

See Also WinAddSwitchEntry, WinChangeSwitchEntry

■ SWENTRY

```
typedef struct _SWENTRY { /* swent */
    HSWITCH hswitch;
    SWCNTRL swctl;
} SWENTRY;
```

The SWENTRY structure contains information about a Task Manager switch entry.

Fields **hswitch** Identifies the entry.

swctl Specifies the SWCNTRL structure that contains information about the switch-entry program.

See Also WinQuerySwitchList, SWCNTRL

■ SWP

```
typedef struct _SWP { /* swp */
    USHORT fs;
    SHORT cy;
    SHORT cx;
    SHORT y;
    SHORT x;
    HWND hwndInsertBehind;
    HWND hwnd;
} SWP;
```

The SWP structure contains information about a window.

Fields

fs Specifies window-positioning options. This field can be one or more of the following values:

Value	Meaning
SWP_ACTIVATE	The window is activated and the focus is set to the window that lost the focus the last time the frame window was deactivated. The activated window cannot become the top window if it owns other frame windows.
SWP_DEACTIVATE	Deactivates the window (if it is the active window).
SWP_EXTSTATECHANGE	Used by the application to pass an additional flag to the portion of the code that is handling messages.
SWP_FOCUSACTIVATE	Specifies that a frame window is receiving the focus.
SWP_FOCUSDEACTIVATE	Specifies that a frame window is losing the focus.
SWP_HIDE	Specifies that the window is to be hidden when created.
SWP_MAXIMIZE	With SWP_MINIMIZE, causes a window to be minimized, maximized, or restored. SWP_MAXIMIZE and SWP_MINIMIZE are mutually exclusive. If either of these values is specified, then both SWP_MOVE and SWP_SIZE also must be specified. The WinSetWindowPos and WinSetMulti-WindowPos functions depend on the previous state of the window.
SWP_MINIMIZE	(See SWP_MAXIMIZE, above.)
SWP_MOVE	Changes the window position.
SWP_NOADJUST	Prevents the window from readjusting its position by not sending a WM_ADJUSTWINDOWPOS message while processing.
SWP_NOREDRAW	Does not redraw changes.
SWP_RESTORE	Restores a minimized or maximized window.
SWP_SHOW	Specifies that the window is to be shown when created.
SWP_SIZE	Changes the window size.
SWP_ZORDER	Changes the relative window placement.

cy Specifies the window height.

cx Specifies the window width.

y Specifies the position of the lower edge of the window.

x Specifies the position of the left edge of the window.

hwndInsertBehind Identifies the window behind which this window is placed.

hwnd Identifies the window.

See Also

WinFormatFrame, WinGetMaxPosition, WinQueryTaskSizePos, WinQuery-WindowPos, WinSetMultWindowPos, WinSetWindowPos

■ TRACKINFO

```
typedef struct _TRACKINFO { /* ti */
    SHORT cxBorder;
    SHORT cyBorder;
    SHORT cxGrid;
    SHORT cyGrid;
    SHORT cxKeyboard;
    SHORT cyKeyboard;
    RECTL rclTrack;
    RECTL rclBoundary;
    POINTL ptlMinTrackSize;
    POINTL ptlMaxTrackSize;
    USHORT fs;
    USHORT cxLeft;
    USHORT cyBottom;
    USHORT cxRight;
    USHORT cyTop;
} TRACKINFO;
```

The **TRACKINFO** structure contains information about a tracking rectangle used by the **WinTrackRect** function.

Fields

cxBorder Specifies the border width.

cyBorder Specifies the border height.

cxGrid Specifies the horizontal bounds of the tracking movements.

cyGrid Specifies the vertical bounds of the tracking movements.

cxKeyboard Specifies the amount of horizontal movement that occurs when the user presses the left arrow key.

cyKeyboard Specifies the amount of vertical movement that occurs when the user presses the left arrow key.

rclTrack Specifies the starting tracking rectangle. This is modified as the rectangle is tracked and holds the new tracking position when tracking is complete.

rclBoundary Specifies an absolute boundary for the tracking rectangle.

ptlMinTrackSize Specifies the minimum tracking size.

ptlMaxTrackSize Specifies the maximum tracking size.

fs Specifies tracking options. This field can be a combination of the following values:

Option	Meaning
TF_LEFT	Tracks the left side of the rectangle.
TF_TOP	Tracks the top side of the rectangle.
TF_RIGHT	Tracks the right side of the rectangle.

Option	Meaning
TF_BOTTOM	Tracks the bottom side of the rectangle.
TF_MOVE	Tracks all sides of the rectangle.
TF_POINTERPOS	Repositions the pointer according to the other options specified.
TF_LEFT	Vertically centers the pointer at the left of the tracking rectangle.
TF_TOP	Horizontally centers the pointer at the top of the tracking rectangle.
TF_RIGHT	Vertically centers the pointer at the right of the tracking rectangle.
TF_BOTTOM	Horizontally centers the pointer at the bottom of the tracking rectangle.
TF_MOVE	Centers the pointer in the tracking rectangle.
TF_GRID	Restricts tracking to the grid defined by <code>cxGrid</code> and <code>cyGrid</code> .
TF_STANDARD	The width, height, grid-width, and grid-height are all multiples of border-width and border-height.
TF_ALINBOUNDARY	Performs tracking so that no part of the tracking rectangle ever falls outside the bounding rectangle.

cxLeft This field is reserved.

cyBottom This field is reserved.

cxRight This field is reserved.

cyTop This field is reserved.

See Also

`WinTrackRect`

■ USERBUTTON

```
typedef struct _USERBUTTON { /* ubtn */
    HWND    hwnd;
    HPS     hps;
    USHORT  fsState;
    USHORT  fsStateOld;
} USERBUTTON;
```

The **USERBUTTON** structure is used by applications creating custom buttons. When a custom button is to be drawn, the owner receives a `WM_CONTROL` message with the low word of the first parameter equal to `BN_PAINT`. The second parameter is a pointer to the **USERBUTTON** structure that contains the information necessary for drawing the button.

Fields

hwnd Identifies the window.

hps Identifies the presentation space.

fsState Specifies the new state of the user button.

fsStateOld Specifies the previous state of the user button.

See Also WM_CONTROL

■ WNDPARAMS

```
typedef struct _WNDPARAMS { /* wprm */
    USHORT fsStatus;
    USHORT cchText;
    PSZ pszText;
    USHORT cbPresParams;
    PVOID pPresParams;
    USHORT cbCtlData;
    PVOID pCtlData;
} WNDPARAMS;
```

The WNDPARAMS structure contains information about a window.

Fields **fsStatus** Specifies the window parameters which are to be set or queried.

cchText Specifies the length of the window text.

pszText Points to the window text.

cbPresParams Specifies the length of the presentation parameters.

pPresParams Points to the presentation parameters. This field is currently not used.

cbCtlData Specifies the length of the class-specific data.

pCtlData Points to the class-specific data.

See Also WM_QUERYWINDOWPARAMS, WM_SETWINDOWPARAMS

■ XYWINSIZE

```
typedef struct _XYWINSIZE { /* xywin */
    SHORT x;
    SHORT y;
    SHORT cx;
    SHORT cy;
    SHORT fsWindow;
} XYWINSIZE;
```

The XYWINSIZE structure contains information about how a program is started.

Fields **x** Specifies the position of the left side of the window.

y Specifies the position of the lower side of the window.

cx Specifies the width of the window.

cy Specifies the height of the window.

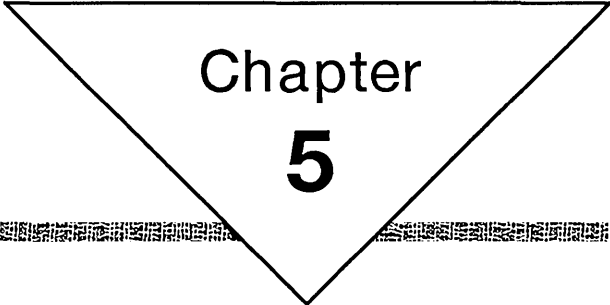
fsWindow Specifies various status flags. This field can be one or more of the following values:

Value	Meaning
XYF_INVISIBLE	The window is invisible.
XYF_MAXIMIZED	The window is maximized on start up.
XYF_MINIMIZED	The window is minimized on start up.
XYF_NOAUTOCLOSE	The window does not automatically close on exit. This field is ignored unless used by an advanced video-input-and-output (AVIO) application.
XYF_NORMAL	The window is visible.

See Also

PIBSTRUCT





Chapter
5

File Formats

5.1	Introduction	529
5.2	Font-File Format	529
5.3	Font Signature	530
5.4	Font Metrics	530
5.5	Font Character Definition	536
5.5.1	Font-Definition Header	536
5.5.2	Definition Data	538
5.5.2.1	Image Format	538
5.5.2.2	Outline Format	539
5.5.3	Kerning-Pair Table	539
5.6	Code-Page Font Support	540

5.1 Introduction

An MS OS/2 Presentation Manager font file is a set of structures containing the data for the characters in a font. Each font includes structures defining the font metrics and the character definitions data. MS OS/2 supports both image and vector fonts. Image fonts define their character glyphs using pel images. Vector fonts define their character glyphs using vector data that traces the outline of the character.

Although many types of fonts are possible, MS OS/2 Presentation Manager recognizes three standard font types. A type 1 font is a fixed-pitch font (that is, the widths of all characters in the font are the same). A type 2 font is a proportionally spaced font (each character has a unique width which is explicitly given in the font). A type 3 font is also a proportionally spaced font, but three values are used to specify the width of each character. These values, called the *a*-, *b*-, and *c*-space values, define the amount of space to move before drawing the character, the width of the character, and the amount of space to move after the character is drawn. Although the *b*-space is always positive, the *a*- and *c*-space values can be any integers. For all Presentation Manager font types, the height of all characters in the font is the same.

The MS OS/2 Presentation Manager font file is described in detail in the following sections.

5.2 Font-File Format

The MS OS/2 Presentation Manager font file consists of two parts. The first part contains the general attributes of the font, describing font features such as typeface style and point size. The second part contains the actual definitions of the font characters. The font file has the following general form:

```
start signature
font metrics
font character-definition header
definition data
kerning-pair table
end signature
```

The first part of a font file has the following form:

```
typedef struct _FOCAFONT { /* ff */
    FONTSIGNATURE    fsSignature;
    FOCAMETRICS      fmMetrics;
    FONTDEFINITIONHEADER fdDefinitions;
} FOCAFONT;
```

The following are the fields for the FOCAFONT structure:

fsSignature Specifies the font-file signature. This field contains the font-description string that identifies the operating system or context in which the font is used.

fmMetrics Specifies the names, dimensions, and attributes of the font.

fdDefinitions Specifies data defining the characters in the font.

5.3 Font Signature

The **FONTSIGNATURE** structure specifies the beginning and end of the font file. The **FONTSIGNATURE** structure has the following form:

```
typedef struct _FONTSIGNATURE { /* fs */
    ULONG    ulIdentity;
    ULONG    ulSize;
    CHAR     achSignature[12];
} FONTSIGNATURE;
```

The following are the fields for the **FONTSIGNATURE** structure:

ulIdentity Specifies the signature type. It can be one of the following values:

Value	Meaning
0xFFFFFFFFE	Signature starts a font file. The ulSize field must be 20, and the achSignature field is required.
0xFFFFFFFFF	Signature ends a font file. The ulSize field must be 8, and the achSignature field should not be given.

ulSize Specifies the length of the **FONTSIGNATURE** structure (in bytes).

achSignature[12] Specifies an array that contains the font-type description string. The string is the null-terminated string "OS/2 FONT".

