Presentation Manager Programming Reference Vol I

IBM

OS/2® WARP

Version 3

Presentation Manager Programming Reference Vol I

IBM

OS/2® WARP

Version 3

Note

Before using this information and the product it supports, be sure to read the general information under Appendix A, "Notices" on page A-1.

First Edition (October 1994)

Contents

About This Book

This book is the technical reference, in two volumes, for application programmers creating programs using the Presentation Manager* (PM*) functions in the application programming interface (API) of the OS/2* operating system. This reference is intended to be used in conjunction with the *Presentation Manager Programming Guide—The Basics* and the *Presentation Manager Programming Guide—Advanced Topics*.

Who Should Read This Book

The *Presentation Manager Programming Reference* is intended for application programmers who want to develop programs using the Presentation Manager interface. This reference provides technical information about functions, messages and other related information available to the developer.

How This Book is Organized

This book is divided into two volumes. The contents of each volume are described in the lists below:

Volume One (Functions)

- Chapter 1, Introduction to the Presentation Manager

  This chapter introduces the OS/2 Presentation Manager programming notation and conventions used in this book. This chapter contains important information and should be read before using this book.
- Chapter 2, Device Functions
- Chapter 3, Direct Manipulation Functions
- Chapter 4, Dynamic Data Formatting Functions
- Chapter 5, Profile Functions
- Chapter 6, Spooler Functions
- Chapter 7, Window Functions
- Chapter 8, Functions Supplied by Applications

Volume Two (Messages and Related Information)

- Chapter 9, Introduction to Message Processing
- Chapter 10, Default Window Procedure Message Processing
- Chapter 11, Button Control Window Processing
- Chapter 12, Entry Field Control Window Processing
- Chapter 13, Frame Control Window Processing
- Chapter 14, List Box Control Window Processing

- Chapter 15, Menu Control Window Processing
- Chapter 16, Multi-Line Entry Field Control Processing
- Chapter 17, Combination-Box Control Window Processing
- Chapter 18, Scroll Bar Control Window Processing
- Chapter 19, Spin Button Control Window Processing
- Chapter 20, Static Control Window Processing
- Chapter 21, Title Bar Control Window Processing
- Chapter 22, Container Control Window Processing
- Chapter 23, Notebook Control Window Processing
- Chapter 24, Slider Control Window Processing
- Chapter 25, Circular Slider Control Window Processing
- Chapter 26, Value Set Control Window Processing
- Chapter 27, Clipboard Messages
- Chapter 28, Direct Manipulation (Drag) Messages
- Chapter 29, Dynamic Data Exchange Messages
- Chapter 30, Help Manager Messages
- Chapter 31, Resource Files
- Chapter 32, Code Pages
- Appendix A, Data Types
- Appendix B, Error Codes
- Appendix C, Error Explanations
- Appendix D, Standard Bit-Map Formats
- Appendix E, Fonts Supplied with the OS/2 Operating System
- Appendix F, Format of Interchange Files
- Appendix G, Initialization File Information
- Appendix H, Virtual key Definitions
- Glossary
- Index

To illustrate the use of the various PM functions, this reference makes extensive use of code fragments. There are also sample applications available with the *IBM Developer's Toolkit for OS/2 Version 3* (Toolkit). You may find it useful to execute the samples and examine the C files, resource files, makefiles, and other files provided by the Toolkit.

For more information on how to compile and link your programs, refer to the compiler publication for the programming language you are using.

Prerequisite Knowledge

This reference is intended for application designers and programmers who are familiar with the following:

- Information contained in the *Control Program Programming Guide*
- Information contained in the *Presentation Manager Programming Guide—The Basics* and in the *Presentation Manager Programming Guide—Advanced Topics*
- C Programming Language

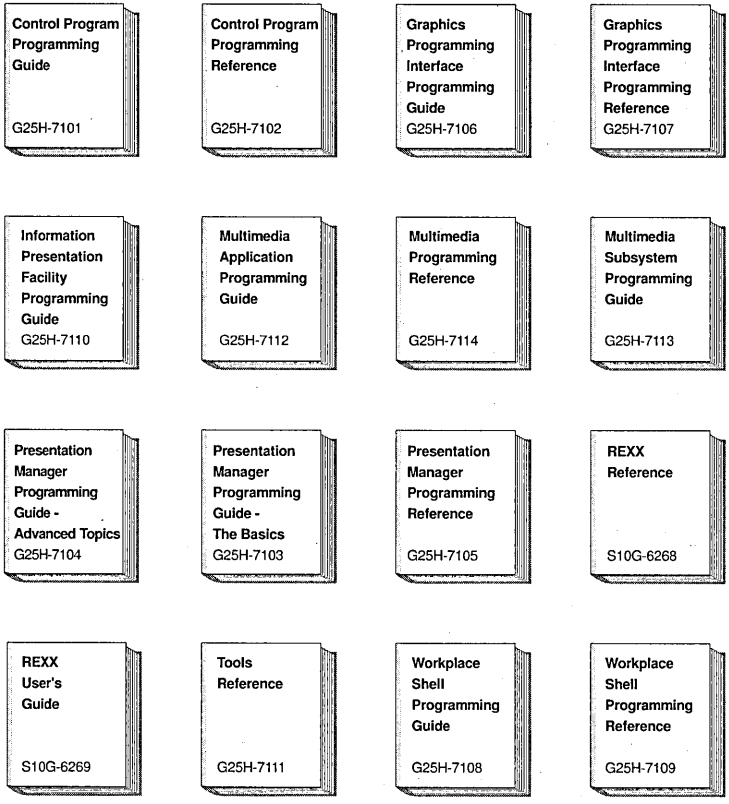
Programming experience on a multitasking operating system would also be helpful.

Related Publications

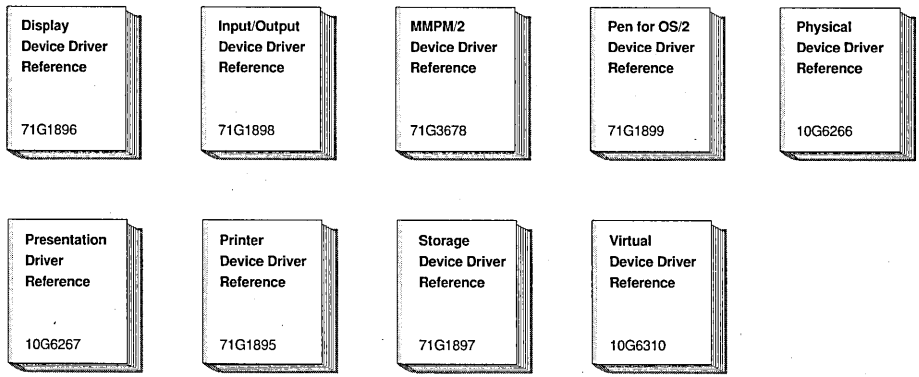
The following diagram provides an overview of the OS/2 Version 3 Technical Library.

Books can be ordered by calling toll free 1-800-342-6672 weekdays between 8:00 a.m. and 8:00 p.m. (EST). In Canada, call 1-800-465-4234.

OS/2 Warp, Version 3 Technical Library
G25H-7116



IBM Device Driver Publications for OS/2



Chapter 1. Introduction to the Presentation Manager

The purpose of this reference is to give important information about functions, messages, constants, and data types. It provides language-dependent information about the functions which enables the user to call functions in the C programming language.

The following information is provided:

- The syntax and parameters for each function
- The syntax of each data type and structure.

Presentation Manager Fundamentals

The Presentation Manager (PM) provides a message-based, event-driven, graphical user interface for the OS/2 environment. PM enables programmers to build applications that conform to Systems Application Architecture* (SAA*) guidelines.

The PM user interface is based on *windows*, an area of the screen through which applications interace with the user. A large number of functions (which begin with the prefix Win) are available for controlling windows. These functions enable an application to create, size, move, and control windows and their contents. Refer to *Presentation Manager Programming Guide—The Basics* for a description of common programming techniques for managing the window environment.

Notation Conventions

The following notation conventions are used in this reference:

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

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

Implicit Pointer If no entry for a data type "Pxxxxxxx" is found in Appendix A, "Data Types" in the *Presentation Manager Programming Reference Volume II*, then it is implicitly a pointer to the data type "xxxxxxx." See "Implicit Pointer Data Types" on page 1-6 for more information about implicit pointers.

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

WM_CREATE	Window message
SV_CXICON	System value
CF_TEXT	Clipboard format.

In this book, references to a complete set of constants with a given prefix is written as shown in the following examples:

Window message	WM_*
System value	SV_*

Parameters and Fields Function parameters and data structure fields are shown in italics.

Conventions Used in Function Descriptions

The documentation of each function contains these sections:

Syntax

The function syntax describes the C-language calling syntax of the function and gives a brief description.

Programming Note:

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

If you are using a compiler that generates a mixed-case external name, you should code the functions in uppercase.

Parameters

Each parameter is listed with its C-language data type, parameter type, and a brief description.

- All data types are written in uppercase to match the header files. A data type of "Pxxxxxxx" implicitly defines a pointer to the data type "xxxxxxx."

  The term NULL applied to a parameter indicates the presence of the parameter, with no value.

  Refer to Appendix A, "Data Types" in the *Presentation Manager Programming Reference Volume II* for a complete list of all data types and their descriptions.

- There are three parameter types:

Input	Specified by the programmer.
Output	Returned by the function.
Input/Output	Specified by the programmer and modified by the function.

- A brief description is provided with each parameter. Where appropriate, restrictions are also included. In some cases, the parameter points to a structure.

Returns

A list of possible return codes or errors (when appropriate) is included in this section. Some functions do not have return codes. Refer to "Error Codes" in the *Presentation Manager Programming Reference Volume II*. for a list of error codes and their numerical values, and "Error Explanations" the *Presentation Manager Programming Reference Volume II*. for a list of error codes and their descriptions.

For some functions, this section includes a statement that the function requires a message queue. This means that, before issuing a call, WinCreateMsgQueue must be

issued by the same thread. For other functions, no previous WinCreateMsgQueue is required, and it is only necessary to issue WinInitialize from the same thread.

Remarks
This section contains additional information about the function, when required.

Related Functions
This list shows the functions (if any) that are related to the function being described.

Example Code
An example of how the function can be used is shown in C language.

Error Severities

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

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

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

Header Files

All functions require an "#include" statement for the system header file OS2.H:

```
#include  <OS2.H>
```

Most functions also require a "#define" statement to select an appropriate (conditional) section of the header file, and hence, the required prototype. Where this is necessary, it is shown at the head of the function definition in the form:

```
#define   INCL_name
```

Note: These "#define" statements must precede the "#include <OS2.H>" statement.

Helper Macros

A series of macros is defined for packing data into, and extracting data from, variables of MPARAM and MRESULT data types. They are used in conjunction with the WinSendMsg and the other message functions, and also inside window and dialog procedures.

These macros always cast their arguments to the specified type, so values of any of the types specified for each macro can be passed without additional casting. NULL can be used to pass unused parameter data.

Macros for packing data into a MPARAM variable are shown below:

```
/* Used to pass any pointer type: */
#define MPFROMP(p)            ((MPARAM)(VOID *)(p))

/* Used to pass a window handle: */
#define MPFROMHWND(hwnd)      ((MPARAM)(HWND)(hwnd))

/* Used to pass a CHAR, UCHAR, or BYTE: */
#define MPFROMCHAR(ch)        ((MPARAM)(USHORT)(ch))

/* Used to pass a SHORT, USHORT, or BOOL: */
#define MPFROMSHORT(s)        ((MPARAM)(USHORT)(s))

/* Used to pass two SHORTs or USHORTs: */
#define MPFROM2SHORT(s1, s2)  ((MPARAM)MAKELONG(s1, s2))

/* Used to pass a SHORT and 2 UCHARs: (WM_CHAR msg)*/
#define MPFROMSH2CH(s, uch1, uch2)
 ((MPARAM)MAKELONG(s, MAKESHORT(uch1, uch2)))

/* Used to pass a LONG or ULONG: */
#define MPFROMLONG(l)         ((MPARAM)(ULONG)(l))
```

Macros for extracting data from a MPARAM variable are shown below:

```
/* Used to get any pointer type: */
#define PVOIDFROMMP(mp)      ((VOID *)(mp))

/* Used to get a window handle: */
#define HWNDFROMMP(mp)       ((HWND)(mp))

/* Used to get CHAR, UCHAR, or BYTE: */
#define CHAR1FROMMP(mp)      ((UCHAR)(mp))
#define CHAR2FROMMP(mp)      ((UCHAR)((ULONG)mp >> 8))
#define CHAR3FROMMP(mp)      ((UCHAR)((ULONG)mp >> 16))
#define CHAR4FROMMP(mp)      ((UCHAR)((ULONG)mp >> 24))

/* Used to get a SHORT, USHORT, or BOOL: */
#define SHORT1FROMMP(mp)     ((USHORT)(ULONG)(mp))
#define SHORT2FROMMP(mp)     ((USHORT)((ULONG)mp >> 16))

/* Used to get a LONG or ULONG: */
#define LONGFROMMP(mp)       ((ULONG)(mp))
```

Macros for packing data into a MRESULT variable are shown below:

```
/* Used to pass any pointer type: */
#define MRFROMP(p)             ((MRESULT)(VOID *)(p))

/* Used to pass a SHORT, USHORT, or BOOL: */
#define MRFROMSHORT(s)         ((MRESULT)(USHORT)(s))

/* Used to pass two SHORTs or USHORTs: */
#define MRFROM2SHORT(s1, s2)  ((MRESULT)MAKELONG(s1, s2))

/* Used to pass a LONG or ULONG: */
#define MRFROMLONG(l)          ((MRESULT)(ULONG)(l))
```

Macros for extracting data from a MRESULT variable are shown below:

```
/* Used to get any pointer type: */
#define PVOIDFROMMR(mr)      ((VOID *)(mr))

/* Used to get a SHORT, USHORT, or BOOL: */
#define SHORT1FROMMR(mr)     ((USHORT)((ULONG)mr))
#define SHORT2FROMMR(mr)     ((USHORT)((ULONG)mr >> 16))

/* Used to get a LONG or ULONG: */
#define LONGFROMMR(mr)       ((ULONG)(mr))
```

The following macros are for use with DDESTRUCT and DDEINIT structures are shown below;

```
/* Used to return a PSZ pointing to the DDE item name: */
#define DDES_PSZITEMNAME(pddes) \
        (((PSZ)pddes) + ((PDDESTRUCT)pddes)->offszItemName)

/* Used to return a PBYTE pointing to the DDE data: */
#define DDES_PABDATA(pddes)       \
        (((PBYTE)pddes) + ((PDDESTRUCT)pddes)->offabData)

/* Used to convert a selector to a PDDESTRUCT: */
#define SELTOPDDES(sel)             ((PDDESTRUCT)MAKEP(sel, 0))

/* Used to PDDESTRUCT to a selector for freeing / reallocating: */
#define PDDESTOSEL(pddes)           (SELECTOROF(pddes))

/* Used to PDDEINIT to a selector for freeing: */
#define PDDEITOSEL(pddei)           (SELECTOROF(pddei))
```

Addressing Elements in Arrays

Constants defining array elements are given values that are zero-based in C; that is, the numbering of the array elements starts at zero, not one.