5.4 Font Metrics

The font-metrics information defines the names, dimensions, and attributes of the font. The font-metrics structure, **FOCAMETRICS**, has the following form:

```
typedef struct _FOCAMETRICS { /* foca */
    ULONG    ulIdentity;
    ULONG    ulSize;
    CHAR     szFamilyname[32];
    CHAR     szFacename[32];
    SHORT    usRegistryId;
    SHORT    usCodePage;
    SHORT    yEmHeight;
    SHORT    yXHeight;
    SHORT    yMaxAscender;
    SHORT    yMaxDescender;
    SHORT    yLowerCaseAscent;
    SHORT    yLowerCaseDescent;
    SHORT    yInternalLeading;
    SHORT    yExternalLeading;
    SHORT    xAveCharWidth;
    SHORT    xMaxCharInc;
    SHORT    xEmInc;
    SHORT    yMaxBaselineExt;
    SHORT    sCharSlope;
    SHORT    sInlineDir;
    SHORT    sCharRot;
    USHORT   usWeightClass;
    USHORT   usWidthClass;
    SHORT    xDeviceRes;
    SHORT    yDeviceRes;
    SHORT    usFirstChar;
    SHORT    usLastChar;
    SHORT    usDefaultChar;
    SHORT    usBreakChar;
    SHORT    usNominalPointSize;
    SHORT    usMinimumPointSize;
```

```

SHORT    usMaximumPointSize;
SHORT    fsTypeFlags;
SHORT    fsDefn;
SHORT    fsSelectionFlags;
SHORT    fsCapabilities;
SHORT    ySubscriptXSize;
SHORT    ySubscriptYSize;
SHORT    ySubscriptXOffset;
SHORT    ySubscriptYOffset;
SHORT    ySuperscriptXSize;
SHORT    ySuperscriptYSize;
SHORT    ySuperscriptXOffset;
SHORT    ySuperscriptYOffset;
SHORT    yUnderscoreSize;
SHORT    yUnderscorePosition;
SHORT    yStrikeoutSize;
SHORT    yStrikeoutPosition;
SHORT    usKerningPairs;
SHORT    usKerningTracks;
PSZ      pszDeviceNameOffset;
} FOCAMETRICS;

```

The following are the fields for the **FOCAMETRICS** structure:

ulIdentity Specifies the identity of the font.

ulSize Specifies the size of the font.

szFamilyName[32] Specifies the family name of the font. Examples of common family names in MS OS/2 version 1.1 are Courier, Helvetica, and Times.

szFaceName[32] Specifies the typeface name of the font. Examples of common typeface names in MS OS/2 version 1.1 are Courier, Helvetica, and Times.

usRegistryID Specifies the registry number of the font.

usCodePage Identifies the code page that an application should use with the particular font.

yEmHeight Specifies the average height of uppercase characters. The height is measured in world coordinates from the baseline to the top of the character.

yXHeight Specifies the average height of lowercase characters. The height is measured in world coordinates from the baseline to the top of the character.

yMaxAscender Specifies the maximum height of any character in the font. The height is measured in world coordinates from the baseline to the top of the tallest character.

yMaxDescender Specifies the maximum depth of any character in the font. The depth is measured in world coordinates from the baseline to the bottom of the lowest character.

yLowerCaseAscent Specifies the maximum height of any lowercase character in the font. The height is measured in world coordinates from the baseline to the top of the ascender of the tallest lowercase character.

yLowerCaseDescent Specifies the maximum depth of any lowercase character in the font. The depth is measured in world coordinates from the baseline to the bottom of the descender of the lowest lowercase character.

yInternalLeading Specifies the amount of space reserved in the top of each character cell for accent marks. This metric is always given in world coordinates.

yExternalLeading Specifies the amount of space that should appear between adjacent rows of text. This metric is always given in world coordinates.

xAveCharWidth Specifies the average character width for characters in the font. The average character width is determined by multiplying the width of each lowercase character by a predetermined constant, adding the results, and then dividing by 1000. Letters and their predetermined constants are listed as follows:

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

xMaxCharInc Specifies the maximum increment between characters in the font.

xEmInc Specifies the width of an uppercase *M* in the font.

yMaxBaselineExtent Specifies the sum of the maximum ascender and maximum descender values.

sCharSlope Specifies the angle (in degrees and minutes) between a vertical line and the upright strokes in characters in the font. The first 9 bits of this value contain the degrees, the next 6 bits contain the minutes, and the last bit is reserved. The slope of characters in a normal font is zero; the slope of italic characters is nonzero.

sInlineDir Specifies an angle (in degrees and minutes, increasing clockwise) from the *x*-axis that the system uses when it draws a text string. The system draws each consecutive character from the text string in the inline direction. The inline-direction angle for a Swiss font is zero; the inline direction for a Hebrew font is 180.

Inline direction, like other rotations, is represented by a two-part unsigned discontinuous value. The first 9 bits are in the range 0 through 359, representing the number of degrees in the rotation. The next 6 bits are 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 invalid.

sCharRot Specifies the angle (in degrees and minutes) between the baseline of characters in the font and the *x*-axis. This is the angle assigned by the font designer.

usWeightClass Specifies the thickness of the strokes that form the characters in the font. This field can be one of the following values:

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

usWidthClass Specifies the relative-aspect ratio of characters in the font in relation to the normal-aspect ratio for a font of this type. The possible values are listed here:

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

xDeviceRes Specifies the horizontal resolution of the target device for which the font was originally designed. This value is given in pels per inch.

yDeviceRes Specifies the vertical resolution of the target device for which the font was originally designed. This value is given in pels per inch.

usFirstChar Specifies the code point for the first character in the font.

usLastChar Specifies the code point for the last character in the font. This code point is an offset from the **usFirstChar** value. All code points between the first and last character specified must be supported by the font.

usDefaultChar Specifies the code point for the default character in the font. This code point is an offset from the **usFirstChar** value. The default character is the character the system uses when an application specifies a code point that is out of the range of a font's code page.

usBreakChar Specifies the code point for the space character in the font. This code point is an offset from the **usFirstChar** value.

usNominalPointSize Specifies the height of the font in decipoints (1/720 inch). The nominal point size is the point size in which the font was designed to be drawn.

usMinimumPointSize Specifies the minimum height of the font (in decipoints). A font should not be reduced to a size smaller than this value.

usMaximumPointSize Specifies the maximum height of the font (in decipoints). A font should not be increased to a size larger than this value.

fsTypeFlags Specifies whether the font is proportional or fixed, whether it is licensed or not licensed, and whether it consumes more or less than 64K of memory. This field can be a combination of the following values:

Value	Meaning
0x0001	Specifies a fixed-pitch font. If not given, the font is proportionally spaced.
0x0002	Specifies a licensed font. If not given, the font is not licensed.
0x8000	Specifies a font that requires more than 64K of memory. If not given, the font requires 64K or less.

All other values are reserved. The remaining bits in the field must be set to zero.

fsDefn Specifies whether the font is an image or a vector font. This field can be a combination of the following values:

Value	Meaning
0x0001	Specifies a vector font. If not given, the font is an image font.
0x8000	Specifies an engine font. If not given, the font is a device font.

All other values are reserved. The remaining bits in the field must be set to zero.

fsSelectionFlags Specifies whether the font is normal or italic, whether it is underscored, whether it uses positive- or negative-image characters, whether it uses solid or outlined characters, and whether it uses overstruck characters. This field can be a combination of the following values:

Value	Meaning
0x0001	Characters are italic if this bit is set; otherwise, they are normal.
0x0002	Characters are underscored if this bit is set; otherwise, they are not underscored.
0x0004	Characters are drawn using negative images if this bit is set; otherwise, they are drawn using positive images.

Value	Meaning
0x0008	Characters are outlined if this bit is set; otherwise, they are solid.
0x0010	Characters are overstruck if this bit is set; otherwise, they are not overstruck.

All other values are reserved. Remaining bits in the field must be set to zero.

fsCapabilities Specifies whether the characters in this font can be mixed with graphics. If bit 0 of this field is set, the characters cannot be mixed with graphics; otherwise, they can be mixed with graphics. All other bits of this field are reserved and must be set to zero.

ySubscriptXSize Specifies the horizontal size (in world coordinates) for subscripts in the font.

ySubscriptYSize Specifies the vertical size (in world coordinates) for subscripts in the font.

ySubscriptXOffset Specifies the horizontal offset from the left edge of the character cell for subscripts in the font.

ySubscriptYOffset Specifies the vertical offset from the character-cell baseline for subscripts in the font.

ySuperscriptXSize Specifies the horizontal size (in world coordinates) for superscripts in the font.

ySuperscriptYSize Specifies the vertical size (in world coordinates) for superscripts in the font.

ySuperscriptXOffset Specifies the horizontal offset from the left edge of the character cell for superscripts in the font.

ySuperscriptYOffset Specifies the vertical offset from the character-cell baseline for superscripts in the font.

yUnderscoreSize Specifies the width of the underscore (in world coordinates).

yUnderscorePosition Specifies the distance from the baseline to the underscore line (in world coordinates).

yStrikeoutSize Specifies the width of the overstrike (in world coordinates).

yStrikeoutPosition Specifies the position of the overstrike in relation to the baseline.

usKerningPairs Specifies the number of kerning pairs in the kerning-pair table for the font.

usKerningTracks Reserved; must be zero.

pszDeviceNameOffset Points to the offset from the beginning of the resource to a null-terminated string that specifies the name of the device.

5.5 Font Character Definition

The font character definition consists of the character-definition data and information specifying the format of the character-definition data. It consists of a font-definition header, followed by the character-definition data and a kerning-pair table (if necessary). The following sections describe each part of the font character definition.

5.5.1 Font-Definition Header

The font-definition header specifies the dimensions and attributes of the character-definition data. The header has the following form:

```
typedef struct _FONTDEFINITIONHEADER {
    ULONG    ulIdentity;
    ULONG    ulSize;
    SHORT    fsFontdef;
    SHORT    fsChardef;
    SHORT    usCellSize;
    SHORT    xCellWidth;
    SHORT    yCellHeight;
    SHORT    xCellIncrement;
    SHORT    xCellA;
    SHORT    xCellB;
    SHORT    xCellC;
    SHORT    pCellBaseOffset;
} FONTDEFINITIONHEADER;
```

The following are the fields for the **FONTDEFINITIONHEADER** structure:

ulIdentity Specifies the font-definition header identifier. This field must be 2.

ulSize Specifies the length of the header plus all width and offsets (in bytes).

fsFontdef Specifies the attributes of the font definition. This field is an array of bit flags specifying which information in the font-definition header applies to all characters. This field can be a combination of the following flags:

Flag	Description
0x0001	Specifies that the width for all characters is in the font-definition header. If not given, the individual character widths are in the definition data.
0x0002	Specifies that height for all characters is in the font-definition header. If not given, the individual character heights are in the definition data.
0x0004	Specifies that the character increment is the same as the character width. If not given, the character increment for all characters is in the font-definition header.
0x0008	Specifies that the <i>a</i> -space is the same as the <i>a</i> -space in the font-definition header. If not given, no <i>a</i> -space is defined.

Flag	Description
0x0010	Specifies that the <i>b</i> -space is the same as the <i>b</i> -space in the font-definition header. If not given, no <i>b</i> -space is defined.
0x0020	Specifies that the <i>c</i> -space is the same as the <i>c</i> -space in the font-definition header. If not given, no <i>c</i> -space is defined.
0x0040	Specifies that the baseline offset for all characters is in the font-definition header.

fsChardef Specifies the attributes of the character definition. This field is an array of bit flags specifying which information in the font-definition header applies to all characters. This field can be a combination of the following flags:

Flag	Description
0x0001	Specifies that the width for each character is in the definition data. If not given, the width for all characters is in the font-definition header.
0x0002	Specifies the height for each character is in the definition data. If not given, the height for all characters is in the font-definition header.
0x0004	Specifies the character increment for each character is in the definition data. If not given, the character increment for all characters is in the font-definition header.
0x0008	Specifies the <i>a</i> -space for each character is in the definition data. If not given, the <i>a</i> -space for all characters is in the font-definition header.
0x0010	Specifies the <i>b</i> -space for each character is in the definition data. If not given, the <i>b</i> -space for all characters is in the font-definition header.
0x0020	Specifies the <i>c</i> -space for each character is in the definition data. If not given, the <i>c</i> -space for all characters is in the font-definition header.
0x0040	Specifies that the baseline offset for each character is in the definition data. If not given, the baseline offset for all characters is in the font-definition header.
0x0080	Specifies the offset to the glyph data is in the definition data. If not given, no glyph offsets are defined.

usCellSize Specifies the length of each cell (in bytes). This value is 6 for type 1 and 2 fonts. It is 10 for type 3 fonts.

xCellWidth Specifies the width of the cell for each character. This applies to type 1 fonts only. This field is zero for type 2 and 3 fonts.

yCellHeight Specifies the height of the cell for each character. This applies to all font types.

xCellIncrement Specifies the width of the cell for each character. This applies to type 1 fonts only. This field is zero for type 2 and 3 fonts.

xCellA Specifies the *a*-space value for the font. This applies to type 3 fonts only. This field is zero for type 2 and 3 fonts.

xCellB Specifies the *b*-space value for the font. This applies to type 3 fonts only. This field is zero for type 2 and 3 fonts.

xCellC Specifies the *c*-space value for the font. This applies to type 3 fonts only. This field is zero for type 2 and 3 fonts.

pCellBaseOffset Specifies the baseline offset for the characters in the font. This applies to all font types.

5.5.2 Definition Data

The definition data defines the image of the characters in the font. The definition data has the following form:

```
struct _DEFINITIONDATA {
    ULONG ulIdentity;
    ULONG ulSize;
    BYTES abData[];
} DEFINITIONDATA;
```

The following are the fields for the **DEFINITIONDATA** structure:

ulIdentity Specifies the definition-data signature. It must be 2.

ulSize Specifies the length of the definition data (in bytes).

abData[] Specifies the data defining the characters. The content depends on the file format.

5.5.2.1 Image Format

For image fonts, characters are stored as individual bitmaps. Each bitmap starts on a byte boundary.

The number of bytes in each bitmap is the product of the character height and the character width expressed in bytes. If the character width is not a multiple of 8, the width must be rounded to the next highest byte value before multiplying.

The order of bytes in each bitmap defines one or more 8-bit-wide vertical column. The bitmap represents one or more consecutive 8-bit-wide columns; the left column is stored first. For each column, there are the same number of bytes as rows in the bitmap (as defined by the character height). The byte for the top

row is stored first. For example, the 15-bit-wide letter *H* shown below is stored in this order: A1, B1, C1, . . . , M1, A2, B2, . . . , M2.

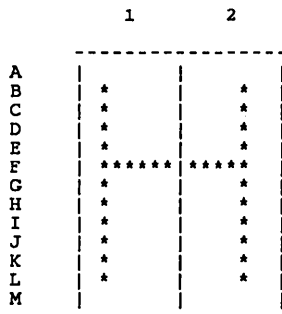


Image fonts contain a null character. The character-definition record for the null character occurs after the last non-null character. The format has (`usLastChar + 2`) total characters, although the null character is not counted in the range returned. The null character is composed of zeros and is always 8 pels wide.

5.5.2.2 Outline Format

For outline fonts, characters are stored as a set of graphics orders. The following is a list of the graphics orders permitted within a character definitions.

Graphics orders	Description
GLINE, GCLINE	Line.
GRLINE, GCRLINE	Relative line.
GBAR	Begin area.
GEAR	End area.
GFLT, GCFLT	Fillet.
GSCOL, GSECOL	Set color.
GSLT	Set line type.
GSLW	Set line width.
GESD	End symbol definition.

5.5.3 Kerning-Pair Table

The kerning-pair table is present in the font character definition if the `usKerningPairs` field in the FOCAMETRICS structure is 1. If it is present, the code points are words, not bytes. This table should be sorted by `usFirstChar` and `usSecondChar` order to allow binary searches. The kerning-pair table has the following form:

```
struct _KERNINGPAIRTABLE {
    ULONG ulIdentity;
    ULONG ulSize;
    KERNINGPAIRS akrnpr[];
} KERNINGPAIRTABLE;
```

The following are the fields for the **KERNINGPAIRTABLE** structure:

ulIdentity Specifies the kerning-pair-table signature. This value must be 3.

ulSize Specifies the length of the kerning-pair table (in bytes).

akrnpr[] Specifies an array of **KERNINGPAIRS** structures that contain the kerning data for each kerning pair. The **KERNINGPAIRS** structure has the following form:

```
typedef struct _KERNINGPAIRS {
    SHORT sFirstChar;
    SHORT sSecondChar;
    SHORT sKerningAmount;
} KERNINGPAIRS;
```

For a full description, see Chapter 4, “Types, Macros, Structures.”

5.6 Code-Page Font Support

MS OS/2 Presentation Manager supports multiple code pages for text input and output. For text output using fonts, a single font resource is used to support all code pages. The following section describes how code-page font support is provided and gives details about the font-resource format.

MS OS/2 Presentation Manager supports the following code pages for text output:

Number	Code page
500	EBCDIC CECP international version
437	Original PC ASCII code page
850	New PC ASCII code page supporting U.S. English and many European languages
860	PC ASCII for Portuguese
863	PC ASCII for French-Canadian
865	PC ASCII for Nordic languages

Most of the characters required by each code page are common—for example, the first 128 characters of all ASCII code pages are identical. This makes it possible for a single font definition to support all the code pages. Such a font contains an ordered list of all the character definitions (glyphs) used by the collection of code pages listed above.

To use such a font, all that is required is a mapping from the code points of the current code page to the glyphs of the font. Such mapping is provided for each code page. To make the translation of text strings from code page to code page easier, mapping from the universal set of characters to each code page is also provided.

The ordering of the characters is the same in all fonts containing multiple code pages. This means only one set of translation tables is necessary. The ordering of characters in these fonts is based on code page 850, with additional characters added beyond character 256. This makes mapping code page 850 to the fonts simple. It also provides simple mappings for the first 128 characters of all the ASCII code pages.

Table 5.1 shows the glyphs added to code page 850. The glyphs are listed in the order they occur in the font, starting at character 256.

Table 5.1 Additional Glyphs

Index number	Glyph ID	Symbol
256	SC040000	Cent sign
257	SC050000	Yen sign
258	SC060000	Pesetas sign
259	SM680000	Left-hand not sign
260	SF190000	Double line join single vertical
261	SF200000	Single line join double vertical
262	SF210000	Single line, upper-right corner double
263	SF220000	Double line, upper-right corner single
264	SF270000	Single line, lower-right corner double
265	SF280000	Double line, lower-right corner single
266	SF360000	Single vertical join double line
267	SF370000	Double vertical join single line
268	SF450000	Double horizontal join single line above
269	SF460000	Single horizontal join double line above
270	SF470000	Double horizontal join single line below
271	SF480000	Single horizontal join double line below
272	SF490000	Double line, lower-left corner single
273	SF500000	Single line, lower-left corner double
274	SF510000	Single line, upper-left corner double
275	SF520000	Double line, upper-left corner single
276	SF530000	Double vertical cross single
277	SF540000	Single vertical cross double
278	SF580000	Left hand half-block
279	SF590000	Right hand half-block
280	GA010000	Greek alpha lowercase
281	GG020000	Greek gamma uppercase
282	GP010000	Greek pi lowercase
283	GS020000	Greek sigma uppercase
284	GS010000	Greek sigma lowercase

Table 5.1 (Continued)

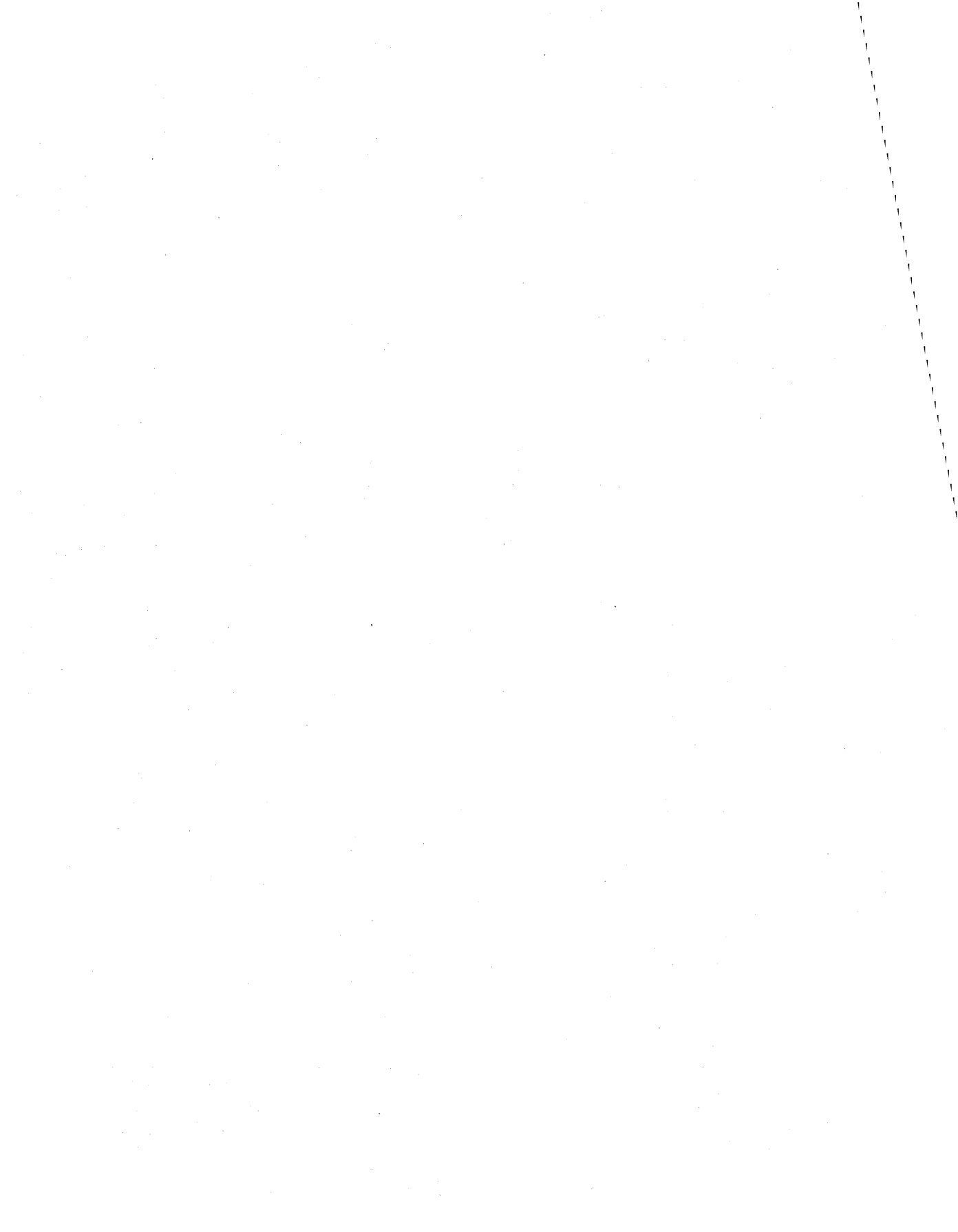
Index number	Glyph ID	Symbol
285	GT010000	Greek tau lowercase
286	GF020000	Greek phi uppercase
287	GT620000	Greek theta uppercase
288	GO320000	Greek omega uppercase
289	GD010000	Greek delta lowercase
290	SA450000	Infinity sign
291	GF010001	Greek phi lowercase
292	GE010000	Greek epsilon lowercase
293	SA380000	Mathematical intersection sign
294	SA480000	Mathematical equivalence sign
295	SA530000	Mathematical greater than or equal sign
296	SA520000	Mathematical less than or equal sign
297	SS260000	Mathematical integral sign, upper-half
298	SS270000	Mathematical integral sign, lower-half
299	SA700000	Mathematical approximately equal sign
300	SA790000	Mathematical product dot
301	SA800000	Mathematical square-root sign
302	LN011000	Superscript small <i>n</i>
512	SD110000	Dead acute accent
513	SD130000	Dead grave accent
514	SD150000	Dead circumflex accent
515	SD170000	Dead umlaut accent
516	SD190000	Dead tilde accent
517	SD410000	Dead cedilla accent
518	LE120000	Swiss <i>E</i> acute with CAPSLOCK
519	LE140000	Swiss <i>E</i> grave with CAPSLOCK
520	LA140000	Swiss <i>A</i> grave with CAPSLOCK
521	LU180000	Swiss <i>U</i> umlaut with CAPSLOCK
522	LO180000	Swiss <i>O</i> umlaut with CAPSLOCK
523	LA180000	Swiss <i>A</i> umlaut with CAPSLOCK

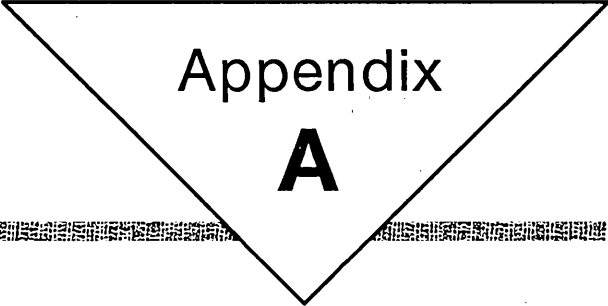
Glyphs with indexes 512 through 517 are used for dead accents. Glyphs with indexes 518 through 522 are required to support the Swiss keyboard and its three combination keys with code pages 437 and 850. Glyphs with indexes 768 through 1023 are reserved for DBCS.

The fonts have 300 characters in all (including the null character). This number of characters is supported by the font definition for both image and vector fonts.

Appendixes

Appendix A	Error Values	545
Appendix B	Device Capabilities	551





Error Values

A.1 Introduction	547
A.2 Errors	547

A.1 Introduction

This chapter contains the possible error values that can be returned by the MS OS/2 Presentation Manager functions. Before you can use these errors in your application, you must define the `INCL_WINERRORS`, `INCL_GPIERRORS`, or `SHL_ERRORS` constant before including the `os2.h` file. For example, to include errors for the `Gpi` functions, you must define the `INCL_GPIERRORS` constant, as shown in the following code:

```
#define INCL_GPIERRORS
#include <os2.h>
```

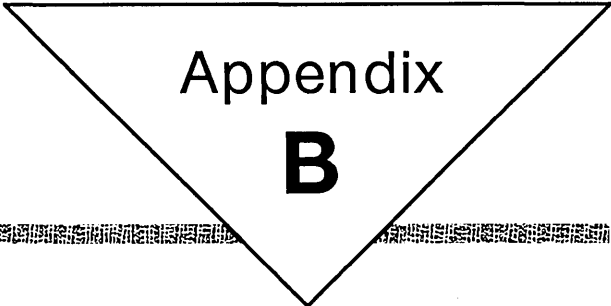
A.2 Errors

The following list gives the error values that may be returned by the `Dev`, `Gpi`, and `Win` functions. The error values are listed in numerical order, and the corresponding error constant is given for each value.