For example, in the DevQueryCaps function, the sixth element of the *alArray* parameter is CAPS_HEIGHT, which is equated to 5.

Count parameters related to such arrays always mean the actual number of elements available; therefore, again using the DevQueryCaps function as an example, if all elements up to and including CAPS_HEIGHT are provided for, *lCount* could be set to (CAPS_HEIGHT+1).

In functions for which the starting array element can be specified, this is always zero-based, and so the C element number constants can be used directly. For example, to start with the CAPS_HEIGHT element, the *lStart* parameter can be set to CAPS_HEIGHT.

Implicit Pointer Data Types

A data type name beginning with "P" (for example, PERRORCODE) is likely to be a pointer to another data type (in this instance, ERRORCODE).

In the data type summary, Appendix A, "Data Types" in the *Presentation Manager Programming Reference Volume II*, no explicit "typedefs" are shown for pointers; therefore, if no data type definition can be found in the summary for a data type name "Pxxxxxx," it represents a pointer to the data type "xxxxxx," for which a definition should be found in the reference.

The implicit type definition needed for such a pointer "Pxxxxxx" is:

```
typedef xxxxxx *Pxxxxxx;
```

Such definitions are provided in the header files.

Storage Mapping of Data Types

The storage mapping of the data types is dependent on the machine architecture. To be portable, applications must access the data types using the definitions supplied for the environment in which they will execute.

Double-Byte Character Set (DBCS)

Throughout this publication, you will see references to specific value for character strings. The values are for single-byte character set (SBCS). If you use the double-byte character set (DBCS), note that one DBCS character equals two SBCS characters.

Programming Considerations

This section provides information you need to consider before you begin programming with Presentation Manager functions.

Stack Size

Existing 16-bit applications (small and tiny models) must have a 4KB stack available when they enter system calls; otherwise, the stack can overflow into the data area.

Presentation Manager

The Presentation Manager component of the OS/2* operating system is based on the IBM Systems Application Architecture* (SAA*) Common Programming Interface–a an architecture for the design and development of applications.

The Presentation Manager component implements the Common User Access* (CUA*) interface, which you can use to attain consistency in the appearance and behavior of your applications.

C++ Considerations

This section contains several topics you should take into consideration if you are using C++ **.

C++ Header Files

OS/2 functions that used to take a PSZ as a parameter, and that do not modify the contents of the passed string, have been updated in the C++ header files to take a PCSZ data type parameter. The use of PCSZ allows for better optimization by the compiler and is more

semantically compatible with C++. Existing code that calls functions that use PSZ will continue to work correctly.

Several of the typedefs have been changed in the C++ header files. For example, many items that are `unsigned char` in the C header files are `char` in the C++ header files. For instance,

```
typedef unsigned char BYTE;
```

has changed to

```
typedef char BYTE;
```

The existing samples that are included in the IBM Developer's Toolkit for OS/2 Version 3 can be used with either set of the header files.

PCSZ Data Type

Note: The PCSZ data type is defined in the C++ header files included with this product. The use of the "const" keyword is not necessarily specific to C++. Certain C compilers support it as well.

If a function takes as a parameter a string that is not changed by the function, the string parameter can be declared as a "const" string, or a PCSZ. PCSZ is defined in the C++ header files as a "const" pointer to a NULL-delimited string. The "const" means that the function will not change the contents of the string.

Declaring the parameter as PCSZ informs the C++ compiler that the function will not change the string. Therefore, the compiler simply passes a pointer to the string in the function parameter list. If the parameter is declared as a normal PSZ (not "const"), the compiler assumes that the function might change the string. Under these circumstances the compiler will add code to make a copy of the string then pass a pointer to the copy, rather than pass a pointer to the original string.

A smaller, faster executable is often produced if the data item passed in a parameter list is declared as "const."

If the data item is declared as "const" then it must not be changed by the function.

LINK386

The C++ compiler will provide a dynamic link library which is be used by LINK386 when generating error messages. This DLL will convert a compiler generated mangled name into the function prototype. If the DLL is not present, an error message will be displayed and LINK386 will display the compiler-generated mangled name in error messages.

Chapter 2. Device Functions

The following table shows all the Device (Dev) functions in alphabetic order.

C Name
DevCloseDC
DevEscape
DevOpenDC
DevPostDeviceModes
DevQueryCaps
DevQueryDeviceNames
DevQueryHardcopyCaps

DevCloseDC

This function closes a device context.

Syntax

```
#define INCL_DEV /* Or use INCL_PM, Also in COMON section */
#include <os2.h>
HMF DevCloseDC  (HDC hdc)
```

Parameters

hdc (HDC) – input
Device-context handle.

Returns

hmf (HMF) – returns
Error indicator metafile handle (for a metafile device context)

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

Possible returns from WinGetLastError

PMERR_NOT_CREATED_BY_DEVOPENDC (0x20DC)	An attempt has been made to destroy a device context using DevCloseDC that was not created using DevOpenDC.
PMERR_DC_IS_ASSOCIATED (0x2017)	An attempt was made to associate a presentation space with a device context that was already associated or to destroy a device context that was associated.
PMERR_INV_HDC (0x207C)	An invalid device-context handle or (micro presentation space) presentation-space handle was specified.

Remarks

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

If the device context being closed is a memory device context that has a bit map currently selected into it (see "GpiSetBitmap" in the *Presentation Manager Programming Reference*) the bit map is automatically deselected before the device context is closed.

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

Related Functions

Prerequisite Functions

- DevOpenDC

Related Functions

- WinOpenWindowDC

Example Code

This example calls DevCloseDC to close a device context based on the handle returned from DevOpenDC.

```
#define INCL_DEV                /* Device Function definitions  */
#include <os2.h>

HDC hdc;                    /* Device-context handle                */
HMF hmf;                    /* error code (or metafile handle if
                               metafile device context)             */

/* close the device context associated with handle hdc */
hmf = DevCloseDC(hdc);
```

DevEscape

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

Syntax

```
#define INCL_DEV /* Or use INCL_PM, */
#include <os2.h>
LONG DevEscape (HDC hdc, LONG lCode, LONG lInCount, PBYTE pbInData,
                PLONG plOutCount, PBYTE pbOutData)
```

Parameters

hdc (HDC) – input
Device-context handle.

lCode (LONG) – input
Escape code.

If the device context is of type OD_QUEUED with a PM_Q_STD spool file, some escapes are sent to the presentation driver and others are recorded in the spool file (depending on the escape code). If the device context is of type OD_METAFILE, all escapes are metafiled. If the device context is of any type other than OD_QUEUED (with a PM_Q_STD spool file) or OD_METAFILE, all escapes are sent to the presentation driver.

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

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

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

The following escapes are defined:

DEVESC_QUERYESCSUPPORT
DEVESC_GETSCALINGFACTOR
DEVESC_STARTDOC
DEVESC_ENDDOC
DEVESC_ABORTDOC
DEVESC_NEWFRAME
DEVESC_RAWDATA
DEVESC_QUERYVIOCELLSIZES
DEVESC_SETMODE

lInCount (LONG) – input
Input data count.

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

pbInData (PBYTE) – input
The input data required for this escape.

plOutCount (PLONG) – in/out
Output data count.

plOutCount is the number of bytes of data in the *pbOutData* buffer.

If data is returned in *pbOutData*, *plOutCount* is updated to the number of bytes of data returned.

pbOutData (PBYTE) – output
Output data.

pbOutData is a buffer that receives the output from this escape. If *plOutCount* is null, no data is returned.

Returns

lResult (LONG) – returns
Implementation error indicator:

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

Possible returns from WinGetLastError

PMERR_INV_ESC_CODE (0x206D)	An invalid escape code was used in a call to DevEscape.
PMERR_INV_HDC (0x207C)	An invalid device-context handle or (micro presentation space) presentation-space handle was specified.
PMERR_INV_LENGTH_OR_COUNT (0x2092)	An invalid length or count parameter was specified.

PMERR_ESC_CODE_NOT_SUPPORTED (0x202B)	The code specified with DevEscape is not supported by the target device driver.
PMERR_INV_ESCAPE_DATA (0x206E)	An invalid data parameter was specified with DevEscape.

Remarks

The data fields for standard escapes are:

DEVESC_QUERYESCSUPPORT

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

This escape is not metafiled or recorded.

llnCount	Number of bytes pointed to by *pbInData*.
pbInData	The buffer contains an escape code value specifying the escape function to be checked.
plOutCount	Not used; can be set to 0.
pbOutData	Not used; can be set to null.

DEVESC_GETSCALINGFACTOR

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

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

This escape is not metafiled or recorded.

llnCount	Not used; can be set to 0.
pbInData	Not used; can be set to null.
plOutCount	The number of bytes of data pointed to by *pbOutData*. On return, this is updated to the number of bytes returned.
pbOutData	The address of a SFACTORS structure, which on return contains the scaling factors for the x and y axes.

DEVESC_STARTDOC

Indicates that a new print job is starting. All subsequent output to the device context is spooled under the same job identifier until a DEVESC_ENDDOC occurs.

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

This escape is metafiled but not recorded.

llnCount	Number of bytes pointed to by *pbInData*.
pbInData	The buffer contains a null-terminated string, specifying the name of the document.
plOutCount	Not used; can be set to 0.
pbOutData	Not used; can be set to null.

DEVESC_ENDDOC

Ends a print job started by DEVESC_STARTDOC.

This escape is metafiled but not recorded.

llnCount	Not used; can be set to 0.
pbInData	Not used; can be set to null.
plOutCount	Set equal to 2.
pbOutData	The buffer contains a USHORT specifying the job identifier if a spooler print job is created.

DEVESC_ABORTDOC

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

This escape is metafiled but not recorded.

llnCount	Not used; can be set to 0
pbInData	Not used; can be set to null
plOutCount	Not used; can be set to 0
pbOutData	Not used; can be set to null.

DEVESC_NEWFRAME

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

This escape is metafiled and recorded.

llnCount	Not used; can be set to 0
pbInData	Not used; can be set to null
plOutCount	Not used; can be set to 0
pbOutData	Not used; can be set to null.

DEVESC_RAWDATA

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

If DEVESC_RAWDATA is mixed with other data (such as 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 driver. For example, a presentation driver might ignore GPI data if DEVESC_RAWDATA is mixed with it on the same page. In general, DEVESC_RAWDATA should be sent either to a separate page (using the DEVESC_NEWFRAME escape to obtain a new page) or to a separate document (using the DEVESC_STARTDOC and DEVESC_ENDDOC escapes to create a new document).

This escape is metafiled and recorded.

llnCount	Number of bytes pointed to by *pbInData*
pbInData	Pointer to the raw data
plOutCount	Not used; can be set to 0
pbOutData	Not used; can be set to null.

DEVESC_QUERYVIOCELLSIZES

Returns the VIO cell sizes supported by the presentation driver.

This escape is not metafiled or recorded.

llnCount Not used; can be set to 0

pbInData Not used; can be set to null.

plOutCount The number of bytes of data pointed to by *pbOutData*. It must be an even multiple of the size in bytes of the LONG data type. On return, this is updated to the number of bytes returned.

pbOutData The address of a buffer, which on return contains a VIOSIZECOUNT structure, immediately followed by *count* copies of a VIOFONTCELLSIZE structure.

If *plOutCount* is less than the size of a LONG data type, *plOutCount* is updated to zero, and nothing is returned in the buffer pointed to by *pbOutData*.

If *plOutCount* is equal to the size of a LONG data type, *pbOutData* returns the number of VIO cell sizes that can be returned by this escape. The buffer pointed to by *pbOutData* is updated so that *maxcount* is the number of VIO cell sizes that can be returned.

If *plOutCount* is greater than the size of a LONG data type, *pbOutData* returns the VIO cell sizes that are supported. The buffer pointed to by *pbOutData* is updated so that:

- *maxcount* is the number of VIO cell sizes that can be returned
- *count* is the number of VIO cell sizes returned (may be zero if *plOutCount* is equal to twice the size of a LONG data type)
- *count count* copies of a VIOFONTCELLSIZE structure are returned.

DEVESC_SETMODE

Sets the printer into a particular mode. It is optional for printer drivers to support this escape, but those that do support it need to be aware of the code page of any built-in fonts. For example, if only code page 437 is built in, it is used if 437 is requested by DEVESC_SETMODE; however, if code page 865 is requested, a suitable code page/font could be downloaded.

This escape is metafiled and recorded.

llnCount Number of bytes pointed to by *pbInData*
pbInData Buffer contains an ESCSETMODE structure
plOutCount Not used; can be set to 0
pbOutData Not used; can be set to null.

Related Functions

Prerequisite Functions

- DevOpenDC

Related Functions

- GpiAssociate (for DEVESC_STARTDOC)
- GpiErase (for DEVESC_NEWFRAME)

Graphic Elements and Orders

DevEscape functions generate orders only when metafiling.

Order **GEESCP**

Example Code

This example uses DevEscape to access facilities of a device that would otherwise be unavailable through the normal Device API set. Here, a new page in a print job is started.

```
#define INCL_DEV        /* Device Function definitions  */
#include <os2.h>

LONG  lResult;          /* Error code or not implemented
                           warning code                    */
HDC   hdc;              /* Device-context handle           */
LONG  plOutCount;       /* length of output buffer(input),
                           number of bytes returned(output)*/
PBYTE pbOutData;        /* output buffer                   */

/* for the NEWFRAME, input and output buffers are not used,
   so set the buffer lengths to zero(0) and set the buffers to
   NULL */
plOutCount = 0;
pbOutData = NULL;

lResult = DevEscape(hdc, DEVESC_NEWFRAME, 0L, NULL, &plOutCount,
                    pbOutData);
```

DevOpenDC

This function creates a device context.