0000	PMERR_OK	1206	PMERR_INVALID_WINDOW
1001	PMERR_INVALID_HWND	1207	PMERR_INVALID_POST_MSG
1002	PMERR_INVALID_HMQ	1208	PMERR_INVALID_PARAMETERS
1003	PMERR_PARAMETER_OUT_OF_RANGE	1209	PMERR_INVALID_PROGRAM_TYPE
1004	PMERR_WINDOW_LOCK_UNDERFLOW	120A	PMERR_NOT_EXTENDED_FOCUS
1005	PMERR_WINDOW_LOCK_OVERFLOW	120B	PMERR_INVALID_SESSION_ID
1006	PMERR_BAD_WINDOW_LOCK_COUNT	120C	PMERR_SMG_INVALID_ICON_FILE
1007	PMERR_WINDOW_NOT_LOCKED	120D	PMERR_SMG_ICON_NOT_CREATED
1008	PMERR_INVALID_SELECTOR	120E	PMERR_SHL_DEBUG
1009	PMERR_CALL_FROM_WRONG_THREAD	1301	PMERR_OPENING_INI_FILE
100A	PMERR_RESOURCE_NOT_FOUND	1302	PMERR_INI_FILE_CORRUPT
100B	PMERR_INVALID_STRING_PARM	1303	PMERR_INVALID_PARM
100C	PMERR_INVALID_HHEAP	1304	PMERR_NOT_IN_IDX
100D	PMERR_INVALID_HEAP_POINTER	1306	PMERR_INI_WRITE_FAIL
100E	PMERR_INVALID_HEAP_SIZE_PARM	1307	PMERR_IDX_FULL
100F	PMERR_INVALID_HEAP_SIZE	1308	PMERR_INI_PROTECTED
1010	PMERR_INVALID_HEAP_SIZE_WORD	1309	PMERR_MEMORY_ALLOC
1011	PMERR_HEAP_OUT_OF_MEMORY	130A	PMERR_INI_INIT_ALREADY_DONE
1012	PMERR_HEAP_MAX_SIZE_REACHED	130B	PMERR_INVALID_INTEGER
1013	PMERR_INVALID_HATOMBL	130C	PMERR_INVALID_ASCIIZ
1014	PMERR_INVALID_ATOM	130D	PMERR_CAN_NOT_CALL_SPOOLER
1015	PMERR_INVALID_ATOM_NAME	1401	PMERR_WARNING_WINDOW_NOT_KILLED
1016	PMERR_INVALID_INTEGER_ATOM	1402	PMERR_ERROR_INVALID_WINDOW
1017	PMERR_ATOM_NAME_NOT_FOUND	1403	PMERR_ALREADY_INITIALIZED
1018	PMERR_QUEUE_TOO_LARGE	1405	PMERR_MSG_PROG_NO_MOU
1019	PMERR_INVALID_FLAG	1406	PMERR_MSG_PROG_NON_RECOV
101A	PMERR_INVALID_HACCEL	1407	PMERR_WINCONV_INVALID_PATH
101B	PMERR_INVALID_HPTRL	1408	PMERR_PL_NOT_INITIALISED
101C	PMERR_INVALID_HENUM	1409	PMERR_PL_NOT_INITIALISED
101D	PMERR_INVALID_SRC_CODEPAGE	140A	PMERR_NO_TASK_MANAGER
101E	PMERR_INVALID_DST_CODEPAGE	140B	PMERR_SAVE_NOT_IN_PROGRESS
101F	PMERR_UNKNOWN_COMPONENT_ID	140C	PMERR_NO_STACK_SPACE
1020	PMERR_UNKNOWN_ERROR_CODE	140D	PMERR_INVALID_COLOR_FIELD
1021	PMERR_SEVERITY_LEVELS	140E	PMERR_INVALID_COLOR_VALUE
1034	PMERR_INVALID_RESOURCE_FORMAT	140F	PMERR_COLOR_WRITE
1036	PMERR_NO_MSG_QUEUE	1501	PMERR_TARGET_FILE_EXISTS
1037	PMERR_WIN_DEBUGMSG	1502	PMERR_SOURCE_SAME_AS_TARGET
1038	PMERR_QUEUE_FULL	1503	PMERR_SOURCE_FILE_NOT_FOUND
1101	PMERR_INVALID_PIB	1504	PMERR_INVALID_NEW_PATH
1102	PMERR_INSUFF_SPACE_TO_ADD	1505	PMERR_TARGET_FILE_NOT_FOUND
1103	PMERR_INVALID_GROUP_HANDLE	1506	PMERR_INVALID_DRIVE_NUMBER
1104	PMERR_DUPLICATE_TITLE	1507	PMERR_NAME_TOO_LONG
1105	PMERR_INVALID_TITLE	1508	PMERR_NOT_ENOUGH_ROOM_ON_DISK
1106	PMERR_INVALID_TARGET_HANDLE	1509	PMERR_NOT_ENOUGH_MEM
1107	PMERR_HANDLE_NOT_IN_GROUP	150B	PMERR_LOG_DRV_DOES_NOT_EXIST
1108	PMERR_INVALID_PATH_STATEMENT	150C	PMERR_INVALID_DRIVE
1109	PMERR_NO_PROGRAM_FOUND	150D	PMERR_ACCESS_DENIED
110A	PMERR_INVALID_BUFFER_SIZE	150E	PMERR_NO_FIRST_SLASH
110B	PMERR_BUFFER_TOO_SMALL	150F	PMERR_READ_ONLY_FILE
110C	PMERR_PL_INITIALIZATION_FAIL	2001	PMERR_ALREADY_IN_AREA
110D	PMERR_CANT_DESTROY_SYS_GROUP	2002	PMERR_ALREADY_IN_ELEMENT
110E	PMERR_INVALID_TYPE_CHANGE	2003	PMERR_ALREADY_IN_PATH
110F	PMERR_INVALID_PROGRAM_HANDLE	2004	PMERR_ALREADY_IN_SEG
1110	PMERR_NOT_CURRENT_PL_VERSION	2005	PMERR_AREA_INCOMPLETE
1111	PMERR_INVALID_CIRCULAR_REF	2006	PMERR_BASE_ERROR
1112	PMERR_MEMORY_ALLOCATION_ERR	2007	PMERR_BITBLT_LENGTH_EXCEEDED
1113	PMERR_MEMORY_DEALLOCATION_ERR	2008	PMERR_BITMAP_IN_USE
1114	PMERR_TASK_HEADER_TOO_BIG	2009	PMERR_BITMAP_IS_SELECTED
1200	PMERR_DOS_ERROR	200A	PMERR_BITMAP_NOT_FOUND
1201	PMERR_NO_SPACE	200B	PMERR_BITMAP_NOT_SELECTED
1202	PMERR_INVALID_SWITCH_HANDLE	200C	PMERR_BOUNDS_OVERFLOW
1203	PMERR_NO_HANDLE	200D	PMERR_CALLED_SEG_IS_CHAINED
1204	PMERR_INVALID_PROCESS_ID	200E	PMERR_CALLED_SEG_IS_CURRENT
1205	PMERR_NOT_SHELL	200F	PMERR_CALLED_SEG_NOT_FOUND

2010	PMERR_CANNOT_DELETE_ALL_DATA	2050	PMERR_INV_CHAR_SHEAR_ATTR
2011	PMERR_CANNOT_REPLACE_ELEMENT_0	2051	PMERR_INV_CLIP_PATH_OPTIONS
2012	PMERR_COL_TABLE_NOT_REALIZABLE	2052	PMERR_INV_CODEPAGE
2013	PMERR_COL_TABLE_NOT_REALIZED	2053	PMERR_INV_COLOR_ATTR
2014	PMERR_COORDINATE_OVERFLOW	2054	PMERR_INV_COLOR_DATA
2015	PMERR_CORR_FORMAT_MISMATCH	2055	PMERR_INV_COLOR_FORMAT
2016	PMERR_DATA_TOO_LONG	2056	PMERR_INV_COLOR_INDEX
2017	PMERR_DC_IS_ASSOCIATED	2057	PMERR_INV_COLOR_OPTIONS
2018	PMERR_DESC_STRING_TRUNCATED	2058	PMERR_INV_COLOR_START_INDEX
2019	PMERR_DEVICE_DRIVER_ERROR_1	2059	PMERR_INV_COORD_OFFSET
201A	PMERR_DEVICE_DRIVER_ERROR_2	205A	PMERR_INV_COORD_SPACE
201B	PMERR_DEVICE_DRIVER_ERROR_3	205B	PMERR_INV_COORDINATE
201C	PMERR_DEVICE_DRIVER_ERROR_4	205C	PMERR_INV_CORRELATE_DEPTH
201D	PMERR_DEVICE_DRIVER_ERROR_5	205D	PMERR_INV_CORRELATE_TYPE
201E	PMERR_DEVICE_DRIVER_ERROR_6	205E	PMERR_INV_CURSOR_BITMAP
201F	PMERR_DEVICE_DRIVER_ERROR_7	205F	PMERR_INV_DC_DATA
2020	PMERR_DEVICE_DRIVER_ERROR_8	2060	PMERR_INV_DC_TYPE
2021	PMERR_DEVICE_DRIVER_ERROR_9	2061	PMERR_INV_DEVICE_NAME
2022	PMERR_DEVICE_DRIVER_ERROR_10	2062	PMERR_INV_DEV_MODES_OPTIONS
2023	PMERR_DEV_FUNC_NOT_INSTALLED	2063	PMERR_INV_DRAW_CONTROL
2024	PMERR_DOSOPEN_FAILURE	2064	PMERR_INV_DRAW_VALUE
2025	PMERR_DOSREAD_FAILURE	2065	PMERR_INV_DRAWING_MODE
2026	PMERR_DRIVER_NOT_FOUND	2066	PMERR_INV_DRIVER_DATA
2027	PMERR_DUP_SEG	2067	PMERR_INV_DRIVER_NAME
2028	PMERR_DYNAMIC_SEG_SEQ_ERROR	2068	PMERR_INV_DRAW_BORDER_OPTION
2029	PMERR_DYNAMIC_SEG_ZERO_INV	2069	PMERR_INV_EDIT_MODE
202A	PMERR_ELEMENT_INCOMPLETE	206A	PMERR_INV_ELEMENT_OFFSET
202B	PMERR_ESC_CODE_NOT_SUPPORTED	206B	PMERR_INV_ELEMENT_POINTER
202C	PMERR_EXCEEDS_MAX_SEG_LENGTH	206C	PMERR_INV_END_PATH_OPTIONS
202D	PMERR_FONT_AND_MODE_MISMATCH	206D	PMERR_INV_ESC_CODE
202E	PMERR_FONT_FILE_NOT_LOADED	206E	PMERR_INV_ESCAPE_DATA
202F	PMERR_FONT_NOT_LOADED	206F	PMERR_INV_EXTENDED_LCID
2030	PMERR_FONT_TOO_BIG	2070	PMERR_INV_FILL_PATH_OPTIONS
2031	PMERR_HARDWARE_INIT_FAILURE	2071	PMERR_INV_FIRST_CHAR
2032	PMERR_HBITMAP_BUSY	2072	PMERR_INV_FONT_ATTRS
2033	PMERR_HDC_BUSY	2073	PMERR_INV_FONT_FILE_DATA
2034	PMERR_HRGN_BUSY	2074	PMERR_INV_FOR_THIS_DC_TYPE
2035	PMERR_HUGE_FONTS_NOT_SUPPORTED	2075	PMERR_INV_FORMAT_CONTROL
2036	PMERR_ID_HAS_NO_BITMAP	2076	PMERR_INV_FORMS_CODE
2037	PMERR_IMAGE_INCOMPLETE	2077	PMERR_INV_FONTDEF
2038	PMERR_INCOMPAT_COLOR_FORMAT	2078	PMERR_INV_GEOM_LINE_WIDTH_ATTR
2039	PMERR_INCOMPAT_COLOR_OPTIONS	2079	PMERR_INV_GETDATA_CONTROL
203A	PMERR_INCOMPATIBLE_BITMAP	207A	PMERR_INV_GRAPHICS_FIELD
203B	PMERR_INCOMPATIBLE_METAFILE	207B	PMERR_INV_HBITMAP
203C	PMERR_INCORRECT_DC_TYPE	207C	PMERR_INV_HDC
203D	PMERR_INSUFFICIENT_DISK_SPACE	207D	PMERR_INV_HJOURNAL
203E	PMERR_INSUFFICIENT_MEMORY	207E	PMERR_INV_HMF
203F	PMERR_INV_ANGLE_PARM	207F	PMERR_INV_HPS
2040	PMERR_INV_ARC_CONTROL	2080	PMERR_INV_HRGN
2041	PMERR_INV_AREA_CONTROL	2081	PMERR_INV_ID
2042	PMERR_INV_ARC_POINTS	2082	PMERR_INV_IMAGE_DATA_LENGTH
2043	PMERR_INV_ATTR_MODE	2083	PMERR_INV_IMAGE_DIMENSION
2044	PMERR_INV_BACKGROUND_COL_ATTR	2084	PMERR_INV_IMAGE_FORMAT
2045	PMERR_INV_BACKGROUND_MIX_ATTR	2085	PMERR_INV_IN_AREA
2046	PMERR_INV_BITBLT_MIX	2086	PMERR_INV_IN_CALLED_SEG
2047	PMERR_INV_BITBLT_STYLE	2087	PMERR_INV_IN_CURRENT_EDIT_MODE
2048	PMERR_INV_BITMAP_DIMENSION	2088	PMERR_INV_IN_DRAW_MODE
2049	PMERR_INV_BOX_CONTROL	2089	PMERR_INV_IN_ELEMENT
204A	PMERR_INV_BOX_ROUNDING_PARM	208A	PMERR_INV_IN_IMAGE
204B	PMERR_INV_CHAR_ANGLE_ATTR	208B	PMERR_INV_IN_PATH
204C	PMERR_INV_CHAR_DIRECTION_ATTR	208C	PMERR_INV_IN_RETAIN_MODE
204D	PMERR_INV_CHAR_MODE_ATTR	208D	PMERR_INV_IN_SEG
204E	PMERR_INV_CHAR_POS_OPTIONS	208E	PMERR_INV_IN_VECTOR_SYMBOL
204F	PMERR_INV_CHAR_SET_ATTR	208F	PMERR_INV_INFO_TABLE

2090	PMERR_INV_JOURNAL_OPTION	20D0	PMERR_INV_TRANSFORM_TYPE
2091	PMERR_INV_KERNING_FLAGS	20D1	PMERR_INV_USAGE_PARM
2092	PMERR_INV_LENGTH_OR_COUNT	20D2	PMERR_INV_VIEWING_LIMITS
2093	PMERR_INV_LINE_END_ATTR	20D3	PMERR_INFILE_BUSY
2094	PMERR_INV_LINE_JOIN_ATTR	20D4	PMERR_INL_FUNC_DATA_TOO_LONG
2095	PMERR_INV_LINE_TYPE_ATTR	20D5	PMERR_KERNING_NOT_SUPPORTED
2096	PMERR_INV_LINE_WIDTH_ATTR	20D6	PMERR_LABEL_NOT_FOUND
2097	PMERR_INV_LOGICAL_ADDRESS	20D7	PMERR_MATRIX_OVERFLOW
2098	PMERR_INV_MARKER_BOX_ATTR	20D8	PMERR_METAFILE_INTERNAL_ERROR
2099	PMERR_INV_MARKER_SET_ATTR	20D9	PMERR_METAFILE_IN_USE
209A	PMERR_INV_MARKER_SYMBOL_ATTR	20DA	PMERR_METAFILE_LIMIT_EXCEEDED
209B	PMERR_INV_MATRIX_ELEMENT	20DB	PMERR_NAME_STACK_FULL
209C	PMERR_INV_MAX_HITS	20DC	PMERR_NOT_CREATED_BY_DEVOPENDC
209D	PMERR_INV_METAFILE	20DD	PMERR_NOT_IN_AREA
209E	PMERR_INV_METAFILE_LENGTH	20DE	PMERR_NOT_IN_DRAW_MODE
209F	PMERR_INV_METAFILE_OFFSET	20DF	PMERR_NOT_IN_ELEMENT
20A0	PMERR_INV_MICROPS_DRAW_CONTROL	20E0	PMERR_NOT_IN_IMAGE
20A1	PMERR_INV_MICROPS_FUNCTION	20E1	PMERR_NOT_IN_PATH
20A2	PMERR_INV_MICROPS_ORDER	20E2	PMERR_NOT_IN_RETAIN_MODE
20A3	PMERR_INV_MIX_ATTR	20E3	PMERR_NOT_IN_SEG
20A4	PMERR_INV_MODE_FOR_OPEN_DYN	20E4	PMERR_NO_BITMAP_SELECTED
20A5	PMERR_INV_MODE_FOR_REOPEN_SEG	20E5	PMERR_NO_CURRENT_ELEMENT
20A6	PMERR_INV_MODIFY_PATH_MODE	20E6	PMERR_NO_CURRENT_SEG
20A7	PMERR_INV_MULTIPLIER	20E7	PMERR_NO_METAFILE_RECORD_HANDLE
20A8	PMERR_INV_NESTED FIGURES	20E8	PMERR_ORDER_TOO_BIG
20A9	PMERR_INV_OR_INCOMPAT_OPTIONS	20E9	PMERR_OTHER_SET_ID_REFS
20AA	PMERR_INV_ORDER_LENGTH	20EA	PMERR_OVERRAN_SEG
20AB	PMERR_INV_ORDERING_PARM	20EB	PMERR_OWN_SET_ID_REFS
20AC	PMERR_INV_OUTSIDE_DRAW_MODE	20EC	PMERR_PATH_INCOMPLETE
20AD	PMERR_INV_PAGE_VIEWPORT	20ED	PMERR_PATH_LIMIT_EXCEEDED
20AE	PMERR_INV_PATH_ID	20EE	PMERR_PATH_UNKNOWN
20AF	PMERR_INV_PATH_MODE	20EF	PMERR_PEL_IS_CLIPPED
20B0	PMERR_INV_PATTERN_ATTR	20F0	PMERR_PEL_NOT_AVAILABLE
20B1	PMERR_INV_PATTERN_REF_PT_ATTR	20F1	PMERR_PRIMITIVE_STACK_EMPTY
20B2	PMERR_INV_PATTERN_SET_ATTR	20F2	PMERR_PROLOG_ERROR
20B3	PMERR_INV_PATTERN_SET_FONT	20F3	PMERR_PROLOG_SEG_ATTR_NOT_SET
20B4	PMERR_INV_PICK_APERTURE_OPTION	20F4	PMERR_PS_BUSY
20B5	PMERR_INV_PICK_APERTURE_POSN	20F5	PMERR_PS_IS_ASSOCIATED
20B6	PMERR_INV_PICK_APERTURE_SIZE	20F6	PMERR_RAM_INL_FILE_TOO_SMALL
20B7	PMERR_INV_PICK_NUMBER	20F7	PMERR_REALIZE_NOT_SUPPORTED
20B8	PMERR_INV_PLAY_METAFILE_OPTION	20F8	PMERR_REGION_IS_CLIP_REGION
20B9	PMERR_INV_PRIMITIVE_TYPE	20F9	PMERR_RESOURCE_DEPLETION
20BA	PMERR_INV_PS_SIZE	20FA	PMERR_SEG_AND_REFSEG_ARE_SAME
20BB	PMERR_INV_PUTDATA_FORMAT	20FB	PMERR_SEG_CALL_RECURSIVE
20BC	PMERR_INV_QUERY_ELEMENT_NO	20FC	PMERR_SEG_CALL_STACK_EMPTY
20BD	PMERR_INV_RECT	20FD	PMERR_SEG_CALL_STACK_FULL
20BE	PMERR_INV_REGION_CONTROL	20FE	PMERR_SEG_IS_CURRENT
20BF	PMERR_INV_REGION_MIX_MODE	20FF	PMERR_SEG_NOT_CHAINED
20C0	PMERR_INV_REPLACE_MODE_FUNC	2100	PMERR_SEG_NOT_FOUND
20C1	PMERR_INV_RESERVED_FIELD	2101	PMERR_SEG_STORE_LIMIT_EXCEEDED
20C2	PMERR_INV_RESET_OPTIONS	2102	PMERR_SETID_IN_USE
20C3	PMERR_INV_RGBCOLOR	2103	PMERR_SETID_NOT_FOUND
20C4	PMERR_INV_SCAN_START	2104	PMERR_STARTDOC_NOT_ISSUED
20C5	PMERR_INV_SEG_ATTR	2105	PMERR_STOP_DRAW_OCCURRED
20C6	PMERR_INV_SEG_ATTR_VALUE	2106	PMERR_TOO_MANY_METAFILES_IN_USE
20C7	PMERR_INV_SEG_CH_LENGTH	2107	PMERR_TRUNCATED_ORDER
20C8	PMERR_INV_SEG_NAME	2108	PMERR_UNCHAINED_SEG_ZERO_INV
20C9	PMERR_INV_SEG_OFFSET	2109	PMERR_UNSUPPORTED_ATTR
20CA	PMERR_INV_SETID	210A	PMERR_UNSUPPORTED_ATTR_VALUE
20CB	PMERR_INV_SETID_TYPE		
20CC	PMERR_INV_SET_VIEWPORT_OPTION		
20CD	PMERR_INV_SHARPNESS_PARM		
20CE	PMERR_INV_SOURCE_OFFSET		
20CF	PMERR_INV_STOP_DRAW_VALUE		



Device Capabilities

B.1	Introduction	553
B.2	About Device Capabilities	553

B.1 Introduction

This appendix describes the MS OS/2 constants that you can use to determine the capabilities of a device. These constants, used in the **DevQueryCaps** function, determine the type of device, its vertical and horizontal resolution, its color and font support, and several other characteristics.

B.2 About Device Capabilities

To determine a capability of a device, you use the appropriate constant (or range of constants) in the *lStartitem* and *allItems* parameters of **DevQueryCaps**. The following is a list, in numerical order, describing each of the constants:

CAPS_FAMILY Specifies one of five device types. It can be one of the following values:

Value	Meaning
OD_QUEUED	A device, such as a printer or plotter, for which to queue output.
OD_DIRECT	A device, such as a printer or plotter, for which to not queue output.
OD_INFO	Same as for OD_DIRECT, but used only to retrieve information (for example, font metrics). You can draw to a presentation space associated with such a device context, but you cannot update any output.
OD_METAFILE	A device context that is used to draw a metafile. The graphics field defines the area of interest within the metafile picture.
OD_MEMORY	A device context that is used to contain a bitmap.

CAPS_IO_CAPS Specifies the device input/output capability. The possible values are as follows:

Value	Meaning
1	Dummy device
2	Output
3	Input
4	Output and input

CAPS_TECHNOLOGY Specifies the technology. The possible values are as follows:

Value	Meaning
0	Unknown (for example, metafile)
1	Vector plotter
2	Raster display
3	Raster printer
4	Raster camera

CAPS_DRIVER_VERSION Specifies the device-driver version number.

CAPS_WIDTH Specifies the media width (for a full-screen maximized window on a display) in pels.

CAPS_HEIGHT Specifies the media depth (for a full-screen maximized window on a display) 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_IN_CHARS Specifies the media width (for a full-screen maximized window on a display) in character columns.

CAPS_HEIGHT_IN_CHARS Specifies the media depth (for a full-screen maximized window on a display) in character rows.

CAPS_HORIZONTAL_RESOLUTION Specifies the horizontal resolution (in pels per meter) of the device.

CAPS_VERTICAL_RESOLUTION Specifies the vertical resolution (in pels per meter) of the device.

CAPS_CHAR_WIDTH Specifies the default width (in pels) of the character box.

CAPS_CHAR_HEIGHT Specifies the default height (in pels) of the character box.

CAPS_SMALL_CHAR_WIDTH Specifies the default width (in pels) of the small character box. This value is zero if there is only one size of character box.

CAPS_SMALL_CHAR_HEIGHT Specifies the default height (in pels) of the small character box. This value is zero if there is only one size of character box.

CAPS_COLORS Specifies the 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 Specifies the preferred number of color planes for bitmaps used by the device.

CAPS_COLOR_BITCOUNT Specifies the preferred number of bits per pel (bitcount) for bitmaps used by the device.

CAPS_COLOR_TABLE_SUPPORT Specifies the color tables supported by the device. The bits are set as follows:

Bit	Setting
0	Set to 1 if an RGB color table can be loaded, with a minimum support of 8 bits each for red, green, and blue.
1	Set to 1 if a color table with other than 8 bits for each primary color can be loaded.
2	Set to 1 if true mixing occurs when the logical color table has been realized, provided that the size of the logical color table is not greater than the number of distinct colors supported.
3	Set to 1 if a loaded color table can be realized.

CAPS_MOUSE_BUTTONS Specifies the number of available mouse or drawing-tablet buttons.

CAPS_FOREGROUND_MIX_SUPPORT Specifies the foreground mix-mode support. The possible values are as follows:

Value	Meaning
1	OR
2	Overpaint
8	XOR
16	Leave-alone
32	AND
64	Mixes 7 through 17

The value returned is the sum of the values appropriate to the mix modes that are supported. A device capable of supporting the OR mix mode must, as a minimum, return 19 (1 + 2 + 16), signifying support for the mandatory mix modes OR, "overpaint," and "leave-alone." Note that these numbers correspond to the decimal representation of a bit string that is 7 bits long, with each bit set to 1 if the appropriate mode is supported.

CAPS_BACKGROUND_MIX_SUPPORT Specifies the background mix-mode support. The possible values are as follows:

Value	Meaning
1	OR
2	Overpaint
8	XOR
16	Leave-alone

The value returned is the sum of the values appropriate to the mix modes that are supported. A device must, as a minimum, return 18 (2 + 16), signifying support for the mandatory background mix modes "overpaint" and "leave-alone." Note that these numbers correspond to the decimal representation of a bit string that is 5 bits long, with each bit set to 1 if the appropriate mode is supported.

CAPS_VIO_LOADABLE_FONTS Specifies the number of fonts that may be loaded for Vio functions.

CAPS_WINDOW_BYTE_ALIGNMENT Specifies whether the client area of video-input-and-output windows should be byte-aligned. The possible values are as follows:

Value	Meaning
0	Must be byte-aligned.
1	More efficient if byte-aligned, but not required.
2	Does not matter whether byte-aligned.

CAPS_BITMAP_FORMATS Specifies the number of bitmap formats supported by the device.

CAPS_RASTER_CAPS Specifies the raster capabilities of the device.

CAPS_MARKER_HEIGHT Specifies the default depth (in pels) of the marker box.

CAPS_MARKER_WIDTH Specifies the default width (in pels) of the marker box.

CAPS_DEVICE_FONTS Specifies the number of device-specific fonts.

CAPS_GRAPHICS_SUBSET Specifies the graphics-drawing subset supported.

CAPS_GRAPHICS_VERSION Specifies the graphics-architecture version supported.

CAPS_GRAPHICS_VECTOR_SUBSET Specifies the graphics-vector-drawing subset supported.

CAPS_DEVICE_WINDOWING Specifies whether the device supports windows. Bit 0 is 1 if the device supports windows. Other bits are reserved and must be zero.

CAPS_ADDITIONAL_GRAPHICS Specifies additional graphics support. Bit 0 is 1 if the device supports geometric-line types. Bit 1 is 1 if the device supports kerning. Other bits are reserved and must be zero.