Syntax

```
#define INCL_DEV /* Or use INCL_PM, Also in COMON section */
#include <os2.h>
HDC DevOpenDC (HAB hab, LONG lType, PSZ pszToken, LONG lCount,
               PDEVOPENDATA pdopData, HDC hdcComp)
```

Parameters

hab (HAB) – input
Anchor-block handle.

lType (LONG) – input
Type of device context:

OD_QUEUED	A device, such as a printer or plotter, for which output is to be queued. Certain restrictions apply for this device type; see "Metafile Specification" in the *Presentation Manager Programming Reference Volume II*.
OD_DIRECT	A device, such as a printer or plotter, for which output is not to be queued.
OD_INFO	A device, such as a printer or plotter, but the device context is used only to retrieve information (for example, font metrics). Drawing can be performed to a presentation space associated with such a device context, but no output medium is updated.
OD_METAFILE	The device context is used to write a metafile. The presentation page defines the area of interest within the picture in the metafile. See OD_METAFILE_NOQUERY. Certain restrictions apply for this device type; see "Metafile Specification" in the *Presentation Manager Programming Reference Volume II*.
OD_METAFILE_NOQUERY	The device context is used to write a metafile. Functionally, this device type is the same as OD_METAFILE, except that querying of attributes is not allowed with a presentation space while it is associated with an OD_METAFILE_NOQUERY device context. If querying of attributes is not required,

OD_METAFILE_NOQUERY should be used in preference to OD_METAFILE, since it gives improved performance.

Certain restrictions apply for this device type; see "Metafile Specification" in the *Presentation Manager Programming Reference Volume II*.

OD_MEMORY — A device context that is used to contain a bit map. The *hdcComp* parameter identifies a device with which the memory device context is to be compatible.

pszToken (PSZ) – input
Device-information token.

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

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

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

lCount (LONG) – input
Number of items.

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

pdopData (PDEVOPENDATA) – input
Open-device-context data area.

hdcComp (HDC) – input
Compatible-device-context handle.

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

If *hdcComp* is NULLHANDLE, compatibility with the screen is assumed.

Returns

hdc (HDC) – returns
Device-context handle:

DEV_ERROR	Error
<>0	Device-context handle.

Possible returns from WinGetLastError

PMERR_INV_DC_TYPE (0x2060)	An invalid type parameter was specified with DevOpenDC, or a function was issued that is invalid for a OD_METAFILE_NOQUERY device context.

PMERR_INV_LENGTH_OR_COUNT (0x2092)	An invalid length or count parameter was specified.
PMERR_INV_DC_DATA (0x205F)	An invalid data parameter was specified with DevOpenDC.
PMERR_INV_HDC (0x207C)	An invalid device-context handle or (micro presentation space) presentation-space handle was specified.
PMERR_INV_DRIVER_NAME (0x2067)	A driver name was specified which has not been installed.
PMERR_INV_LOGICAL_ADDRESS (0x2097)	An invalid device logical address was specified.

Remarks

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

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

The device context is owned by the process from which DevOpenDC is issued. It cannot be accessed directly from any other process. If it still exists when the process terminates, it is automatically deleted by the system. When using a device context type of OD_METAFILE_NOQUERY the querying of attributes is not allowed. To improve performance of this type of metafile no error checking is performed to ensure that such API calls are not attempted. Query calls are accepted but the results returned are undefined.

This function requires the existence of a message queue.

Related Functions

Prerequisite Functions

- WinInitialize

Related Functions

- DevCloseDC
- PrfQueryProfileString
- WinOpenWindowDC
- WinQueryWindow

Example Code

This example calls DevOpenDC to create a memory device context with screen compatibility and then associates that context with a newly created presentation space.

```
#define INCL_DEV                /* Device Function definitions  */
#define INCL_GPICONTROL         /* GPI control Functions        */
#include <os2.h>

HDC   hdc;              /* Device-context handle                */
HAB   hab;              /* Anchor-block handle                  */
 /* context data structure */
DEVOPENSTRUC dop = {NULL, "DISPLAY", NULL, NULL, NULL, NULL,
                    NULL, NULL, NULL};
HPS    hps;             /* presentation-space handle            */
SIZEL  sizl={0, 0};     /* use same page size as device        */

/* create memory device context */
hdc = DevOpenDC(hab, OD_MEMORY, "*", 5L, (PDEVOPENDATA)&dop, NULLHANDLE);

/* create a presentation space associated with the context */
hps = GpiCreatePS(hab, hdc, &sizl, GPIA_ASSOC | PU_PELS);
```

DevPostDeviceModes

This function returns, and optionally sets job properties.

Syntax

```
#define INCL_DEV /* Or use INCL_PM, */
#include <os2.h>
LONG DevPostDeviceModes  (HAB hab, PDRIVDATA pdrivDriverData,
                          PSZ pszDriverName, PSZ pszDeviceName,
                          PSZ pszName, ULONG flOptions)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pdrivDriverData (PDRIVDATA) – in/out
Driver data.

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

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

pszDriverName (PSZ) – input
Device-driver name. A string containing the name of the presentation driver; for example, "LASERJET."

pszDeviceName (PSZ) – input
Device-type name.

Null-terminated string in a 32-byte field, identifying the device type; for example, "HP LaserJet IID" (model number). Valid names are defined by device drivers.

Note: This parameter always overrides the data in the *szDeviceName* field of the DRIVDATA structure, passed in the *pdrivDriverData* parameter.

pszName (PSZ) – input
Device name.

A name that identifies the device; for example, "PRINTER1." If DPDM_POSTJOBPROP is specified in the *flOptions* parameter, the *pszName* parameter can be NULL.

flOptions (ULONG) – input
Dialog options.

Options that control whether a dialog is displayed.

DPDM_POSTJOBPROP

This function allows the user to set properties for the print job by displaying a dialog and returning the updated job properties. Examples of job properties are paper size, paper orientation, and single-sided or duplex.

The printer is configured in the shell using a dialog provided by the presentation driver. The configuration describes the actual printer setup such as number of paper bins, available paper sizes, and any installed hardware fonts.

Before the job properties dialog is displayed the presentation driver merges any changes in the printer configuration with the data passed in the *pdrivDriverData* parameter. This allows, for example new paper sizes to be added into the job properties dialog. The parameter *pszName* can be specified as NULL although this is not recommended because the presentation driver cannot easily find the printer configuration to merge.

It is the responsibility of the application to retrieve and store job properties. An application can choose to store job properties either on a per document or per application basis. The job properties can then be passed into DevOpenDC. Initial (default) job properties can be retrieved using DPDM_QUERYJOBPROP option.

The application cannot tell if the user modified the job properties or just cancelled the dialog. Hence the job properties returned in the *pdrivDriverData* parameter must always be stored.

The shell allows users to specify default job properties for a printer. The spooler API SplQueryQueue can be used to retrieve these defaults. The spooler automatically adds the default job properties for a printer to any jobs that are submitted without job properties.

DPDM_QUERYJOBPROP

Do not display a dialog. Return the default job properties. These defaults are derived from the defaults for the chosen device; for example, "HP Laserjet IID" and the printer setup specified via the shell printer driver configuration dialog.

Returns

lDriverCount (LONG) – returns
Size/error indicator.

Value depends on what was passed as the pointer to *pdrivDriverData*. If NULL was passed, the following values are possible:

DPDM_ERROR	Error
DPDM_NONE	No settable options
>0	Size in bytes required for *pdrivDriverData*.

If any other value was passed, the following values are possible:

DPDM_ERROR	Error
DPDM_NONE	No settable options
DEV_OK	OK.

Possible returns from WinGetLastError

PMERR_INV_DRIVER_DATA (0x2066)	Invalid driver data was specified.
PMERR_DRIVER_NOT_FOUND (0x2026)	The device driver specified with DevPostDeviceModes was not found.
PMERR_INV_DEVICE_NAME (0x2061)	An invalid devicename parameter was specified with DevPostDeviceModes.
PMERR_INV_LOGICAL_ADDRESS (0x2097)	An invalid device logical address was specified.

Remarks

An application can first call this function with a NULL data pointer to find out how much storage is needed for the data area. Having allocated the storage, the application can then make the call a second time for the data to be entered. The returned data can then be passed in DevOpenDC as *pdrivDriverData* within the *pdopData* parameter.

Calling this function requires the existence of a message queue.

Use SplEnumDevice or SplEnumPrinter with *flType* set to SPL_PR_DIRECT_DEVICE or SPL_PR_QUEUED_DEVICE to get a list of all the devices.

To get information about a specific device use SplQueryDevice.

Related Functions

- DevOpenDC

Example Code

This example shows how to call DevPostDeviceModes and allocate a new buffer, if necessary, for the larger job properties (DRIVDATA structure).

```
#define INCL_DEV
#define INCL_DOS
#include <os2.h>
#include <memory.h>

{
    ULONG          devrc=FALSE;
    HAB            hab;
    PSZ            pszPrinter;
    HDC            hdc=NULL;
    PDRIVDATA      pOldDrivData;
    PDRIVDATA      pNewDrivData=NULL;
    PDEVOPENSTRUC dops;
    LONG           buflen;

/* check size of buffer required for job properties */
    buflen = DevPostDeviceModes( hab,
                                 NULL,
                                 dops->pszDriverName,
                                 dops->pdriv->szDeviceName,
                                 pszPrinter,
                                 DPDM_POSTJOBPROP
                               );

/* return error to caller */
    if (buflen<=0)
        return(buflen);

/* allocate some memory for larger job properties and */
/* return error to caller */

    if (buflen != dops->pdriv->cb)
    {
        if (DosAllocMem((PPVOID)&pNewDrivData,buflen,fALLOC))
            return(DPDM_ERROR);
    }

/* copy over old data so driver can use old job */
/* properties as base for job properties dialog */
    pOldDrivData = dops->pdriv;
    dops->pdriv = pNewDrivData;
    memcpy( (PSZ)pNewDrivData, (PSZ)pOldDrivData, pOldDrivData->cb );
```

```
/* display job properties dialog and get updated */
/* job properties from driver */

    devrc = DevPostDeviceModes( hab,
                                dops->pdriv,
                                dops->pszDriverName,
                                dops->pdriv->szDeviceName,
                                pszPrinter,
                                DPDM_POSTJOBPROP
                              );
    return(devrc);
}
```

DevQueryCaps

This function queries the device characteristics.

Syntax