CAPS_PHYS_COLORS Specifies the maximum number of distinct colors that can be specified for the device.

CAPS_COLOR_INDEX Specifies the maximum logical-color-table index supported by the device. This value must be at least 7. For EGA and VGA device drivers, the value is 63.

CAPS_GRAPHICS_CHAR_WIDTH Specifies the graphics-character width.

CAPS_GRAPHICS_CHAR_HEIGHT Specifies the graphics-character height.

CAPS_HORIZONTAL_FONT_RES Specifies the optimal horizontal resolution of the font used by the device.

CAPS_VERTICAL_FONT_RES Specifies the optimal vertical resolution for the font used by the device.

Index

- A**
- ACCEL, 486
 - ACCELTABLE, 486
 - ARCPARAMS, 487
 - AREABUNDLE, 487
- B**
- BITMAPINFO, 488
 - BITMAPINFOHEADER, 489
 - Bit masks, 7
 - BM_CLICK, 395
 - BM_QUERYCHECK, 395
 - BM_QUERYCHECKINDEX, 395
 - BM_QUERYHILITE, 396
 - BM_SETCHECK, 396
 - BM_SETDEFAULT, 396
 - BM_SETHILITE, 397
 - BTNCDATA, 489
- C**
- Calling conventions, 5
 - CATCHBUF, 489
 - CHAR1FROMMP, 474
 - CHAR2FROMMP, 474
 - CHAR3FROMMP, 474
 - CHAR4FROMMP, 474
 - CHARBUNDLE, 490
 - CHARMSG, 475
 - C-language format, 5
 - CLASSINFO, 492
 - Code-page font support, 540–542
 - COMMANDMSG, 475
 - Constant names, 10
 - CREATESTRUCT, 493
 - CURSORINFO, 494
- D**
- DDEINIT, 495
 - DDESTRUCT, 495
 - DDES_PABDATA, 476
 - DDES_PSZITEMNAME, 476
 - DevCloseDC, 15
 - DevEscape, 15
 - Device capabilities, 553–557
 - DevOpenDC, 20
 - DEVOPENSTRUC, 496
 - DevPostDeviceModes, 21
 - DevQueryCaps, 22
 - DevQueryDeviceNames, 22
 - DevQueryHardcopyCaps, 23
 - DLGTEMPLATE, 497
 - DLGTITEM, 498
 - DRIVDATA, 498
- E**
- EM_CLEAR, 397
 - EM_COPY, 397
 - EM_CUT, 398
 - EM_PASTE, 398
 - EM_QUERYCHANGED, 398
 - EM_QUERYFIRSTCHAR, 399
 - EM_QUERYSEL, 399
 - EM_SETFIRSTCHAR, 399
 - EM_SETSEL, 400
 - EM_SETTEXTLIMIT, 400
 - ERRINFO, 499
 - Errors, 547–550
 - ERRORIDERROR, 476
 - ERRORIDSEV, 477
- F**
- FATTRS, 499
 - Field names, 8
 - File formats, 529–542
 - FIXEDFRAC, 477
 - FIXEDINT, 477
 - Font
 - character definition, 536
 - code-page support, 540–542
 - file format, 529
 - metrics, 530–535
 - signature, 530
 - FONTMETRICS, 501
 - FRAMECDATA, 505
 - Functions directory, 13–389
- G**
- GpiAssociate, 25
 - GpiBeginArea, 26
 - GpiBeginElement, 27
 - GpiBeginPath, 27
 - GpiBitBlt, 28
 - GpiBox, 32
 - GpiCallSegmentMatrix, 33
 - GpiCharString, 35
 - GpiCharStringAt, 36
 - GpiCharStringPos, 37
 - GpiCharStringPosAt, 38
 - GpiCloseFigure, 40
 - GpiCloseSegment, 40
 - GpiCombineRegion, 41
 - GpiComment, 42
 - GpiConvert, 43
 - GpiCopyMetaFile, 44
 - GpiCorrelateChain, 44
 - GpiCorrelateFrom, 46
 - GpiCorrelateSegment, 47
 - GpiCreateBitmap, 49
 - GpiCreateLogColorTable, 51
 - GpiCreateLogFont, 53
 - GpiCreatePS, 55
 - GpiCreateRegion, 57
 - GpiDeleteBitmap, 58
 - GpiDeleteElement, 59
 - GpiDeleteElementRange, 59
 - GpiDeleteElementsBetweenLabels, 60
 - GpiDeleteMetaFile, 61
 - GpiDeleteSegment, 61
 - GpiDeleteSegments, 62
 - GpiDeleteSetId, 62
 - GpiDestroyPS, 63
 - GpiDestroyRegion, 64
 - GpiDrawChain, 64
 - GpiDrawDynamics, 65
 - GpiDrawFrom, 65
 - GpiDrawSegment, 66
 - GpiElement, 67
 - GpiEndArea, 68
 - GpiEndElement, 68
 - GpiEndPath, 69
 - GpiEqualRegion, 70
 - GpiErase, 70
 - GpiErrorSegmentData, 71
 - GpiExcludeClipRectangle, 72
 - GpiFillPath, 72
 - GpiFullArc, 73
 - GpiGetData, 74
 - GpiImage, 76
 - GpiIntersectClipRectangle, 77
 - GpiLabel, 78
 - GpiLine, 78
 - GpiLoadBitmap, 79
 - GpiLoadFonts, 80
 - GpiLoadMetaFile, 81
 - GpiMarker, 81
 - GpiModifyPath, 82

- GpiMove, 83
- GpiOffsetClipRegion, 84
- GpiOffsetElementPointer, 84
- GpiOffsetRegion, 85
- GpiOpenSegment, 86
- GpiPaintRegion, 87
- GpiPartialArc, 88
- GpiPlayMetaFile, 89
- GpiPointArc, 93
- GpiPolyFillet, 93
- GpiPolyFilletSharp, 95
- GpiPolyLine, 96
- GpiPolyMarker, 97
- GpiPolySpline, 98
- GpiPop, 99
- GpiPtInRegion, 100
- GpiPtVisible, 100
- GpiPutData, 101
- GpiQueryArcParams, 103
- GpiQueryAttrMode, 103
- GpiQueryAttrs, 104
- GpiQueryBackColor, 105
- GpiQueryBackMix, 106
- GpiQueryBitmapBits, 106
- GpiQueryBitmapDimension, 108
- GpiQueryBitmapHandle, 108
- GpiQueryBitmapParameters, 109
- GpiQueryBoundaryData, 109
- GpiQueryCharAngle, 110
- GpiQueryCharBox, 111
- GpiQueryCharDirection, 111
- GpiQueryCharMode, 112
- GpiQueryCharSet, 112
- GpiQueryCharShear, 112
- GpiQueryCharStringPos, 113
- GpiQueryCharStringPosAt, 114
- GpiQueryClipBox, 115
- GpiQueryClipRegion, 116
- GpiQueryColor, 116
- GpiQueryColorData, 117
- GpiQueryColorIndex, 118
- GpiQueryCp, 118
- GpiQueryCurrentPosition, 119
- GpiQueryDefaultViewMatrix, 119
- GpiQueryDefCharBox, 120
- GpiQueryDevice, 120
- GpiQueryDeviceBitmapFormats, 121
- GpiQueryDrawControl, 122
- GpiQueryDrawingMode, 122
- GpiQueryEditMode, 123
- GpiQueryElement, 123
- GpiQueryElementPointer, 124
- GpiQueryElementType, 125
- GpiQueryFontFileDescriptions, 126
- GpiQueryFontMetrics, 127
- GpiQueryFonts, 128
- GpiQueryGraphicsField, 130
- GpiQueryInitialSegmentAttrs, 131
- GpiQueryKerningPairs, 131
- GpiQueryLineEnd, 132
- GpiQueryLineJoin, 132
- GpiQueryLineType, 133
- GpiQueryLineWidth, 133
- GpiQueryLineWidthGeom, 133
- GpiQueryLogColorTable, 134
- GpiQueryMarker, 135
- GpiQueryMarkerBox, 135
- GpiQueryMarkerSet, 136
- GpiQueryMetaFileBits, 136
- GpiQueryMetaFileLength, 137
- GpiQueryMix, 137
- GpiQueryModelTransformMatrix, 138
- GpiQueryNearestColor, 138
- GpiQueryNumberSetIds, 139
- GpiQueryPageViewport, 139
- GpiQueryPattern, 140
- GpiQueryPatternRefPoint, 140
- GpiQueryPatternSet, 141
- GpiQueryPel, 141
- GpiQueryPickAperturePosition, 142
- GpiQueryPickApertureSize, 142
- GpiQueryPS, 143
- GpiQueryRealColors, 143
- GpiQueryRegionBox, 145
- GpiQueryRegionRects, 145
- GpiQueryRGCColor, 146
- GpiQuerySegmentAttrs, 147
- GpiQuerySegmentNames, 148
- GpiQuerySegmentPriority, 149
- GpiQuerySegmentTransformMatrix, 150
- GpiQuerySetIds, 150
- GpiQueryStopDraw, 152
- GpiQueryTag, 152
- GpiQueryTextBox, 153
- GpiQueryViewingLimits, 154
- GpiQueryViewingTransformMatrix, 155
- GpiQueryWidthTable, 155
- GpiRealizeColorTable, 156
- GpiRectInRegion, 157
- GpiRectVisible, 157
- GpiRemovedDynamics, 158
- GpiResetBoundaryData, 159
- GpiResetPS, 159
- GpiRestorePS, 160
- GpiSaveMetaFile, 161
- GpiSavePS, 162
- GpiSetArcParams, 162
- GpiSetAttrMode, 163
- GpiSetAttrs, 164
- GpiSetBackColor, 166
- GpiSetBackMix, 167
- GpiSetBitmap, 168
- GpiSetBitmapBits, 169
- GpiSetBitmapDimension, 170
- GpiSetBitmapId, 170
- GpiSetCharAngle, 171
- GpiSetCharBox, 172
- GpiSetCharDirection, 172
- GpiSetCharMode, 173
- GpiSetCharSet, 174
- GpiSetCharShear, 174
- GpiSetClipPath, 175
- GpiSetClipRegion, 176
- GpiSetColor, 177
- GpiSetCp, 178
- GpiSetCurrentPosition, 179
- GpiSetDefaultViewMatrix, 179
- GpiSetDrawControl, 180
- GpiSetDrawingMode, 182
- GpiSetEditMode, 183
- GpiSetElementPointer, 184
- GpiSetElementPointerAtLabel, 184
- GpiSetGraphicsField, 185
- GpiSetInitialSegmentAttrs, 185
- GpiSetLineEnd, 187
- GpiSetLineJoin, 188
- GpiSetLineType, 188
- GpiSetLineWidth, 189
- GpiSetLineWidthGeom, 190
- GpiSetMarker, 191
- GpiSetMarkerBox, 192
- GpiSetMarkerSet, 192
- GpiSetMetaFileBits, 193
- GpiSetMix, 194
- GpiSetModelTransformMatrix, 195
- GpiSetPageViewport, 196
- GpiSetPattern, 197
- GpiSetPatternRefPoint, 198
- GpiSetPatternSet, 199
- GpiSetPel, 200
- GpiSetPickAperturePosition, 200
- GpiSetPickApertureSize, 201
- GpiSetPS, 201
- GpiSetRegion, 202
- GpiSetSegmentAttrs, 203
- GpiSetSegmentPriority, 204
- GpiSetSegmentTransformMatrix, 205
- GpiSetStopDraw, 206
- GpiSetTag, 207
- GpiSetViewingLimits, 208

GpiSetViewingTransformMatrix, 209
 GpiStrokePath, 210
 GpiUnloadFonts, 210
 GpiUnrealizeColorTable, 211
 GpiWCBitBlt, 211
 GRADIENTL, 506

H

HCINFO, 506
 HPROGARRAY, 507
 HWNDFROMMP, 477
 IMAGEBUNDLE, 507

K

KERNINGPAIRS, 507

L

LINEBUNDLE, 508
 LM_DELETEALL, 401
 LM_DELETEITEM, 401
 LM_INSERTITEM, 401
 LM_QUERYITEMCOUNT, 402
 LM_QUERYITEMHANDLE, 402
 LM_QUERYITEMTEXT, 402
 LM_QUERYITEMTEXTLENGTH, 403
 LM_QUERYSELECTION, 403
 LM_QUERYTOPINDEX, 404
 LM_SEARCHSTRING, 404
 LM_SELECTITEM, 405
 LM_SETITEMHANDLE, 405
 LM_SETITEMHEIGHT, 405
 LM_SETITEMTEXT, 406
 LM_SETTOPINDEX, 406
 LONGFROMMP, 478
 LONGFROMMR, 478

M

Macros, 474-484
 MAKEBERRORID, 478
 MAKEFIXED, 479
 MAKEINTATOM, 479
 MARKERBUNDLE, 508
 MATRIXLF, 509
 MENUITEM, 510
 Messages directory, 393-467
 MM_DELETEITEM, 406
 MM_ENDMENU MODE, 407
 MM_INSERTITEM, 407
 MM_ISITEMVALID, 408

MM_ITEMIDFROMPOSITION, 408
 MM_ITEMPOSITIONFROMID, 408
 MM_QUERYITEM, 409
 MM_QUERYITEMATTR, 409
 MM_QUERYITEMCOUNT, 410
 MM_QUERYITEMTEXT, 410
 MM_QUERYITEMTEXTLENGTH, 411
 MM_QUERYSELITEMID, 411
 MM_REMOVEITEM, 411
 MM_SELECTITEM, 412
 MM_SETITEM, 412
 MM_SETITEMATTR, 413
 MM_SETITEMHANDLE, 414
 MM_SETITEMTEXT, 414
 MM_STARTMENU MODE, 414
 MOUSEMSG, 479
 MOVBLOCKHDR, 511
 MPFROM2SHORT, 479
 MPFROMCHAR, 480
 MPFROMHWN D, 480
 MPFROMLONG, 480
 MPFROMP, 480
 MPFROMSH2CH, 480
 MPFROMSHORT, 481
 MQINFO, 511
 MRFROM2SHORT, 481
 MRFROMLONG, 481
 MRFROMP, 481
 MRFROMSHORT, 482

N

Naming conventions, 7
 Notational conventions, 10

O

OWNERITEM, 512

P

Parameter names, 8
 PDDEITOSEL, 482
 PDDESTOSEL, 482
 PIBSTRUCT, 513
 POINTERINFO, 513
 POINTFX, 514
 POINTL, 514
 Prefixes, 8
 PROGRAMENTRY, 514
 PROGTYPE, 515
 PVOIDFROMMP, 482

PVOIDFROMMR, 483

Q

QMSG, 515

R

RECTFX, 516
 RECTL, 516
 RGB, 517
 RGNRECT, 517

S

SBCDATA, 518
 SBM_QUERYPOS, 415
 SBM_QUERYRANGE, 415
 SBM_SETPOS, 415
 SBM_SETSCROLLBAR, 416
 SELTOPDDES, 483
 SETMEMBACKPTR, 483
 SHORT1FROMMP, 483
 SHORT1FROMMR, 484
 SHORT2FROMMP, 484
 SHORT2FROMMR, 484
 SIZEF, 518
 SIZEL, 518
 SMHSTRUCT, 519
 SM_QUERYHANDLE, 416
 SM_SETHANDLE, 416
 Structures, 7, 485-525
 SWCNTRL, 519
 SWENTRY, 520
 SWP, 520

T

TBM_QUERYHILITE, 417
 TBM_SETHILITE, 417
 TRACKINFO, 522
 TRACKINFO, 523
 Types, 9-10, 472-473

U

USERBUTTON, 523

W

WinAddAtom, 215
 WinAddProgram, 215
 WinAddSwitchEntry, 216
 WinAlarm, 217
 WinAllocMem, 218

- WinAvailMem, 219
- WinBeginEnumWindows, 220
- WinBeginPaint, 220
- WinBroadcastMsg, 221
- WinCalcFrameRect, 222
- WinCallMsgFilter, 223
- WinCancelShutdown, 224
- WinCatch, 224
- WinChangeSwitchEntry, 225
- WinCloseClipbrd, 226
- WinCompareStrings, 226
- WinCopyAccelTable, 227
- WinCopyRect, 228
- WinCpTranslateChar, 228
- WinCpTranslateString, 229
- WinCreateAccelTable, 229
- WinCreateAtomTable, 230
- WinCreateCursor, 230
- WinCreateDlg, 231
- WinCreateFrameControls, 232
- WinCreateGroup, 233
- WinCreateHeap, 234
- WinCreateMenu, 236
- WinCreateMsgQueue, 236
- WinCreatePointer, 237
- WinCreateStdWindow, 238
- WinCreateWindow, 242
- WinDdeInitiate, 244
- WinDdePostMsg, 245
- WinDdeRespond, 246
- WinDefAVioWindowProc, 246
- WinDefDlgProc, 247
- WinDefWindowProc, 247
- WinDeleteAtom, 248
- WinDestroyAccelTable, 249
- WinDestroyAtomTable, 249
- WinDestroyCursor, 249
- WinDestroyHeap, 250
- WinDestroyMsgQueue, 250
- WinDestroyPointer, 251
- WinDestroyWindow, 251
- WinDismissDlg, 252
- WinDispatchMsg, 253
- WinDlgBox, 253
- WinDrawBitmap, 254
- WinDrawBorder, 255
- WinDrawPointer, 257
- WinDrawText, 257
- WinEmptyClipbrd, 260
- WinEnablePhysInput, 260
- WinEnableWindow, 260
- WinEnableWindowUpdate, 261
- WinEndEnumWindows, 261
- WinEndPaint, 262
- WinEnumClipbrdFmts, 262
- WinEnumDlgItem, 263
- WinEqualRect, 264
- WinExcludeUpdateRegion, 264
- WinFillRect, 265
- WinFindAtom, 266
- WinFlashWindow, 266
- WinFocusChange, 267
- WinFormatFrame, 268
- WinFreeErrorInfo, 269
- WinFreeMem, 270
- WinGetClipPS, 270
- WinGetCurrentTime, 271
- WinGetErrorInfo, 272
- WinGetKeyState, 272
- WinGetLastError, 273
- WinGetMaxPosition, 273
- WinGetMinPosition, 274
- WinGetMsg, 274
- WinGetNextWindow, 276
- WinGetPhysKeyState, 276
- WinGetPS, 277
- WinGetScreenPS, 278
- WinGetSysBitmap, 278
- WinInflateRect, 279
- WinInitialize, 280
- WinInSendMessage, 280
- WinIntersectRect, 281
- WinInvalidateRect, 281
- WinInvalidateRegion, 282
- WinInvertRect, 283
- WinIsChild, 283
- WinIsPhysInputEnabled, 284
- WinIsRectEmpty, 284
- WinIsThreadActive, 284
- WinIsWindow, 285
- WinIsWindowEnabled, 285
- WinIsWindowVisible, 285
- WinLoadAccelTable, 286
- WinLoadDlg, 286
- WinLoadMenu, 287
- WinLoadMessage, 288
- WinLoadPointer, 288
- WinLoadString, 289
- WinLockHeap, 290
- WinLockVisRegions, 290
- WinLockWindow, 291
- WinLockWindowUpdate, 291
- WinMapDlgPoints, 292
- WinMapWindowPoints, 292
- WinMessageBox, 293
- WinMsgMuxSemWait, 296
- WinMsgSemWait, 297
- WinMultWindowFromIDs, 298
- WinNextChar, 298
- WinOffsetRect, 299
- WinOpenClipbrd, 299
- WinOpenWindowDC, 300
- WinPeekMsg, 300
- WinPostMsg, 301
- WinPostQueueMsg, 302
- WinPrevChar, 302
- WinProcessDlg, 303
- WinPtInRect, 303
- WinQueryAccelTable, 304
- WinQueryActiveWindow, 304
- WinQueryAtomLength, 305
- WinQueryAtomName, 305
- WinQueryAtomUsage, 306
- WinQueryCapture, 306
- WinQueryClassInfo, 307
- WinQueryClassName, 307
- WinQueryClipbrdData, 308
- WinQueryClipbrdFmtInfo, 308
- WinQueryClipbrdOwner, 310
- WinQueryClipbrdViewer, 310
- WinQueryCp, 311
- WinQueryCpList, 311
- WinQueryCursorInfo, 311
- WinQueryDefinition, 312
- WinQueryDesktopWindow, 313
- WinQueryDlgItemShort, 313
- WinQueryDlgItemText, 314
- WinQueryDlgItemTextLength, 314
- WinQueryFocus, 314
- WinQueryMsgPos, 315
- WinQueryMsgTime, 315
- WinQueryObjectWindow, 316
- WinQueryPointer, 316
- WinQueryPointerInfo, 316
- WinQueryPointerPos, 317
- WinQueryProfileData, 318
- WinQueryProfileInt, 318
- WinQueryProfileSize, 319
- WinQueryProfileString, 320
- WinQueryProgramTitles, 320
- WinQueryQueueInfo, 321
- WinQueryQueueStatus, 322
- WinQuerySysColor, 323
- WinQuerySysModalWindow, 323
- WinQuerySysPointer, 324
- WinQuerySystemAtomTable, 325
- WinQuerySysValue, 325
- WinQueryTaskTitle, 328
- WinQueryUpdateRect, 329
- WinQueryUpdateRegion, 330
- WinQueryVersion, 330
- WinQueryWindow, 331
- WinQueryWindowDC, 332
- WinQueryWindowLockCount, 332
- WinQueryWindowPos, 332

- WinQueryWindowProcess, 333
- WinQueryWindowPtr, 333
- WinQueryWindowRect, 334
- WinQueryWindowText, 334
- WinQueryWindowTextLength, 335
- WinQueryWindowULong, 335
- WinQueryWindowUShort, 336
- WinReallocMem, 337
- WinRegisterClass, 338
- WinRegisterWindowDestroy, 340
- WinReleaseHook, 340
- WinReleasePS, 341
- WinRemoveSwitchEntry, 342
- WinScrollWindow, 342
- WinSendDlgItemMsg, 344
- WinSendMsg, 345
- WinSetAccelTable, 346
- WinSetActiveWindow, 346
- WinSetCapture, 347
- WinSetClipbrdData, 347
- WinSetClipbrdOwner, 349
- WinSetClipbrdViewer, 350
- WinSetCp, 350
- WinSetDlgItemShort, 351
- WinSetDlgItemText, 351
- WinSetFocus, 352
- WinSetHook, 353
- WinSetKeyboardStateTable, 354
- WinSetMultWindowPos, 355
- WinSetOwner, 356
- WinSetParent, 357
- WinSetPointer, 357
- WinSetPointerPos, 358
- WinSetRect, 359
- WinSetRectEmpty, 359
- WinSetSysColors, 360
- WinSetSysModalWindow, 361
- WinSetSysValue, 362
- WinSetWindowBits, 365
- WinSetWindowPos, 366
- WinSetWindowPtr, 369
- WinSetWindowText, 369
- WinSetWindowULong, 370
- WinSetWindowUShort, 371
- WinShowCursor, 371
- WinShowPointer, 372
- WinShowTrackRect, 372
- WinShowWindow, 373
- WinStartTimer, 374
- WinStopTimer, 374
- WinSubclassWindow, 375
- WinSubstituteStrings, 375
- WinSubtractRect, 376
- WinTerminate, 377
- WinThrow, 377
- WinTrackRect, 378
- WinTranslateAccel, 380
- WinUnionRect, 381
- WinUpdateWindow, 382
- WinUpper, 382
- WinUpperChar, 383
- WinValidateRect, 383
- WinValidateRegion, 384
- WinWaitMsg, 384
- WinWindowFromDC, 385
- WinWindowFromID, 386
- WinWindowFromPoint, 387
- WinWriteProfileData, 387
- WinWriteProfileString, 388
- WM_ACTIVATE, 417
- WM_ADJUSTWINDOWPOS, 418
- WM_BUTTON1DBLCLK, 418
- WM_BUTTON1DOWN, 419
- WM_BUTTON1UP, 420
- WM_BUTTON2DBLCLK, 419
- WM_BUTTON2DOWN, 420
- WM_BUTTON2UP, 420
- WM_BUTTON3DBLCLK, 419
- WM_BUTTON3DOWN, 421
- WM_BUTTON3UP, 421
- WM_CALCVALIDRECTS, 421
- WM_CHAR, 422
- WM_CLOSE, 424
- WM_COMMAND, 424
- WM_CONTROL, 425
- WM_CONTROLHEAP, 425
- WM_CONTROLPOINTER, 426
- WM_CREATE, 426
- WM_DDE_ACK, 427
- WM_DDE_ADVISE, 428
- WM_DDE_DATA, 428
- WM_DDE_EXECUTE, 429
- WM_DDE_INITIATE, 429
- WM_DDE_INITIATEACK, 430
- WM_DDE_POKE, 430
- WM_DDE_REQUEST, 431
- WM_DDE_TERMINATE, 431
- WM_DDE_UNADVISE, 431
- WM_DESTROY, 432
- WM_DESTROYCLIPBOARD, 432
- WM_DRAWCLIPBOARD, 433
- WM_DRAWITEM, 433
- WM_ENABLE, 434
- WM_ERASEBACKGROUND, 434
- WM_FLASHWINDOW, 435
- WM_FOCUSCHANGE, 436
- WM_FORMATFRAME, 436
- WM_HELP, 437
- WM_HITTEST, 438
- WM_HSCROLL, 439
- WM_HSCROLLCLIPBOARD, 439
- WM_INITDLG, 440
- WM_INITMENU, 441
- WM_JOURNALNOTIFY, 441
- WM_MATCHMnemonic, 442
- WM_MEASUREITEM, 442
- WM_MENUEND, 443
- WM_MENUSELECT, 443
- WM_MINMAXFRAME, 444
- WM_MOUSEMOVE, 444
- WM_MOVE, 445
- WM_NEXTMENU, 445
- WM_NULL, 446
- WM_OTHERWINDOWDESTROYED, 446
- WM_PAINT, 446
- WM_PAINTCLIPBOARD, 447
- WM_QUERYACCELTABLE, 447
- WM_QUERYBORDERSIZE, 447
- WM_QUERYCONVERTPOS, 448
- WM_QUERYDLGCODE, 448
- WM_QUERYFOCUSCHAIN, 449
- WM_QUERYFRAMECTLCOUNT, 450
- WM_QUERYFRAMEINFO, 450
- WM_QUERYICON, 451
- WM_QUERYTRACKINFO, 451
- WM_QUERYWINDOWPARAMS, 452
- WM_QUIT, 453
- WM_RENDERALLFMts, 453
- WM_RENDERFMT, 454
- WM_SEM1, 454
- WM_SEM2, 455
- WM_SEM3, 456
- WM_SEM4, 456
- WM_SETACCELTABLE, 456
- WM_SETBORDERSIZE, 456
- WM_SETFOCUS, 457
- WM_SETICON, 457
- WM_SETSELECTION, 457
- WM_SETWINDOWPARAMS, 458
- WM_SHOW, 458
- WM_SIZE, 459
- WM_SIZECLIPBOARD, 459
- WM_SUBSTITUTESTRING, 460
- WM_SYSCOLORCHANGE, 460
- WM_SYSCOMMAND, 461
- WM_SYSVALUECHANGED, 461
- WM_TIMER, 462
- WM_TRACKFRAME, 462
- WM_TRANSLATEACCEL, 463
- WM_UPDATEFRAME, 464
- WM_VIOCHAR, 464
- WM_VSCROLL, 466

564 W

WM_VSCROLLCLIPBOARD, 467
WNDPARAMS, 524

X
XYWINSIZE, 524

Step up to Presentation Manager with the Microsoft OS/2 Presentation Manager Softset.