```
#define INCL_DEV /* Or use INCL_PM, Also in COMON section */
#include <os2.h>
BOOL DevQueryCaps (HDC hdc, LONG lStart, LONG lCount, PLONG alArray)
```

Parameters

hdc (HDC) – input
Device-context handle.

lStart (LONG) – input
First item of information.

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

lCount (LONG) – input
Count of items of information.

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

alArray (PLONG) – output
Device capabilities.

Array of *lCount* elements, starting with *lStart*. The array elements are numbered consecutively, starting with CAPS_FAMILY. The element number constants start with 0. See the appropriate bindings reference.

If *lStart* + *lCount* –1 exceeds the current highest-defined element number, elements beyond the highest are returned as 0.

CAPS_FAMILY
Device type (values as for *lType* in DevOpenDC).

CAPS_IO_CAPS
Device input/output capability:

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

CAPS_TECHNOLOGY
Technology:

CAPS_TECH_UNKNOWN	Unknown
CAPS_TECH_VECTOR_PLOTTER	Vector plotter

CAPS_TECH_RASTER_DISPLAY	Raster display
CAPS_TECH_RASTER_PRINTER	Raster printer
CAPS_TECH_RASTER_CAMERA	Raster camera
CAPS_TECH_POSTSCRIPT	PostScript** device.

CAPS_DRIVER_VERSION
Version identifier of the presentation driver.

The high order word of the version identifier is 0. The low order word identifies the release, for example 0x0120 is release 1.2.

CAPS_WIDTH
Media width (for a full screen, maximized window for displays) in pels.

CAPS_HEIGHT
Media depth (for a full screen, maximized window for displays) in pels. (For a plotter, a pel is defined as the smallest possible displacement of the pen and can be smaller than a pen width.)

CAPS_WIDTH_IN_CHARS
Media width (for a full screen, maximized window for displays) in default character columns.

CAPS_HEIGHT_IN_CHARS
Media depth (for a full screen, maximized window for displays) in default character rows.

CAPS_HORIZONTAL_RESOLUTION
Horizontal resolution of device in pels per meter.

CAPS_VERTICAL_RESOLUTION
Vertical resolution of device in pels per meter.

CAPS_CHAR_WIDTH
Default character-box width in pels for VIO.

CAPS_CHAR_HEIGHT
Default character-box height in pels for VIO.

CAPS_SMALL_CHAR_WIDTH
Default small-character box width in pels for VIO. This is 0 if there is only one character-box size.

CAPS_SMALL_CHAR_HEIGHT
Default small-character box height in pels for VIO. This is 0 if there is only one character-box size.

CAPS_COLORS
Number of distinct colors supported at the same time, including reset (gray scales count as distinct colors). If loadable color tables are supported, this is the number of entries in the device color table. For plotters, the value returned is the number of pens plus one (for the background).

CAPS_COLOR_PLANES
Number of color planes.

CAPS_COLOR_BITCOUNT
Number of adjacent color bits for each pel (within one plane).

CAPS_COLOR_TABLE_SUPPORT
Loadable color table support:

CAPS_COLTABL_RGB_8	1 if RGB color table can be loaded, with a minimum support of 8 bits each for red, green, and blue.
CAPS_COLTABL_RGB_8_PLUS	1 if color table with other than 8 bits for each primary color can be loaded.
CAPS_COLTABL_TRUE_MIX	1 if true mixing occurs when the logical color table has been realized, providing that the size of the logical color table is not greater than the number of distinct colors supported (see element CAPS_COLORS).
CAPS_COLTABL_REALIZE	1 if a loaded color table can be realized.

CAPS_MOUSE_BUTTONS
The number of pointing device buttons that are available. A returned value of 0 indicates that there are no pointing device buttons available.

CAPS_FOREGROUND_MIX_SUPPORT
Foreground mix support:

CAPS_FM_OR	Logical OR.
CAPS_FM_OVERPAINT	Overpaint.
CAPS_FM_XOR	Logical XOR.
CAPS_FM_LEAVEALONE	Leave alone.
CAPS_FM_AND	Logical AND.
CAPS_FM_GENERAL_BOOLEAN	All other mix modes; see "GpiSetMix" in *Graphics Programming Interface Programming Reference*.

The value returned is the sum of the values appropriate to the mixes supported. A device capable of supporting OR must, as a minimum, return CAPS_FM_OR + CAPS_FM_OVERPAINT + CAPS_FM_LEAVEALONE, signifying support for the mandatory mixes OR, overpaint, and leave-alone.

Note that these numbers correspond to the decimal representation of a bit string that is six bits long, with each bit set to 1 if the appropriate mode is supported.

Those mixes returned as supported are guaranteed for all primitive types. For more information, see "GpiSetMix" in *Graphics Programming Interface Programming Reference*.

CAPS_BACKGROUND_MIX_SUPPORT
Background mix support:

CAPS_BM_OR	Logical OR.
CAPS_BM_OVERPAINT	Overpaint.

CAPS_BM_XOR	Logical XOR.
CAPS_BM_LEAVEALONE	Leave alone.
CAPS_BM_AND	Logical AND.
CAPS_BM_GENERAL_BOOLEAN	All other mix modes; see "GpiSetMix" in *Graphics Programming Interface Programming Reference*.
CAPS_BM_SRCTRANSPARENT	Provides a transparent overlay function by not copying pels from the source bit map to the output bit map if they match the presentation space background color.
CAPS_BM_DESTTRANSPARENT	Provides a transparent underlay function by copying only the pels that match the presentation space background color from the source bit map to the output bit map.

The value returned is the sum of the values appropriate to the mixes supported. A device must, as a minimum, return CAPS_BM_OVERPAINT + CAPS_BM_LEAVEALONE, signifying support for the mandatory background mixes overpaint, and leave-alone.

Note that these numbers correspond to the decimal representation of a bit string that is four bits long, with each bit set to 1 if the appropriate mode is supported.

Those mixes returned as supported are guaranteed for all primitive types. For more information, see "GpiSetMix" in *Graphics Programming Interface Programming Reference*.

CAPS_VIO_LOADABLE_FONTS
Number of fonts that can be loaded for VIO.

CAPS_WINDOW_BYTE_ALIGNMENT
Whether or not the client area of VIO windows should be byte-aligned:

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

CAPS_BITMAP_FORMATS
Number of bit-map formats supported by device.

CAPS_RASTER_CAPS
Capability for device raster operations:

CAPS_RASTER_BITBLT	1 if GpiBitBlt and GpiWCBitBlt is supported.
CAPS_RASTER_BANDING	1 if banding is supported
CAPS_RASTER_BITBLT_SCALING	1 if GpiBitBlt and GpiWCBitBlt with scaling is supported.

CAPS_RASTER_SET_PEL — 1 if GpiSetPel is supported.

CAPS_RASTER_FONTS — 1 if this device can draw raster fonts.

CAPS_RASTER_FLOOD_FILL — 1 if GpiFloodFill is supported.

CAPS_MARKER_HEIGHT
Default marker-box height in pels.

CAPS_MARKER_WIDTH
Default marker-box width in pels.

CAPS_DEVICE_FONTS
Number of device-specific fonts.

CAPS_GRAPHICS_SUBSET
Graphics drawing subset supported. (3 indicates GOCA DR/3)

CAPS_GRAPHICS_VERSION
Graphics architecture version number supported. (1 indicates Version 1)

CAPS_GRAPHICS_VECTOR_SUBSET
Graphics vector drawing subset supported. (2 indicates GOCA VS/2)

CAPS_DEVICE_WINDOWING
Device windowing support:

CAPS_DEV_WINDOWING_SUPPORT — 1 if device supports windowing.

Other bits are reserved 0.

CAPS_ADDITIONAL_GRAPHICS
Additional graphics support:

CAPS_GRAPHICS_KERNING_SUPPORT — 1 if device supports kerning.

CAPS_FONT_OUTLINE_DEFAULT — 1 if device has a default outline font.

CAPS_FONT_IMAGE_DEFAULT — 1 if device has a default image font.

CAPS_SCALED_DEFAULT_MARKERS — 1 if default markers are to be scaled by the marker-box attribute.

CAPS_COLOR_CURSOR_SUPPORT — 1 if device supports colored cursors.

CAPS_PALETTE_MANAGER — 1 if device supports palette functions (see "GpiCreatePalette" in the *Graphics Programming Interface Programming Reference*).

CAPS_COSMETIC_WIDELINE_SUPPORT — 1 if device supports cosmetic thick lines (see "GpiSetLineWidth" in the *Graphics Programming*

Interface Programming Reference).

Other bits are reserved 0.

CAPS_PHYS_COLORS

Maximum number of distinct colors available on the device.

CAPS_COLOR_INDEX

Maximum logical color-table index supported for this device. For the EGA and VGA drivers, the value is 63.

CAPS_GRAPHICS_CHAR_WIDTH

Default graphics character-box width, in pels.

CAPS_GRAPHICS_CHAR_HEIGHT

Default graphics character-box height, in pels.

CAPS_HORIZONTAL_FONT_RES

Effective horizontal device resolution in pels per inch, for the purpose of selecting fonts.

For printers, this is the actual device resolution, but for displays it may differ from the actual resolution for reasons of legibility.

CAPS_VERTICAL_FONT_RES

Effective vertical device resolution in pels per inch, for the purpose of selecting fonts.

CAPS_DEVICE_FONT_SIM

Identifies which simulations are valid on device fonts.

Valid flags are:

CAPS_DEV_FONT_SIM_BOLD
CAPS_DEV_FONT_SIM_ITALIC
CAPS_DEV_FONT_SIM_UNDERSCORE
CAPS_DEV_FONT_SIM_STRIKEOUT

CAPS_LINEWIDTH_THICK

Cosmetic thickness of lines and arcs on this device, when *fxLineWidth* is LINEWIDTH_THICK (see "GpiSetLineWidth" in the *Graphics Programming Interface Programming Reference*). The units are pels. A value of 0 is interpreted as 2 pels.

CAPS_DEVICE_POLYSET_POINTS

Number of points in a polyset that a device can handle.

Returns

rc (BOOL) – returns

Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INV_HDC (0x207C)	An invalid device-context handle or (micro presentation space) presentation-space handle was specified.
PMERR_INV_QUERY_ELEMENT_NO (0x20BC)	An invalid start parameter was specified with DevQueryCaps.
PMERR_INV_LENGTH_OR_COUNT (0x2092)	An invalid length or count parameter was specified.

Remarks

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

Related Functions

Prerequisite Functions

- DevOpenDC (for CAPS_FAMILY)

Related Functions

- DevQueryDeviceNames
- DevQueryHardcopyCaps

Example Code

In this example the driver is queried to see if it supports input, output, or both. Note that a valid device context handle must be passed. This example assumes a DevOpenDC call has been made to obtain the device context handle.

```
#define INCL_DEV
#include <OS2.H>

HDC hdc;
LONG lStart;
LONG lCount;
BOOL flreturn;
LONG alArray[CAPS_TECHNOLOGY];
lCount = CAPS_TECHNOLOGY;
lStart = CAPS_FAMILY;

flreturn = DevQueryCaps(hdc,        /* device context handle */
                        lStart,     /* number of first item  */
                        lCount,     /* count of items        */
                        alArray);   /* array of longs which  */
                                    /* will contain the return */
                                    /* information.          */

switch(alArray[CAPS_IO_CAPS])       /* we test the CAPS_IO_CAPS */
                                    /* element of the array to  */
                                    /* find out which options   */
{                                   /* are supported.           */
 case CAPS_IO_SUPPORTS_OP:             /* device supports output.*/

 break;
 case CAPS_IO_SUPPORTS_IP:           /* device supports input. */

 break;
 case CAPS_IO_SUPPORTS_IO:           /* device supports both  */
                                    /* input and output.       */
 break;
 default:
 break;
}
```

DevQueryDeviceNames

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

Syntax

```
#define INCL_DEV /* Or use INCL_PM, */
#include <os2.h>
BOOL DevQueryDeviceNames  (HAB hab, PSZ pszDriverName, PLONG pldn,
                           PSTR32 aDeviceName, PSTR64 aDeviceDesc,
                           PLONG pldt, PSTR16 aDataType)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pszDriverName (PSZ) – input
Fully-qualified name of the file containing the presentation driver.

The file-name extension is DRV.

pldn (PLONG) – in/out
Maximum number of device names and descriptions that can be returned.

On input, it must be greater or equal to 0. *pldn* can have the following values:

Zero — The number of device names and descriptions supported is returned; *aDeviceName* and *aDeviceDesc* are not updated.

Nonzero — *pldn* is updated to the number returned in *aDeviceName* and *aDeviceDesc*; *aDeviceName* and *aDeviceDesc* are updated.

aDeviceName (PSTR32) – output
Device-name array.

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

aDeviceDesc (PSTR64) – output
Device-description array.

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

pldt (PLONG) – in/out
Maximum number of data types that can be returned.

On input, it must be greater or equal to 0. *pldt* can have the following values:

Zero	The number of data types supported is returned, and *aDataType* is not updated.
Nonzero	*pldt* is updated to the number returned, and *aDataType* is updated.

aDataType (PSTR16) – output
Data type array.

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

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INV_LENGTH_OR_COUNT (0x2092)	An invalid length or count parameter was specified.

Remarks

An application can first call this function with *pldn* and *pldt* set to 0 to find how much storage is needed for the data areas. Having allocated the storage, the application calls the function a second time for the data to be entered.

"HP Laserjet IID" is an example of a device name, "Hewlett-Packard Laserjet IID" is an example of a device description, and "PM_Q_STD" is an example of a data type.

Related Functions

- DevQueryCaps
- DevQueryHardcopyCaps

Example Code

This example uses DevQueryDeviceNames to return the names, descriptions, and data types of supported devices for a presentation driver. The first call to DevQueryDeviceNames determines the number of names, description, and data types available; after allocating the arrays, the second call actually returns the information in the arrays.

```
#define INCL_DEV                /* Device Function definitions  */
#define INCL_DOSMEMMGR          /* DOS Memory Manager Functions */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
HAB   hab;              /* Anchor-block handle                  */
LONG  pldn = 0L;        /* number of device names/descriptions  */
LONG  pldt = 0L;        /* number of data types                 */
PSTR32 aDeviceName;     /* array of device names                */
PSTR64 aDeviceDesc;     /* array of device descriptions         */
PSTR16 aDataType;       /* array of data types                  */

/* query number of supported names/descriptions/data types
   (pldn & pldt both 0) */
fSuccess = DevQueryDeviceNames(hab, "IBM4201.DRV", &pldn,
                               aDeviceName, aDeviceDesc, &pldt,
                               aDataType);

if (fSuccess)
   {
   /* allocate arrays */
   DosAllocMem((VOID *)aDeviceName, (ULONG)pldn*sizeof(STR32),
               PAG_COMMIT | PAG_WRITE);
   DosAllocMem((VOID *)aDeviceDesc, (ULONG)pldn*sizeof(STR64),
               PAG_COMMIT | PAG_WRITE);
   DosAllocMem((VOID *)aDataType, (ULONG)pldt*sizeof(STR16),
               PAG_COMMIT | PAG_WRITE);

   /* query supported device information */
   fSuccess = DevQueryDeviceNames(hab, "IBM4201.DRV", &pldn,
                                  aDeviceName, aDeviceDesc, &pldt,
                                  aDataType);
   }
```

DevQueryHardcopyCaps

This function queries the hard-copy capabilities of a device.

Syntax

```
#define INCL_DEV /* Or use INCL_PM, */
#include <os2.h>
LONG DevQueryHardcopyCaps  (HDC hdc, LONG lStartForm, LONG lForms,
                            PHCINFO phciHcInfo)
```

Parameters

hdc (HDC) – input
Device-context handle.

lStartForm (LONG) – input
Start-forms code.

Forms-code number from which the query is to start. The first forms code has the value 0. *lStartForm* is used with *lForms*.

lForms (LONG) – input
Number of forms to query.

If 0, the number of forms codes defined is returned. If greater than zero, this function returns the number of forms codes for which information is returned.

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

phciHcInfo (PHCINFO) – output
Hard-copy capabilities information.

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

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

For a job to be selected by the spooler for printing, each one of the forms specified in the FORM spooler parameter (see *pszSpoolerParams* in DEVOPENSTRUC) must be either HCAPS_CURRENT or HCAPS_SELECTABLE. In other cases, the spooler holds the job with a "forms mismatch" error.

Returns

lFormsReturned (LONG) – returns
Details of forms:

DQHC_ERROR	Error.
>=0	If *lForms* equals 0, number of forms available. If *lForms* does not equal 0, number of forms returned.

Possible returns from WinGetLastError

PMERR_INV_HDC (0x207C)	An invalid device-context handle or (micro presentation space) presentation-space handle was specified.
PMERR_INV_FORMS_CODE (0x2076)	An invalid forms code parameter was specified with DevQueryHardcopyCaps.
PMERR_INV_LENGTH_OR_COUNT (0x2092)	An invalid length or count parameter was specified.

Related Functions

Prerequisite Functions

- DevOpenDC

Related Functions

- DevQueryCaps
- DevQueryDeviceNames

Example Code

The height and width of the capability of the output device is queried for each form code available. Note that a valid device context handle must be passed. This example assumes a DevOpenDC call has been made to obtain the device context handle of a say a printer.

```
#define INCL_DEV
#include <OS2.H>

HDC hdc;
LONG lStartForm;      /* Form code number from which the query */
                      /* is to start                           */
LONG lForms;          /* number of forms to query              */
/* array of structures containing return information. */
HCINFO ahciHcInfo[5];
LONG lreturn;
int i;
HCINFO height[5];
HCINFO width[5];

lStartForm = 0L;
lForms = 0L;           /* the actual number of forms codes is */
                       /* returned. There will be lreturn     */
                       /* copies of the HINFO structure.      */

lreturn = DevQueryHardcopyCaps(hdc,
                               lStartForm,
                               lForms,
                               ahciHcInfo);
if (lreturn > 5)
{
 lreturn = 5L;         /* we only want the first five form codes */
}                      /* if there are more than five            */

for(i = 0; i < lreturn; i++)
{
 width[lreturn].cx = ahciHcInfo[lreturn].cx;
 height[lreturn].cy = ahciHcInfo[lreturn].cy;
}
```

Chapter 3. Direct Manipulation Functions

This section describes functions that an application would use to initiate or participate in a direct manipulation operation. The following table shows all the direct manipulation (Drg) functions in alphabetic order.

C Name	C Name
DrgAcceptDroppedFiles	DrgQueryDragitemCount
DrgAccessDraginfo	DrgQueryDragitemPtr
DrgAddStrHandle	DrgQueryDragStatus
DrgAllocDraginfo	DrgQueryNativeRMF
DrgAllocDragtransfer	DrgQueryNativeRMFLen
DrgCancelLazyDrag	DrgQueryStrName
DrgDeleteDraginfoStrHandles	DrgQueryStrNameLen
DrgDeleteStrHandle	DrgQueryTrueType
DrgDrag	DrgQueryTrueTypeLen
DrgDragFiles	DrgReallocDragInfo
DrgFreeDraginfo	DrgReleasePS
DrgFreeDragtransfer	DrgSendTransferMsg
DrgGetPS	DrgSetDragImage
DrgLazyDrag	DrgSetDragitem
DrgLazyDrop	DrgSetDragPointer
DrgPostTransferMsg	DrgVerifyNativeRMF
DrgPushDraginfo	DrgVerifyRMF
DrgQueryDraginfoPtr	DrgVerifyTrueType
DrgQueryDraginfoPtrFromDragitem	DrgVerifyType
DrgQueryDraginfoPtrFromHwnd	DrgVerifyTypeSet
DrgQueryDragitem	

DrgAcceptDroppedFiles

This function handles the file direct manipulation protocol for a given window.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgAcceptDroppedFiles (HWND Hwnd, PSZ pPath, PSZ pTypes,
                            ULONG ulDefaultOp, ULONG ulReserved)
```

Parameters

Hwnd (HWND) – input
Handle of calling window.

pPath (PSZ) – input
Directory in which to place the dropped files.

If NULL, the files are placed in the current directory.

pTypes (PSZ) – input
List of types that are acceptable to the drop.

This string is of the form:

```
type[,type...]
```

When this pointer is NULL, any type of file will be accepted.

ulDefaultOp (ULONG) – input
Default drag operation for this window.

The operation is either DO_MOVE or DO_COPY.

ulReserved (ULONG) – input
Reserved.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

This function handles the file direct manipulation protocol for a given window. The window responds (DOR_DROP, *usDefaultOp*) to DM_DRAGOVER messages for items with a type matching the acceptable type string and with a rendering mechanism and format of <DRM_OS2FILE,DRF_UNKNOWN>. Not all dragged objects must match this criteria for the drop to be acceptable.

After the drop occurs, this function handles the conversation required to complete the direct manipulation operation for all acceptable objects. A DM_ENDCONVERSATION (DMFL_TARGETFAIL) message is sent to the source when an object is unacceptable.

When an error occurs during a move or copy, the caller is sent a DM_DRAGERROR message. The caller can take corrective action.

As the move or copy operation is successfully completed for each file, a DM_DRAGFILECOMPLETE message is sent to the caller. No message is sent when the operation fails.

The function returns TRUE if the operation is successful and FALSE if an error occurs.

Related Functions

- DrgDragFiles

Example Code

This example uses the DrgAcceptDroppedFiles function to define the direct manipulation protocol of the given window, accept all file types, and use the current directory as the drop directory.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL   fSuccess;        /* Indicate success or failure          */
HWND   Hwnd;            /* Handle of calling window             */
PSZ    pszPath;         /* Directory in which to place the      */
                        /* dropped files                        */
PSZ    pszTypes;        /* A list of types that are acceptable  */
ULONG  ulDefaultOp;     /* Default drag operation               */

pszPath = NULL;         /* Drop file in current directory       */
pszTypes = NULL;        /* Accept any file type                 */
ulDefaultOp = DO_MOVE;  /* Default drag operation is move       */

fSuccess = DrgAcceptDroppedFiles(Hwnd, pszPath, pszTypes,
                                 ulDefaultOp, 0);
```

DrgAccessDraginfo

This function accesses a DRAGINFO structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgAccessDraginfo  (PDRAGINFO pDraginfo)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_ACCESS_DENIED (0x150D)	The memory block was not allocated properly.

Remarks

This function is used by the target of a drag operation to access a DRAGINFO structure. The address of the structure is passed in a drag message (DM_DRAGOVER, DM_DROP, or DM_DROPHELP).

To release the structure, use the DrgFreeDraginfo function.

Related Functions

- DrgAllocDraginfo
- DrgDrag
- DrgFreeDraginfo
- DrgPushDraginfo

Example Code

This example uses the DrgAccessDraginfo function to make an existing drag information structure (created by the DrgAllocDraginfo function) available.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL      fSuccess;      /* Indicate success or failure          */
DRAGINFO  Draginfo;      /* Drag-information structure           */

fSuccess = DrgAccessDraginfo(&Draginfo);
```

DrgAddStrHandle

This function creates a handle to a string.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
HSTR DrgAddStrHandle  (PSZ pString)
```

Parameters

pString (PSZ) – input
String for which a handle is to be created.

Returns

hstr (HSTR) – returns
String handle.

NULLHANDLE	Error occurred.
Other	String handle created.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32768 to +32767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.
PMERR_RESOURCE_DEPLETION (0x20F9)	An internal resource depletion error has occurred.

Remarks

The handle can be used by any application to reference the input string.

This function must be called by the source of a drag whenever a string is to be passed in a DRAGINFO structure.

Related Functions

- DrgDeleteStrHandle
- DrgQueryStrName

Example Code

This example calls the DrgAddStrHandle function to create handles for strings that are used in a DRAGITEM structure.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions  */
#include <os2.h>

USHORT   ID_ITEM = 1;   /* Drag item identifier                 */
HWND     hwnd;          /* Window handle                        */
DRAGITEM ditem;         /* DRAGITEM structure                   */

                        /* Initialize the DRAGITEM structure    */
ditem.hwndItem = hwnd;          /* Conversation partner         */
ditem.ulItemID = ID_ITEM;       /* Identifies item being dragged */
ditem.hstrType = DrgAddStrHandle(DRT_TEXT);     /*  Item is text */
ditem.hstrRMF = DrgAddStrHandle("<DRM_OS2FILE,DRF_TEXT>");
ditem.hstrContainerName = DrgAddStrHandle("C:\\");
ditem.hstrSourceName = DrgAddStrHandle("C:\\CONFIG.SYS");
ditem.hstrTargetName = DrgAddStrHandle("C:\\OS2\\CONFIG.SYS");
ditem.cxOffset = 0;             /* X-offset of the origin of the */
                                /* image from the pointer hotspot*/
ditem.cyOffset = 0;             /* Y-offset of the origin of the */
                                /* image from the pointer hotspot*/
ditem.fsControl = 0;            /* Source item control flags    */
                                /* object is open               */
ditem.fsSupportedOps = 0;
```

DrgAllocDraginfo

This function allocates a DRAGINFO structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
PDRAGINFO DrgAllocDraginfo (ULONG cDitem)
```

Parameters

cDitem (ULONG) – input
Number of objects being dragged.

This number must be greater than 0.

Returns

Draginfo (PDRAGINFO) – returns
Pointer to the DRAGINFO structure.

NULL	Error occurred.
Other	The DRAGINFO structure.

Possible returns from WinGetLastError

PMERR_INSUFFICIENT_MEMORY (0x203E)	The operation terminated through insufficient memory.
PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range –32768 to +32767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

Remarks

This function must be called before the DrgDrag function is called.

The caller can define a default operation for the objects represented by the DRAGINFO structure by modifying the *usOperation* field. If the *usOperation* field is modified, the new value will be sent to the target as the operation whenever a DO_DEFAULT operation would normally be sent. The caller should not modify any other part of the DRAGINFO structure. The DRAGITEM structures associated with the DRAGINFO structure should only be altered with DrgSetDragitem or by using a pointer obtained with DrgQueryDragitemPtr.

Related Functions

- DrgAccessDraginfo
- DrgDrag
- DrgFreeDraginfo
- DrgPushDraginfo

Example Code

This example calls the DrgAllocDraginfo function to create a Drag structure for a single object and uses the new structure to set the DRAGITEM (DrgSetDragitem) structure.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#include <os2.h>

PDRAGINFO  pdinfo;       /* Pointer to DRAGINFO structure       */
HWND       hwnd;         /* Handle of calling (source) window   */
BOOL       flResult;     /* Result indicator                    */
DRAGITEM   ditem;        /* DRAGITEM structure                  */

pdinfo = DrgAllocDraginfo(1); /* Create the DRAGINFO structure  */
                              /* Set the drag item              */
flResult= DrgSetDragitem(pdinfo, &ditem, (ULONG)sizeof(ditem), 0);
```

DrgAllocDragtransfer

This function allocates a specified number of DRAGTRANSFER structures from a single segment.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
PDRAGTRANSFER DrgAllocDragtransfer  (ULONG cdxfer)
```

Parameters

cdxfer (ULONG) – input
Number of DRAGTRANSFER structures to be allocated.

This number must be greater than 0.

Returns

pDragtransfer (PDRAGTRANSFER) – returns
Pointer to an array of DRAGTRANSFER structures.

NULL	Error occurred.
Other	The array of DRAGTRANSFER structures.

Possible returns from WinGetLastError

PMERR_MEMORY_ALLOCATION_ERR (0x1112)	An error occurred during memory management.
PMERR_INSUFFICIENT_MEMORY (0x203E)	The operation terminated through insufficient memory.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Remarks

This function must be called before sending a DM_RENDER message.

Related Functions

- DrgFreeDragtransfer
- DrgSendTransferMsg

Example Code

This example calls the DrgAllocDragtransfer function to allocate a single DRAGTRANSFER structure and adds a pointer to a DRAGITEM structure for an object that will be transferred.

```
#define INCL_WINSTDDRAG   /* Direct Manipulation (Drag) Functions*/
#include <os2.h>

PDRAGTRANSFER  pResult;   /* Pointer to DRAGTRANSFER structure   */
PDRAGITEM      pDragitem; /* Pointer to DRAGITEM structure       */

pResult = DrgAllocDragtransfer(1);

if (pResult != NULL)      /* Indicate DRAGITEM to be transferred */
    pResult->pditem = pDragitem;
```

DrgCancelLazyDrag

This function is called to cancel the current drag operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgCancelLazyDrag ()
```

Parameters

None.

Returns

rc (BOOL) – returns
Success indicator.

Possible values are described in the following list:

TRUE Lazy drag is successfully canceled.
FALSE An error occurred.

Remarks

This function posts the DM_DROPNOTIFY message to the source window, specifying a target window handle of zero in *hwndTarget*. The source window must then free DRAGINFO using DrgFreeDraginfo.

Example Code

This example shows the canceling of a lazy drag operation.

```
#define  INCL_WINSTDDRAG
#include <os2.h>

BOOL       bResult;        /* Return code from API */

bResult=DrgCancelLazyDrag();
```

DrgDeleteDraginfoStrHandles

This function deletes each unique string handle in a DRAGINFO structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgDeleteDraginfoStrHandles  (PDRAGINFO  pDraginfo)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure that contains string handles to delete.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32768 to +32767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

Remarks

Using this function is equivalent to calling the DrgDeleteStrHandle function for each unique string in a DRAGINFO structure.

This function must be called by the target of a direct manipulation operation either:

- After processing a DM_DROPHELP message
  or
- After completing the direct manipulation operation begun as a result of a DM_DROP message.

Related Functions

- DrgDeleteStrHandle

Example Code

This example calls the DrgDeleteDraginfoStrHandles function to delete all unique string handles associated with the specified DRAGINFO structure (previously allocated by the DrgAllocDraginfo function).

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL      fSuccess;     /* Indicate success or failure          */
DRAGINFO  Draginfo;     /* DRAGINFO structure containing string */
                        /* handles to delete                    */

fSuccess = DrgDeleteDraginfoStrHandles (&Draginfo);
```

DrgDeleteStrHandle

This function deletes a string handle.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgDeleteStrHandle  (HSTR Hstr)
```

Parameters

Hstr (HSTR) – input
The string handle to delete.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS (0x1208)

Remarks

This function must be used to delete a string handle created by the DrgAddStrHandle function.

Related Functions

- DrgAddStrHandle
- DrgDeleteDraginfoStrHandles

Example Code

This example calls the DrgDeleteStrHandle function to delete an existing string handle (returned by a previous call to the DrgAddStrHandle function).

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL  fSuccess;         /* Indicate success or failure          */
HSTR  Hstr;             /* String handle                        */

fSuccess = DrgDeleteStrHandle (Hstr);
```