Congratulations on your purchase of the Microsoft® OS/2 Programmer's Reference Library, a complete guide to the features of the Microsoft OS/2 Presentation Manager. Now that you have the documentation, the next step is to purchase Microsoft OS/2 Presentation Manager Softset version 1.1, which Microsoft designed to help software developers create the new generation of graphically based, intuitive, easy-to-use software applications. Softset provides a complete, fully documented set of visual software tools to help you create popular applications for the graphical environment of Presentation Manager.

Softset Features

- Dialog Editor helps you design on-screen dialog boxes.
- Icon Editor helps you customize icons, cursors, and bitmap images for graphical applications.
- Font Editor helps you create your own fonts.
- Resource Compiler helps you bind resource-definition files created with the Dialog, Icon, and Font Editors to .EXE files.
- Other Softset tools help you create and maintain libraries, create message files and dual-mode (DOS-OS/2) programs, and perform many other tasks.

Combine the Softset with the Microsoft OS/2 Programmer's Reference Library and a programming language such as Microsoft C Optimizing Compiler or Microsoft Macro Assembler with OS/2 support for a complete Presentation Manager software development kit. The applications you create in Presentation Manager are fully compatible with IBM® SAA (Systems Application Architecture). Trust the software tools from Microsoft—the company that developed MS® OS/2.

Contact your nearest local software dealer for more information.

Also Available From Microsoft Press

Authoritative Information for OS/2 Programmers

INSIDE OS/2

Gordon Letwin

"The best way to understand the overall philosophy of OS/2 will be to read this book."

— *Bill Gates*

Here — from Microsoft's Chief Architect of Systems Software — is an exciting technical examination of the philosophy, key development issues, programming implications, and role of OS/2 in the office of the future. And Letwin provides the first in-depth look at each of OS/2's design elements. This is a valuable and revealing programmer-to-programmer discussion of the graphical user interface, multitasking, memory management, protection, encapsulation, interprocess communication, and direct device access. You can't get a more inside view.

304 pages, 7³/₈ x 9¹/₄, softcover, \$19.95.

[Order Code 86-96288]

ADVANCED OS/2 PROGRAMMING

Ray Duncan

Authoritative information, expert advice, and great assembly-language code make this comprehensive overview of the features and structure of OS/2 indispensable to any serious OS/2 programmer. Duncan addresses a range of significant OS/2 issues: programming the user interface; mass storage; memory management; multitasking; interprocess communications; customizing filters, device drivers, and monitors; and using OS/2 dynamic link libraries. A valuable reference section includes detailed information on each of the more than 250 system service calls in version 1.1 of the OS/2 kernel.

800 pages, 7³/₈ x 9¹/₄, softcover, \$24.95

[Book Code 86-96106]

PROGRAMMING THE OS/2 PRESENTATION MANAGER

Charles Petzold

New! Here is the first full discussion of the features and operation of the OS/2 1.1 Presentation Manager. If you're developing OS/2 applications, this book will guide you through Presentation Manager's system of windows, messages, and function calls. Petzold includes scores of valuable C programs and utilities. Endorsed by the Microsoft Systems Software group, this book is unparalleled for its clarity, detail, and comprehensiveness. Petzold covers: managing windows ■ handling input and output ■ controlling child windows ■ using bitmaps, icons,

pointers, and strings ■ accessing the menu and keyboard accelerators ■ working with dialog boxes ■ understanding dynamic linking ■ and more.

864 pages, 7³/₈ x 9¹/₄, softcover, \$29.95
[Order Code 86-96791]

ESSENTIAL OS/2 FUNCTIONS: Programmer's Quick Reference

Ray Duncan

Concise information on the essential OS/2 function calls within the application program interface (API). Entries are included for all kernel API functions for OS/2 version 1.0: Dos, Kbd, Mou, and Vio. Brief descriptions of each function are included, as well as a list of the required parameters, returned results, programming notes and warnings, family API call identification, and error codes. Conveniently arranged to provide quick access to the information you need.

172 pages, 4³/₄ x 8, softcover, \$9.95
[Order Code 86-96866]

For the Windows Programmer

PROGRAMMING WINDOWS

Charles Petzold

Your fastest route to successful application programming with Windows. Full of indispensable reference data, tested programming advice, and page after page of creative sample programs and utilities. Topics include getting the most out of the keyboard, mouse, and timer; working with icons, cursors, bitmaps, and strings; exploiting Windows' memory management; creating menus; taking advantage of child window controls; incorporating keyboard accelerators; using dynamically linkable libraries; and mastering the Graphics Device Interface (GDI). A thorough, up-to-date, and authoritative look at Windows' rich graphical environment.

864 pages, 7³/₈ x 9¹/₄
\$24.95 (sc) [Order Code 86-96049]
\$34.95 (hc) [Order Code 86-96130]

Unbeatable Programmer's References

PROGRAMMER'S GUIDE TO PC & PS/2[®] VIDEO SYSTEMS

Richard Wilton

No matter what your hardware configuration, here is all the information you need to create fast, professional, even stunning video graphics on IBM PCs, compatibles, and PS/2s. No other book offers such detailed, specialized programming data, techniques, and advice to help you tackle the exacting

challenges of programming directly to the video hardware. And no other book offers the scores of invaluable source code examples included here. Whatever graphic output you want — text, circles, region fill, alphanumeric character sets, bit blocks, animation — you'll do it cleaner, faster, and more effectively with Wilton's book.

544 pages, 7⁷/₈ x 9¹/₄, softcover, \$24.95
[Order Code 86-96163]

THE 80386 BOOK

Ross P. Nelson

A clear, comprehensive, and authoritative introduction for every serious programmer. Included are scores of superb assembly-language examples along with a detailed analysis of the 80386 chip. Topics covered include: the CPU, the memory architecture, the instructions sets of the 80386 microprocessor and the 80387 math coprocessor, the protection scheme, the implementation of a virtual memory system through paging, and compatibility with earlier Intel microprocessors. Of special note is the comprehensive, clearly organized instruction set reference — guaranteed to be a valuable resource.

464 pages, 7⁷/₈ x 9¹/₄, softcover, \$24.95
[Order Code 86-96494]

THE PROGRAMMER'S PC SOURCEBOOK

Thom Hogan

At last! A reference to save you the time required to find key pieces of technical data. Here is important factual information — previously published in scores of other sources — organized into one convenient reference. Focusing on IBM PCs and compatibles, PS/2s and MS-DOS, the hundreds of charts and tables cover:

■ numeric conversions and character sets ■ DOS commands and utilities ■ DOS function calls and support tables ■ DOS BIOS calls and support tables ■ other interrupts, mouse, and EMS support ■ Microsoft Windows ■ keyboards, video adapters, and peripherals ■ chips, jumpers, switches, and registers ■ hardware descriptions ■ and more.

560 pages, 8¹/₂ x 11, softcover, \$24.95
[Order Code 86-96296]

The Microsoft Press CD-ROM Library

THE MICROSOFT® CD-ROM YEARBOOK: 1989/1990

Microsoft Press

Foreword by Bill Gates

A dynamic, fact-filled portrait and analysis of the wide-ranging, fast-paced CD-ROM industry. Indispensable for anyone involved in the industry as well as an information-packed compendium for those curious about CD-ROM. Readers can use the book as a valuable sourcebook of facts, statistics, and forecasts, or dip into it for fascinating articles, reviews, and analyses of the industry. Articles include:

- an absorbing history—in text and pictures—of the CD-ROM industry
- reviews of products—hardware and software—considered outstanding or standard-setting
- profiles of the leading companies and people in the industry
- an overview of the process of developing a CD-ROM product
- a review of the legal issues of protection, rights and permissions, contracts and royalties surrounding CD-ROM publishing
- the strategies and pitfalls involved in getting a CD-ROM product to market

The breadth of accurate, up-to-date information in THE MICROSOFT CD-ROM YEARBOOK is impressive including:

- comprehensive reference listings of the people, equipment, available titles, sources, and resources in the CD ROM industry
- a glossary of industry terms
- a calendar of industry events and conferences
- specialized bibliographies

This is *the* reference of fact and opinion on the industry.

960 pages, 7³/₈ x 9¹/₄, softcover, \$79.95

[Order Code 86-97203]

CD ROM: THE NEW PAPYRUS

Edited by Steve Lambert and Suzanne Ropiequet

"This 619-page compendium, with contributions from more than 30 optical-memory specialists, promises to become the bible of CD ROM." David Bunnell, Macworld

This special compendium of 45 articles by leading authorities examines every facet of compact disc read only memory technology: hardware, software, applications, publishing systems, marketing, and the user interface. Includes introductory as well as technical information.

608 pages, 7³/₈ x 9¹/₄, softcover, \$21.95

[Order Code 86-95454]

CD ROM 2: OPTICAL PUBLISHING

Edited by Suzanne Ropiequet with John Einberger and Bill Zoellick

"Recommended reading for any information professional." Online Today

This is a comprehensive overview of the entire optical publishing process. Topics include: evaluating and defining storage and retrieval methods; collecting, preparing, and indexing data; updating strategies; data protection and copyrighting; and more. Plus information on the High Sierra Logical Format. In addition, the editors trace the development of two CD ROM projects from initial concept to final product. For publishers, technical managers, and entrepreneurs.

384 pages, 7³/₈ x 9¹/₄, softcover, \$22.95

[Order Code 86-95686]

INTERACTIVE MULTIMEDIA

Foreword by John Sculley

Edited by Sueanne Ambron and Kristina Hooper

Apple Computer Corp. brought together leading researchers and developers to produce this informative collection of 21 articles. The result is a sourcebook of ideas and inspiration for software and hardware developers, educators, publishers, and information providers. The contributors, including Doug Englebart, Sam Gibbon, and Peter Cook, represent the industries — computers, television, and publishing — whose products will provide the content and media for education in the future. Filled with examples and pilot projects that define the new meaning of multimedia. Published with Apple Computer, Inc.

352 pages, 7³/₈ x 9¹/₄, softcover, \$24.95

[Order Code 86-96379]

Microsoft Press books are available wherever books and software are sold.

Or you can place a credit card order by calling 1-800-638-3030 (8 AM to 4:30 PM EST).

In Maryland, call collect: 824-7300.

M I C R O S O F T[®]
OS/2

Programmer's Reference

Including Presentation Manager

The Microsoft[®] Operating System/2 Programmer's Reference Library should be the cornerstone of every OS/2 developer's programming library. These volumes are required references for professional developers creating applications for the retail market; for corporate programmers creating in-house software programs; for hardware manufacturers creating software to support their products; and for all other experienced programmers working in the OS/2 environment.

Each volume in the series is written by a team of OS/2 specialists—many involved in the development and ongoing enhancement of OS/2 at Microsoft. These books provide in-depth, accurate, and up-to-date information from the Microsoft OS/2 Presentation Manager Toolkit—the software development kit essential for creating OS/2 applications.

Volume 1

Volume 1 details the conceptual framework of the MS[®] OS/2 Application Programming Interface (API). Included are thorough descriptions of MS[®] OS/2 programming models, overviews of basic programming considerations, and explanations of the interaction between the API and the rest of the MS[®] OS/2 system. Sections include *Introducing MS[®] OS/2*, *Window Manager*, *Graphics Programming Interface*, and *System Services*.

Volume 2

Volume 2 is a comprehensive, alphabetic listing of MS[®] OS/2 Presentation Manager functions as well as the structures and file formats used with these functions. Each function entry includes information on syntax; descriptions of the function's actions and purpose; parameters and field definitions; return values, error values, and restrictions; source-code examples; and programming notes. Appendix included.

Volume 3

Similar in format to Volume 2, Volume 3 is a comprehensive alphabetic listing of MS[®] OS/2 base functions, including their structures and file formats. Appendixes included.

U.S.A. \$29.95
U.K. £24.95
Austral. \$44.95
(recommended)

ISBN 1-55615-221-3