DrgDrag

This function performs a drag operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
HWND DrgDrag (HWND hwndSource, PDRAGINFO pDraginfo,
              PDRAGIMAGE pdimg, ULONG cdimg, LONG vkTerminate,
              PVOID pReserved)
```

Parameters

hwndSource (HWND) – input
Handle of the source window calling this function.

pDraginfo (PDRAGINFO) – in/out
Pointer to the DRAGINFO structure.

pdimg (PDRAGIMAGE) – input
Pointer to an array of DRAGIMAGE structures.

These structures describe the images that are to be drawn under the direct manipulation pointer during the drag.

cdimg (ULONG) – input
Number of DRAGIMAGE structures in the *pdimg* array. Must be > 0.

vkTerminate (LONG) – input
Pointing device button that ends the drag operation.

Possible values are described in the following list:

VK_BUTTON1	Release of button 1 ends the drag.
VK_BUTTON2	Release of button 2 ends the drag.
VK_BUTTON3	Release of button 3 ends the drag.
VK_ENDDRAG	Release of the system-defined direct manipulation button ends the drag. This is the recommended value if the DrgDrag function call is invoked in response to a WM_BEGINDRAG message.

pReserved (PVOID) – input
Reserved value, must be NULL.

Returns

hwndDest (HWND) – returns

Handle of window on which the dragged objects were dropped.

A return value of NULL indicates that an error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range –32 768 to +32 767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.
PMERR_INSUFFICIENT_MEMORY (0x203E)	The operation terminated through insufficient memory.

Remarks

This function:

- Captures the mouse to the current thread
- Initiates a direct manipulation operation
- Uses the DRAGIMAGE structure to provide visual feedback to the user during the drag operation
- Notifies other windows as the drag object passes over
- Notifies the destination if the object is dropped
- Releases mouse capture.

Note: DrgDrag will fail if it is unable to capture the mouse. For example, if another window in the same thread has already set the capture.

Before invoking DrgDrag, the caller is responsible for:

- Obtaining a DRAGINFO structure using DrgAllocDraginfo
- Initializing the DRAGITEM structures using DrgSetDragitem.

DrgDrag is called when the system-defined direct-manipulation button is pressed while the pointer is over a window and a pointing device movement follows. As the pointer moves over a potential target, a DM_DRAGOVER message is sent to the target. When the pointer moves from one target window to another, a DM_DRAGLEAVE message is sent to the former target.

If the pointer is over a valid target when the direct-manipulation button is released, a DM_DROP message is sent to the target.

Before the DM_DROP message is sent, the *cxOffset* and *cyOffset* fields are copied from the DRAGIMAGE structures to the corresponding fields in the DRAGITEM structures. The values from the first DRAGIMAGE are copied to the first DRAGITEM, from the second DRAGIMAGE to the second DRAGITEM, and so on. The target can use this information to place the images in the same spatial relationship after the drop. If there are more DRAGITEM structures than there are DRAGIMAGE structures, the *cxOffset* and *cyOffset* from the final DRAGIMAGE are placed in each of the remaining DRAGITEM structures.

The caller can define a default operation for the objects represented by the DRAGINFO structure by modifying the *usOperation* field. If the *usOperation* field is modified, the new value will be sent to the target as the operation whenever a DO_DEFAULT operation would normally be sent. The caller should not modify any other part of the DRAGINFO structure. The DRAGITEM structures associated with the DRAGINFO structure should only be altered with DrgSetDragitem or by using a pointer obtained with DrgQueryDragitemPtr.

The following keys are active during the drag operation:

Esc The drag operation is canceled.
F1 A DM_DROPHELP message is posted to the target so that it can provide context help for the drag operation. The drag operation is canceled.

Once the drag is commenced by calling DrgDrag, neither the image under the pointer nor the objects that comprise the drag set can be modified without canceling the drag and restarting.

On return from DrgDrag, the caller must free the DRAGINFO structure using DrgFreeDraginfo.

If the dragged objects are not dropped, NULL is returned.

Related Functions

Prerequisite Functions

- DrgAllocDraginfo

Related Functions

- DrgFreeDraginfo
- DrgLazyDrag
- DrgQueryDragitemPtr
- DrgSetDragitem

Example Code

This example uses the DrgDrag function to drag a single object in response to the direct-manipulation button being pressed while the pointer is over a drag object. The example shows the initialization of the DRAGITEM, DRAGINFO, and DRAGIMAGE structures used by the DrgDrag function.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions  */
#define INCL_WININPUT   /* Window Input Functions               */
#include <os2.h>

PDRAGINFO pdinfo;       /* Pointer to DRAGINFO structure        */
HWND      hwnd;         /* Handle of calling (source) window    */
BOOL      flResult;     /* Result indicator                     */
DRAGITEM  ditem;        /* DRAGITEM structure                   */
DRAGIMAGE dimg;         /* DRAGIMAGE structure                  */
HBITMAP   hbm;          /* Bit-map handle                       */
HWND      hwndDrop;     /* Handle of drop (target) window       */

case WM_BEGINDRAG:

    /*********************************************************/
    /* Initialize the DRAGITEM structure                     */
    /*********************************************************/
    ditem.hwndItem = hwnd;       /* Conversation partner         */
    ditem.ulItemID = ID_ITEM;    /* Identifies item being dragged*/
    ditem.hstrType = DrgAddStrHandle(DRT_TEXT);   /* Text item   */
    ditem.hstrRMF = DrgAddStrHandle("<DRM_OS2FILE,DRF_TEXT>");
    ditem.hstrContainerName = DrgAddStrHandle("C:\\");
    ditem.hstrSourceName = DrgAddStrHandle("C:\\CONFIG.SYS");
    ditem.hstrTargetName = DrgAddStrHandle("C:\\OS2\\CONFIG.SYS");
    ditem.cxOffset = 0;          /* X-offset of the origin of    */
                                 /* the image from the pointer   */
                                 /* hotspot                      */
    ditem.cyOffset = 0;          /* Y-offset of the origin of    */
                                 /* the image from the pointer   */
                                 /* hotspot                      */
    ditem.fsControl = 0;         /* Source item control flags    */
                                 /* object is open               */
    ditem.fsSupportedOps = 0;

    /*********************************************************/
    /* Create the DRAGINFO structure                         */
    /*********************************************************/
    pdinfo = DrgAllocDraginfo(1);
    if (!pdinfo) return (FALSE); /* If allocation fails,         */
                                 /* return FALSE                 */

    /*********************************************************/
    /* Initialize the DRAGIMAGE structure                    */
    /*********************************************************/
    dimg.cb = sizeof(DRAGIMAGE); /* Size control block           */
    dimg.cptl       = 0;
    dimg.hImage = hbm;           /* Image handle passed to       */
                                 /* DrgDrag                      */
    dimg.sizlStretch.cx = 20L;   /* Size to stretch ico or bmp to*/
    dimg.sizlStretch.cy = 20L;
    dimg.fl = DRG_BITMAP |       /* Flags passed to DrgDrag      */
              DRG_STRETCH;       /* Stretch to size specified    */
                                 /* in sizlStretch               */
    dimg.cxOffset = 0;           /* Offset of the origin of      */
    dimg.cyOffset = 0;           /* the image from the pointer   */
                                 /* hotspot                      */
```

```
/*************************************************************/
/* Set the drag item                                         */
/*************************************************************/
flResult= DrgSetDragitem(pdinfo, &ditem, (ULONG)sizeof(ditem),
                         0);

/*************************************************************/
/* Perform the drag operation:                               */
/* - Give the user a visual cue by changing the pointer to a */
/*   bit map                                                 */
/* - Send DM_DRAGOVER messages to the target window (in this */
/*   case it is also the source)                             */
/* NOTE: DrgDrag will fail if another window in the same     */
/*       thread already has the capture.                     */
/*************************************************************/
hwndDrop = DrgDrag(hwnd,    /* Source of the drag            */
           pdinfo,          /* Pointer to DRAGINFO structure */
          (PDRAGIMAGE)&dimg, /* Drag image                   */
           1,               /* Size of the pdimg array       */
           VK_ENDDRAG,      /* Release of direct-manipulation */
                            /* button ends the drag          */
           NULL);           /* Reserved                      */
```

DrgDragFiles

This function begins a direct manipulation operation for one or more files.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgDragFiles (HWND Hwnd, PAPSZ pFiles, PAPSZ pTypes,
                   PAPSZ pTargets, ULONG cFiles, HPOINTER hptrDrag,
                   ULONG vkTerm, BOOL fSourceRender, ULONG ulReserved)
```

Parameters

Hwnd (HWND) – input
Handle of calling window.

pFiles (PAPSZ) – input
The names of the files to be dragged.

pTypes (PAPSZ) – input
The file types of the files to be dragged.

pTargets (PAPSZ) – input
Target file names.

cFiles (ULONG) – input
Number of files to be dragged.

hptrDrag (HPOINTER) – input
Icon to display during the drag.

vkTerm (ULONG) – input
Button that ends the drag.

Possible values are described in the following list:

VK_BUTTON1	Release of button 1 ends the drag.
VK_BUTTON2	Release of button 2 ends the drag.
VK_BUTTON3	Release of button 3 ends the drag.
VK_ENDDRAG	Release of the system-defined direct manipulation button ends the drag. This is the recommended value if the DrgDrag function call is invoked in response to a WM_BEGINDRAG message.

fSourceRender (BOOL) – input
Flag indicating whether the source must perform the move or copy.

TRUE	The caller will receive a DM_RENDERFILE message for each file.
FALSE	All file manipulation is performed by DrgDragFiles.

ulReserved (ULONG) – input
Reserved.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	The drag operation was initiated successfully.
FALSE	An error occurred.

Remarks

This function begins a direct manipulation operation for one or more files. DRAGINFO and DRAGITEM structures are allocated and initialized, and are then used as input to DrgDrag. All of the post-drag conversation required to complete the direct manipulation operation is handled by an object window created by this function.

The caller should set *fSourceRender* to TRUE if it must perform the file manipulation for any of these files. When *fSourceRender* is TRUE, the caller receives a DM_RENDERFILE message as the drag-object window receives a DM_RENDER message. The caller should move or copy the file after receiving the DM_RENDERFILE message. The caller should then send a DM_FILERENDERED message to the drag-object window, and the drag-object window should send a DM_RENDERCOMPLETE message to the target.

When *pTypes* is NULL, the .TYPE EA is interrogated to determine the type for each file in *pFiles*. When *pTypes* is not NULL, the size of the array is expected to be the same as the size of *pFiles*. When any individual pointer in the array is NULL, the .TYPE EA for the corresponding file is read. When .TYPE EA does not exist for any file for which it is needed, a type of DRT_UNKNOWN is used.

When *pTargets* is NULL, the target name for a file will be the same as the source file name with the path information removed. If *pTargets* is not NULL, the size of the array is expected to be the same as the size of *pFiles*. If any individual pointer in the array is NULL, the target name for the corresponding file will match the source name minus the path information.

The rendering mechanism and format for each file is:

```
<DRM_OS2FILE,DRF_UNKNOWN>.
```

When an error occurs during the move or copy, the caller is sent a DM_DRAGERROR message. The caller can take corrective action.

As the operation is complete for each file in the list, a DM_DRAGFILECOMPLETE message is sent to the caller of DrgDragFiles. The caller is thus notified that resources can be freed for a particular file.

This function returns TRUE if the drag operation was initiated successfully and FALSE if an error occurred.

Related Functions

- DrgAcceptDroppedFiles

Example Code

This example calls the DrgDragFiles function to begin direct manipulation for a single file object, using the same source and target name, and determining the file type based on the file's type EA.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#define INCL_WININPUT   /* Window Input Functions               */
#include <os2.h>

BOOL     fSuccess;      /* Indicate success or failure         */
HWND     Hwnd;          /* Handle of calling window            */
PSZ      pFiles[1];     /* The names of the files to be dragged */
PSZ      pTypes[1];     /* The file types of the files to be   */
                        /* dragged                             */
PSZ      pTargets[1];   /* The target file names               */
HPOINTER hptrDrag;      /* Icon to display during drag         */

pFiles[0] = "FILENAME.EXT";  /* Copy file name to string array */
pTargets[0] = NULL;     /* Use source name as target name      */
pTypes[0] = NULL;       /* Query type EA to determine file type */

fSuccess = DrgDragFiles(Hwnd, pFiles, pTypes, pTargets, 1,
                        hptrDrag, VK_BUTTON2, FALSE, 0L);
```

DrgFreeDraginfo

This function frees a DRAGINFO structure allocated by DrgAllocDraginfo.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgFreeDraginfo  (PDRAGINFO  pDraginfo)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_MEMORY_DEALLOCATION_ERR (0x1113)	An error occurred during memory management.
PMERR_SOURCE_SAME_AS_TARGET (0x1502)	The direct manipulation source and target process are the same.

Remarks

DrgFreeDraginfo fails with an error of PMERR_SOURCE_SAME_AS_TARGET if it is called by the process that called DrgDrag before DrgDrag returns. When a process is performing a drag operation between two of its own windows, this prevents the source window from freeing the DRAGINFO structure before the target window finishes processing.

Related Functions

Prerequisite Functions

- DrgAllocDraginfo

Related Functions

- DrgAccessDraginfo
- DrgDrag
- DrgPushDraginfo

Example Code

This example calls the DrgFreeDraginfo function to free an existing DRAGINFO structure allocated by the DrgAllocDraginfo function after a drag operation has completed.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL       fSuccess;    /* Indicate success or failure          */
PDRAGINFO  pdinfo;      /* Pointer to DRAGINFO structure        */
HWND       hwnd;        /* Handle of calling (source) window    */
DRAGIMAGE  dimg;        /* DRAGIMAGE structure                  */
HWND       hwndDrop;    /* Handle of drop (target) window       */

/*******************************************************************/
/* Perform the drag operation:                                     */
/* - Give the user a visual cue by changing the pointer to a       */
/*   bit map                                                       */
/* - Send DM_DRAGOVER messages to the target window (in this       */
/*   case it is also the source)                                   */
/*******************************************************************/
hwndDrop = DrgDrag(hwnd,       /* Source of the drag               */
           pdinfo,             /* Pointer to DRAGINFO structure    */
           (PDRAGIMAGE)&dimg,  /* Drag image                       */
           1,                  /* Size of the pdimg array          */
           VK_ENDDRAG,         /* Release of drag button           */
                               /* Terminates the drag              */
           NULL);              /* Reserved                         */

fSuccess = DrgFreeDraginfo(&pdinfo);
```

DrgFreeDragtransfer

This function frees the storage associated with a DRAGTRANSFER structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgFreeDragtransfer (PDRAGTRANSFER pdxfer)
```

Parameters

pdxfer (PDRAGTRANSFER) – input
Pointer to the DRAGTRANSFER structures to be freed.

Returns

rc (BOOL) – returns
Return code.

TRUE	The structure was freed successfully.
FALSE	The deallocation failed.

Possible returns from WinGetLastError

PMERR_MEMORY_DEALLOCATION_ERR (0x1113)	An error occurred during memory management.

Remarks

This function frees the DRAGTRANSFER structures allocated by calls to DrgAllocDragtransfer. When all of the DRAGTRANSFER structures have been freed, the memory block containing the DRAGTRANSFER array is deallocated.

Related Functions

- DrgAllocDragtransfer

Example Code

This example calls the DrgFreeDragtransfer function to free an existing DRAGTRANSFER structure allocated by the DrgAllocDragtransfer function.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL          fSuccess; /* Indicate success or failure          */
DRAGTRANSFER  dxfer;    /* Pointer to DRAGTRANSFER structure    */

fSuccess = DrgFreeDragtransfer(&dxfer);
```

DrgGetPS

This function gets a presentation space that is used to provide target feedback to the user during a drag operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
HPS DrgGetPS  (HWND Hwnd)
```

Parameters

Hwnd (HWND) – input
Handle of the window for which presentation space is required.

Returns

Hps (HPS) – returns
Presentation-space handle used for drawing in the window.

NULLHANDLE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_NOT_DRAGGING (0x1f00)	A drag operation is not in progress at this time.

Remarks

This function returns a handle to a presentation space that can be used for drawing while a direct manipulation operation is in progress.

DrgGetPS is called only during a direct manipulation operation. This function is called only after a DM_DRAGOVER, DM_DRAGLEAVE, or DM_DROP message has been received.

In order to draw target emphasis, an application must use DrgGetPS and DrgReleasePS to unlock its window.

The presentation space created with DrgGetPS must be freed with DrgReleasePS.

Related Functions

- DrgReleasePS

Example Code

This example uses the DrgGetPS function to get a presentation space handle which is used during drag operations such as loading a drag bit map. When finished with the presentation space, release it with the DrgReleasePS function.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

HPS   hps;              /* Presentation space handle           */
HWND  hwnd;             /* Handle of the window for which      */
                        /* presentation space is required      */

case DM_DRAGOVER:
    hps = DrgGetPS(hwnd);

DrawTargetEmphasis(hps, hwnd);
DrgReleasePS(hps);
```

DrgLazyDrag

This function is called when a direct-manipulation button is pressed wile the lazy drag augmentation key is held to initiate a pickup and drop (lazy drag) operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgLazyDrag  (HWND hwndSource, PDRAGINFO pDraginfo,
                   PDRAGIMAGE pdimg, ULONG cdimg, PVOID Reserved)
```

Parameters

hwndSource (HWND) – input
Handle of the source window that is calling this function.

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure which contains information about the objects being dragged.

pdimg (PDRAGIMAGE) – input
Pointer to an array of DRAGIMAGE structures.

These structures describe the images that are to be drawn under the direct-manipulation pointer during the drag.

cdimg (ULONG) – input
Number of DRAGIMAGE structures in the *pdimg* array.

Reserved (PVOID) – input
Reserved value, must be 0.

Returns

rc (BOOL) – returns
Success indicator.

Possible values are described in the following list:

TRUE A lazy drag operation was successfully started.
FALSE An error occurred while initiating a lazy drag operation.

Remarks

Before initiating a lazy drag operation by calling DrgLazyDrag, the application must allocate the DRAGINFO structure using DrgAllocDraginfo.

DrgLazyDrag is called when the direct-manipulation button is pressed while holding down the lazy drag augmentation key, currently the ALT key. As the pointer moves over a potential target, a DM_DRAGOVER message is sent to the target window. When the pointer moves from one target window to another, a DM_DRAGLEAVE message is sent to the former target.

If the pointer is over a valid window target when the direct-manipulation button is pressed and the lazy drag "end drag" augmentation key (Ctrl+Shift to link, Ctrl to copy and Shift to move) is pressed, a DM_DROP message is sent to the target window. The source window posts a DM_DROPNOTIFY message, informing the target window that a drop has occurred.

param2 parameter of DM_DROPNOTIFY message contains the handle of the target window that the drag set was dropped on. If *param2* is zero, the drag set was not dropped; in other words, the drag operation was canceled. The source window must check *pDraginfo* to see if the target window is different from the source. If the source and target are different, the source window must free DRAGINFO upon receipt of the DM_DROPNOTIFY message. If the source and target window handles are the same, the target must free the DRAGINFO after completing the post-drop conversation.

The caller can define a default operation for the objects represented by the DRAGINFO structure by modifying its *usOperate* field. If *usOperate* is modified, the new value is sent to the target as the operation when a NO_DEFAULT operation would normally be sent. The caller must not modify any other part of the DRAGINFO structure. The DRAGITEM structures associated with the DRAGINFO structure must only be altered with DrgSetDragitem or by using a pointer obtained from DrgQueryDragitemPtr.

A window receives a WM_PICKUP message when the direct-manipulation button is pressed holding down the lazy drag augmentation key. In response to this message, an application is responsible for:

- Obtaining a DRAGINFO structure using DrgAllocDraginfo
- Initializing the DRAGITEM structures using DrgSetDragitem.

Objects are added to the drag set whenever a WM_PICKUP message is received. The first time that message is received, the application must call DrgLazyDrag to begin the lazy drag operation. Each subsequent WM_PICKUP message that is received during the course of a lazy drag operation indicates that objects are to be added to the drag set. If objects are currently being dragged, the application must reallocate the DRAGINFO structure using DrgReallocDragInfo. DrgReallocDragInfo frees the current DRAGINFO and allocates a new one. The lazy drag operation can then continue by making another call to DrgLazyDrag.

Objects can also be removed from the drag set during the course of a lazy drag operation. It is the application's responsibility to define what action initiates such a "putback" operation; for example, if a "pickup" operation executed while the pointer is not over a valid object. It is the application's responsibility to decide if a putback operation may apply to individual objects within the drag set or the drag set as a whole. If the putback operation is applied to the drag set as a whole, the result is the same as canceling the drag. If the put-back operation is applied to individual objects within the drag set, the application must free the DRAGINFO

and reallocate it to represent the new state of the drag set. If the drag operation is to continue, the application must make another call to DrgLazyDrag.

DrgLazyDrag returns as soon as it completes initialization for the drag. The pointing device remains active during a lazy drag and can be used in the same manner as if no drag operation were in progress. As soon as DrgLazyDrag returns, the application can free its DRAGIMAGE array.

Note: Since the lazy drag operation is non-modal, the mouse pointer may by used as if no drag operation were in progress.

Once a standard drag operation is commenced by calling DrgDrag, the objects that comprise the drag set can not be modified without canceling the drag and restarting. The drag set can be modified during the course of the drag operation. When the drag set is modified, the application must reallocate the DRAGINFO structure and make another call to DrgLazyDrag.

Note: The mouse changes to an arrow with an attache case attached to the lower right-hand corner when the lazy drag is in progress, rather than a stack of images shown during a standard drag.

The *pdimg* and *cdimg* parameters were added for compatibility with DrgDrag only.

Related Functions

Prerequisite Functions

- DrgAllocDraginfo

Related Functions

- DrgDrag
- DrgFreeDraginfo
- DrgQueryDragitemPtr
- DrgSetDragitem

Example Code

This example shows the proper sequence for initiating a lazy drag operation after the user has selected an object and pressed the direct manipulation button while holding down the lazy drag augmentation key (ALT). The window receives a WM_PICKUP message indicating that a lazy drag operation is to begin.

```
#define  INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO  pdinfo;        /* Pointer to a DRAGINFO structure */
HWND       hwndSource;    /* Handle of the Source window */
DRAGITEM   ditem;         /* DRAGITEM structure */
PDRAGIMAGE pdimg;         /* Pointer to DRAGIMAGE structure */
HBITMAP    hbm;           /* Bit-map handle passed to DrgLazyDrag */
  .
  .
  .
case WM_PICKUP:

    /*********************************************************************/
    /*  Initialize the DRAGITEM structure                                */
    /*********************************************************************/
    ditem.hwndItem=hwndSource;  /* Handle of the source window */
    ditem.ulItemID=ID_ITEM;     /* App defined id of item */
    ditem.hstrType=DrgAddStrHandle("DRT_TEXT"); /* Text item  */
    ditem.hstrRMF=DrgAddStrHandle("<DRM_OS2FILE,DRF_TEXT>");
    ditem.hstrContainerName=DrgAddStrHandle("C:\\");
    ditem.hstrSourceName=DrgAddStrHandle("C:\\CONFIG.SYS");
    ditem.hstrTargetName=DrgAddStrHandle("C:\\OS2\\CONFIG.SYS");
    ditem.cxOffset=0;  /* X-offset of the origin of the image from the */
                       /*   pointer hotspot */
    ditem.cyOffset=0;  /* Y-offset of the origin of the image from the */
                       /*   pointer hotspot */
    ditem.fsControl=0; /* Source item control flags */
    ditem.fsSupportedOps=0;

    /*********************************************************************/
    /*  Create the DRAGINFO structure                                    */
    /*********************************************************************/
    pdinfo=DrgAllocDraginfo(1);
    if(!pdinfo) return FALSE;  /* Return FALSE if initialization fails */

    /*********************************************************************/
    /*  Initialize the DRAGIMAGE structure                               */
    /*********************************************************************/
    pdimg=AllocMem(sizeof(DRAGIMAGE));

    pdimg->cb=sizeof(DRAGIMAGE);       /* Size of the dragimage structure */
    pdimg->cptl=0;                     /* Image is not a polygon */
    pdimg->hImage=hbm;                 /* Handle of image to display */
    pdimg->sizlStretch.cx=20L          /* Size to stretch icon or bit map */
    pdimg->fl=DRG_BITMAP|DRG_STRETCH; /* Flags passed to DrgLazyDrag */
    pdimg->cxOffset=0;                 /* Offset of the origin of image */
    pdimg->cyOffset=0;                 /*   from the pointer hotspot */

    /*********************************************************************/
    /*  Set the DRAGITEM                                                 */
    /*********************************************************************/
    DrgSetDragitem(pdinfo, &ditem, (ULONG)sizeof(ditem, 0);
```

```
/*****************************************************************/
/*  Begin the Lazy Drag operation                                */
/*****************************************************************/
if (DrgLazyDrag (hwndSource,      /* Source of the drag */
                 pdinfo,          /* Pointer to the DRAGINFO */
                 pdimg,           /* DRAGIMAGE array */
                 1,               /* Size of the DRAGIMAGE array */
                 NULL)) {         /* Reserved */
   FreeMem (pdimg);               /* Free DRAGIMAGE if successful */
}
```

DrgLazyDrop

This function is called to invoke a lazy drop operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgLazyDrop  (HWND hwndTarget, ULONG ulsOperation,
                   PPOINTL pptlDrop)
```

Parameters

hwndTarget (HWND) – input
Handle of the target window receiving the drop.

ulsOperation (ULONG) – input
Drop operation code.

Possible values are shown in the following list:

DO_DEFAULT	Default operation.
DO_COPY	Operation is a copy.
DO_MOVE	Operation is a move.
DO_LINK	Operation is a link.

pptlDrop (PPOINTL) – input
Pointer to the drop location in desktop coordinates.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Objects are successfully dropped.
FALSE	An error occurred.

Remarks

This function can be used to implement a "drop" choice from a menu.

Example Code

This example uses DrgLazyDrop to perform a drop to a target window, using the copy operation.

```
#define  INCL_WINSTDDRAG
#include <os2.h>

HWND       hwndTarget;    /* Handle of the Target window */
ULONG      ulDropOp;      /* Operation to be performed on the drop */
POINTL     ptlDrop;       /* Drop location */
BOOL       bSuccess;      /* Return code from API */

ptlDrop.x=5;
ptlDrop.y=10;

usDropOp=DO_COPY;
bSuccess=DrgLazyDrop(hwndTarget, ulDropOp, &ptlDrop);
```

DrgPostTransferMsg

This function posts a message to the other application involved in the direct manipulation operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgPostTransferMsg  (HWND hwndTo, ULONG ulMsgid,
                          PDRAGTRANSFER pdxfer, ULONG fs,
                          ULONG ulReserved, BOOL fRetry)
```

Parameters

hwndTo (HWND) – input
Window handle to which the message is to be posted.

Target *hwndItem* in the DRAGITEM structure.
Source *hwndClient* in the DRAGTRANSFER structure.

ulMsgid (ULONG) – input
Identifier of the message to be posted.

DM_RENDERCOMPLETE is the only valid message.

pdxfer (PDRAGTRANSFER) – input
Pointer to the DRAGTRANSFER structure.

fs (ULONG) – input
Flags to be passed in the *param2* parameter of the message identified by *ulMsgid*.

ulReserved (ULONG) – input
Reserved value, must be 0.

fRetry (BOOL) – input
Retry indicator.

TRUE If the destination queue is full, the message posting is retried at 1-second intervals until the message is posted successfully.

In this case, DrgPostTransferMsg dispatches any messages in the queue by calling WinPeekMsg and WinDispatchMsg in a loop. The application can receive messages sent by other applications while it is trying to post drag transfer messages.

FALSE The call returns FALSE without retrying.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

The *fsReply* field in the DRAGTRANSFER structure is set to 0 before the message is posted. If the posting fails for any reason, FALSE is returned.

Related Functions

- DrgSendTransferMsg

Example Code

This example calls the DrgPostTransferMsg function to respond to a DM_RENDER message from the target. The response consists of a DM_RENDERCOMPLETE message, plus a flag indicating whether the render was successful (DMFL_RENDEROK) or not (DMFL_RENDERFAIL).

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions  */
#include <os2.h>

MPARAM        mp1;      /* Message parameter 1                   */
BOOL          fSuccess; /* Indicate success or failure           */
BOOL          Rendered; /* Success of render operation           */
PDRAGTRANSFER pdxfer;   /* Pointer to DRAGTRANSFER structure     */

case DM_RENDER:
   pdxfer = (PDRAGTRANSFER)PVOIDFROMMP(mp1); /* Get DRAGTRANSFER */
                                             /* structure        */

    /*********************************************************/
    /* Attempt to render file                                */
    /*********************************************************/

    if (Rendered)
       {
       fSuccess = DrgPostTransferMsg(pdxfer->pditem,
                                     DM_RENDERCOMPLETE,
                                     pdxfer,
                                     DMFL_RENDEROK,
                                     0,FALSE);
       return (MRESULT)TRUE;
       }
    else
       {
       fSuccess = DrgPostTransferMsg(pdxfer->pditem,
                                     DM_RENDERCOMPLETE,
                                     pdxfer,
                                     DMFL_RENDERFAIL,
                                     0,FALSE);
       return (MRESULT)FALSE;
       }
```

DrgPushDraginfo

This function gives a process access to a DRAGINFO structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgPushDraginfo  (PDRAGINFO pDraginfo, HWND hwndDest)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure.

hwndDest (HWND) – input
Handle of the window whose process is to be given access to a DRAGINFO structure.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_ACCESS_DENIED (0x150D)	The memory block was not allocated properly.
PMERR_INSUFFICIENT_MEMORY (0x203E)	The operation terminated through insufficient memory.

Remarks

The receiving process is responsible for:

1. Deleting the string handles in the DRAGINFO structure with DrgDeleteDraginfoStrHandles
2. Freeing the DRAGINFO structure using DrgFreeDraginfo.

Related Functions

- DrgAccessDraginfo
- DrgAllocDraginfo
- DrgDrag
- DrgFreeDraginfo

Example Code

This example calls the DrgPushDraginfo function to grant access to a DRAGINFO structure to the process owning the specified window handle. The DRAGINFO structure was previously allocated using the DrgAllocDraginfo function.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL      fSuccess;     /* Indicate success or failure          */
DRAGINFO  Draginfo;     /* Pointer to DRAGINFO structure        */
HWND      hwndDest;     /* Handle of window whose process will  */
                        /* will be given access to the DRAGINFO */
                        /* structure                            */

fSuccess = DrgPushDraginfo(&Draginfo,hwndDest);
```

DrgQueryDraginfoPtr

This function obtains a pointer to the current DRAGINFO structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
PDRAGINFO DrgQueryDraginfoPtr  (PDRAGINFO pRsvd)
```

Parameters

pRsvd (PDRAGINFO) – input
Reserved value, must be NULL.

Returns

pDraginfo (PDRAGINFO) – returns
Pointer to the current DRAGINFO structure.

A return value of NULL indicates that a DRAGINFO has not been allocated.

Remarks

The returned DRAGINFO structure could have been allocated for use in either a standard or lazy drag operation. DrgQueryDragStatus can be used to determine which type of drag is active.

Related Functions

- DrgQueryDraginfoPtrFromDragitem
- DrgQueryDraginfoPtrFromHwnd

Example Code

This example uses DrgQueryDraginfoPtr to obtain a pointer to the current DRAGINFO structure.

```
#define  INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO  pdinfo;       /* Pointer to a DRAGINFO structure */

pdinfo = DrgQueryDraginfoPtr(NULL);
if (pdinfo)
{
  .
  .
  .
}
```

DrgQueryDraginfoPtrFromDragitem

This function is called to obtain a pointer to the DRAGINFO structure associated with a given DRAGITEM structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
PDRAGINFO DrgQueryDraginfoPtrFromDragitem (PDRAGITEM pDragitem)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to a DRAGITEM structure whose corresponding DRAGINFO is to be returned.

Returns

pDraginfo (PDRAGINFO) – returns
Pointer to the DRAGINFO structure for the specified *pDragitem*.

A return value of NULL indicates that a DRAGITEM structure was not found.

Example Code

This example uses DrgQueryDraginfoPtrFromDragitem to obtain the pointer to the DRAGINFO structure associated with a given DRAGITEM structure. The example assumes the application already has a pointer to one of the DRAGITEM structures.

```
#define  INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO  pdinfo;        /* Pointer to a DRAGINFO structure */
PDRAGITEM  pdragitem;     /* Pointer to a DRAGITEM structure */

pdinfo=DrgQueryDraginfoPtrFromDragitem(pdragitem);
if (pdinfo) {
   .
   .
   .
}
```

DrgQueryDraginfoPtrFromHwnd

This function determines whether a particular window has allocated a DRAGINFO structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
PDRAGINFO DrgQueryDraginfoPtrFromHwnd  (HWND hwndSource)
```

Parameters

hwndSource (HWND) – input
Handle of the window whose associated DRAGINFO pointer is to be returned.

Returns

pDraginfo (PDRAGINFO) – returns
Pointer to the DRAGINFO structure allocated by the window specified by *hwndSource*.

If the return value is NULL, the DRAGINFO structure has not been allocated.

Remarks

The application can gain access to the structure DRAGINFO by calling DrgAccessDraginfo.

Example Code

This example uses DrgQueryDraginfoPtrFromHwnd to obtain a pointer to the DRAGINFO structure allocated by the given window.

```
#define  INCL_WINSTDDRAG
#include <os2.h>

HWND       hwndWindow;   /* Window handle */
PDRAGINFO  pinfo;        /* Pointer to a DRAGINFO structure */

pdinfo=DrgQueryDraginfoPtrFromHwnd(hwndWindow);
If (pdinfo) {
  .
  .
  .
}
```

DrgQueryDragitem

This function returns a DRAGITEM structure used in the direct manipulation operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgQueryDragitem  (PDRAGINFO pDraginfo, ULONG cbBuffer,
                        PDRAGITEM pDragitem, ULONG iItem)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure from which the DRAGITEM structure is obtained.

cbBuffer (ULONG) – input
Maximum number of bytes to copy to the buffer.

pDragitem (PDRAGITEM) – output
Pointer to the buffer into which the DRAGITEM structure is copied.

iItem (ULONG) – input
Zero-based index of the DRAGITEM to be returned.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

This function returns the DRAGITEM structure identified by *iItem*.

Related Functions

- DrgQueryDragitemPtr
- DrgSetDragitem

Example Code

This example calls the DrgQueryDragitem function to return the entirety of the first DRAGITEM structure in the given DRAGINFO structure, after which it obtains the source window handle.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

BOOL      fSuccess;     /* Indicate success or failure          */
DRAGINFO  Draginfo;     /* DRAGINFO structure from which the    */
                        /* DRAGITEM structure is obtained       */
ULONG     cbBuffer;     /* Maximum number of bytes to copy      */
DRAGITEM  Dragitem;     /* Buffer into which the DRAGITEM       */
                        /* structure is copied                  */
ULONG     iItem;        /* Zero-based index of the DRAGITEM     */
                        /* to be returned                       */
HWND      hwndSource;   /* Source window handle for the drag    */

cbBuffer = sizeof(DRAGITEM); /* Copy entire DRAGITEM structure  */
iItem = 0;                   /* Return first DRAGITEM           */

fSuccess = DrgQueryDragitem(&Draginfo,cbBuffer,&Dragitem,iItem);

hwndSource = Dragitem.hwndItem;  /* Obtain source window handle */
```

DrgQueryDragitemCount

This function returns the number of objects being dragged during the current direct manipulation operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
ULONG DrgQueryDragitemCount (PDRAGINFO pDraginfo)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure for which number of dragged objects is requested.

Returns

cDitem (ULONG) – returns
Number of objects being dragged.

Example Code

This example calls the DrgQueryDragitemCount function to return the number of DRAGITEM structures in the corresponding DRAGINFO structure, which maps to the number of objects being dragged.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions */
#include <os2.h>

ULONG     cDitem;       /* Number of objects being dragged      */
DRAGINFO  Draginfo;     /* DRAGINFO structure queried for the   */
                        /* number of drag objects               */

cDitem = DrgQueryDragitemCount(&Draginfo);
```

DrgQueryDragitemPtr

This function returns a pointer to the DRAGITEM structure used in the direct manipulation operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
PDRAGITEM DrgQueryDragitemPtr  (PDRAGINFO pDraginfo, ULONG ulIndex)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure from which the DRAGITEM structure is obtained.

ulIndex (ULONG) – input
Zero-based index of the DRAGITEM structure for which the pointer is to be returned.

Returns

Dragitem (PDRAGITEM) – returns
Pointer to the DRAGITEM structure.

Remarks

This function returns a pointer to *ulItemID* in the DRAGITEM structure used in the direct manipulation operation.

Related Functions

- DrgQueryDragitem

Example Code

This example calls the DrgQueryDragitemPtr function to return a pointer to first DRAGITEM structure in the given DRAGINFO structure, after which it obtains the source window handle.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions  */
#include <os2.h>

PDRAGITEM  pDragitem;   /* DRAGITEM structure pointer            */
DRAGINFO   Draginfo;    /* DRAGINFO structure from which the     */
                        /* DRAGITEM structure is obtained        */
ULONG      ulIndex;     /* Zero-based index of the DRAGITEM      */
                        /* structure pointer to be returned      */
HWND       hwndSource;  /* Source window handle for the drag     */

USHORT     usn = 0;     /* Return pointer to first DRAGITEM      */

pDragitem = DrgQueryDragitemPtr(&Draginfo,usn);

hwndSource = pDragitem->hwndItem; /* Obtain source window handle */
```

DrgQueryDragStatus

This function determines the status of the current drag operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
ULONG DrgQueryDragStatus ()
```

Parameters

None.

Returns

rc (ULONG) – returns

Flag indicating the current drag status.

Possible values are shown in the following list:

0	A drag operation is not currently in progress.
DGS_DRAGINPROGRESS	A standard drag operation is in progress.
DGS_LAZYDRAGINPROGRESS	A lazy drag operation is in progress.

Example Code

This example uses DrgQueryDragStatus to determine whether a lazy drag operation is currently in progress.

```
#define  INCL_WINSTDDRAG
#inlcude <os2.h>

if (DrgQueryDragStatus() & DGS_LAZYDRAGINPROGRESS)
{
   .
   . /* Lazy drag is in progress */
   .
{
```

DrgQueryNativeRMF

This function obtains the ordered pair that represents the native rendering mechanism and format of the dragged object.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgQueryNativeRMF  (PDRAGITEM pDragitem, ULONG cbBuffer,
                         PCHAR ppBuffer)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure.

Pointer to the DRAGITEM structure whose native rendering mechanism and format are to be obtained.

cbBuffer (ULONG) – input
Maximum number of bytes to copy to the buffer.

ppBuffer (PCHAR) – output
Pointer to the buffer in which the null-terminated string is to be returned.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range –32 768 to +32 767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

Remarks

If the rendering mechanism and format string for the object are NULL, FALSE is returned. If TRUE is returned, the format of the string is:

```
<mechanism,format>
```

The native rendering mechanism and format are the first ordered pair, or the first ordered pair produced by a cross product, in the string associated with *hstrRMF* in the DRAGITEM structure.

DrgQueryNativeRMFLen can be used to determine the size of the buffer required to hold the string returned by this function.

Related Functions

Prerequisite Functions

- DrgQueryNativeRMFLen

Related Functions

- DrgVerifyNativeRMF

Example Code

This example shows how to obtain the window handle of the source of a drag item.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#define INCL_DOSMEMMGR   /* Memory Management Functions for      */
                         /* DosSubAlloc                          */
#include <OS2.H>

DRAGITEM ditem;
PVOID    pMem;
PSZ      pszBuffer;
ULONG    cb;
BOOL     rc, fResult;

cb = DrgQueryNativeRMFLen(&ditem) + 1;

rc = DosSubAlloc(pMem, (PVOID *) pszBuffer, cb);

if (!rc)

{
   fResult = DrgQueryNativeRMF(&ditem, cb, pszBuffer);
}
```

DrgQueryNativeRMFLen

This function obtains the length of the string representing the native rendering mechanism and format of the dragged object.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
ULONG DrgQueryNativeRMFLen  (PDRAGITEM pDragitem)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose native rendering mechanism and format string length are to be obtained.

Returns

ulLength (ULONG) – returns
String length of the ordered pair.

0	Error occurred.
Other	String length of the ordered pair, excluding the null-terminating byte.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32768 to +32767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

Remarks

This function is used to determine the size of the buffer that contains the string representing the native rendering mechanism and format of the dragged object.

If the input string handle is NULLHANDLE or not valid, a length of 0 is returned.

Related Functions

- DrgQueryNativeRMF

Example Code

This example shows how to obtain the window handle of the source of a drag item.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#define INCL_DOSMEMMGR   /* Memory Management Functions for      */
                         /* DosSubAlloc                          */
#include <OS2.H>

DRAGITEM ditem;
PVOID    pMem;
PSZ      pszBuffer;
ULONG    cb;
BOOL     rc, fResult;

cb = DrgQueryNativeRMFLen(&ditem) + 1;

rc = DosSubAlloc(pMem, (PVOID *) pszBuffer, cb);

if (!rc)

{
   fResult = DrgQueryNativeRMF(&ditem, cb, pszBuffer);
}
```

DrgQueryStrName

This function gets the contents of a string associated with a string handle.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
ULONG DrgQueryStrName  (HSTR Hstr, ULONG cbBuflen, PSZ pBuffer)
```

Parameters

Hstr (HSTR) – input
The handle must have been created with DrgAddStrHandle.

cbBuflen (ULONG) – input
Maximum number of bytes to copy into *pBuffer*.

Must be greater than 0. Otherwise, an error is returned.

pBuffer (PSZ) – output
Buffer where the null-terminated string is returned.

Returns

ulLength (ULONG) – returns
Number of bytes written to *pBuffer*.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range –32768 to +32767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

Remarks

This function should be called whenever the contents of a string referenced by a drag string handle are required. If the input string handle is NULLHANDLE or not valid, a null string is returned.

Related Functions

Prerequisite Functions

- DrgQueryStrNameLen

Related Functions

- DrgAddStrHandle

Example Code

This example shows how to obtain the contents of a string given that the string handle is known. The string handle must have been originally created with the DrgAddStrHandle function.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#define INCL_DOSMEMMGR   /* Memory Management Functions for      */
                         /* DosAllocMem                          */
#include <OS2.H>

HSTR   hstr;             /* Handle to a string.  The handle must */
                         /* have been created with               */
                         /* DrgAddStrHandle.                     */
PSZ    pBuffer;          /* Buffer where the null-terminated     */
                         /* string is returned                   */
ULONG  ulStrlen;         /* String length                        */
ULONG  ulBytesRead;      /* Number of bytes read                 */
ULONG  rc;               /* Return code                          */

ulStrlen = DrgQueryStrNameLen(hstr) + 1;

rc = DosAllocMem((PVOID *) pBuffer,
                 (LONG)ulStrlen,
                 fPERM |
                 PAG_COMMIT);

/*****************************************************************/
/* The ulBytesRead parameter contains the number of bytes        */
/* actually written to the memory pointed to by pBuffer          */
/*****************************************************************/
ulBytesRead = DrgQueryStrName(hstr,
                    ulStrlen,     /* Number of bytes to copy   */
                    pBuffer);
```

DrgQueryStrNameLen

This function gets the length of a string associated with a string handle.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
ULONG DrgQueryStrNameLen  (HSTR Hstr)
```

Parameters

Hstr (HSTR) – input
String handle.

The handle must be created with DrgAddStrHandle.

Returns

cLength (ULONG) – returns
Length of the string associated with *Hstr*.

0	The string handle is NULLHANDLE or is not valid.
Other	The length of the string associated with the string handle, excluding the null terminating byte.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range –32 768 to +32 767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

Remarks

This function should be called before calling the DrgQueryStrName function. It is used to determine and allocate the buffer size for the string associated with the string handle. If the input string handle is NULLHANDLE or not valid, a length of 0 is returned.

Related Functions

- DrgQueryStrName

Example Code

This example shows how to obtain the length of a string given that the string handle is known. The string handle must have been originally created with the DrgAddStrHandle function.

```
#define INCL_WINSTDDRAG /* Direct Manipulation (Drag) Functions  */
#define INCL_DOSMEMMGR  /* Memory Management Functions for       */
                        /* DosAllocMem                          */
#include <OS2.H>

HSTR   hstr;            /* Handle to a string.  The handle must  */
                        /* have been created with                */
                        /* DrgAddStrHandle.                      */
PSZ    pBuffer;         /* Buffer where the null-terminated      */
                        /* string is returned                    */
ULONG  ulStrlen;        /* String length                         */
ULONG  ulBytesRead;     /* Number of bytes read                  */
ULONG  rc;              /* Return code                           */

ulStrlen = DrgQueryStrNameLen(hstr) + 1;

rc = DosAllocMem((PVOID *) pBuffer,
                 (LONG)ulStrlen,
                  fPERM |
                  PAG_COMMIT);

/*********************************************************************/
/* The ulBytesRead parameter contains the number of bytes            */
/* actually written to the memory pointed to by pBuffer              */
/*********************************************************************/
ulBytesRead = DrgQueryStrName(hstr,
                      ulStrlen,     /* Number of bytes to copy   */
                      pBuffer);
```

DrgQueryTrueType

This function obtains the true type of a dragged object.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgQueryTrueType  (PDRAGITEM pDragitem, ULONG cbBuflen,
                        PSZ pBuffer)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose type is to be obtained.

cbBuflen (ULONG) – input
Maximum number of bytes to copy to *pBuffer*. Must be > 0.

pBuffer (PSZ) – output
Buffer in which the null-terminated string is to be returned.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

Remarks

The true type of an object is the first type in the string referenced by *hstrType* in the DRAGITEM structure.

This function can be called after calling the DrgQueryTrueTypeLen function. If the type string for the object is NULLHANDLE, FALSE is returned.

Related Functions

Prerequisite Functions

- DrgQueryTrueTypeLen

Related Functions

- DrgVerifyTrueType

Example Code

This example shows how to obtain the true type of an object.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#include <OS2.H>

BOOL     fSuccess;       /* Return value                        */
DRAGITEM Dragitem;       /* DRAGITEM structure whose true type  */
                         /* is to be obtained                   */

char szBuffer[32];       /* Buffer in which the null-terminated */
                         /* string is to be returned            */

fSuccess = DrgQueryTrueType(&Dragitem,
                            sizeof(szBuffer),
                            szBuffer);
```

DrgQueryTrueTypeLen

This function obtains the length of the string that represents the true type of a dragged object.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
ULONG DrgQueryTrueTypeLen  (PDRAGITEM pDragitem)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose type length is to be obtained.

Returns

ulLength (ULONG) – returns
String length of the first element of the character string associated with *hstrType*.

0	Error occurred.
Other	The length of the first element of the character string associated with *hstrType*, excluding the null-terminating byte.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

Remarks

This function can be used to determine the buffer size to allocate for the string representing the true type of a dragged object. The true type of an object is the first type in the type string referenced by *hstrType* in the DRAGITEM structure.

This function can be called before calling the DrgQueryTrueType function.

If the input string handle is NULLHANDLE or not valid, a length of 0 is returned.

Related Functions

- DrgQueryTrueType

Example Code

This example shows how to obtain the length of the true type string with the DrgQueryTrueTypeLen function.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#define INCL_DOSMEMMGR   /* Memory Management Functions for      */
                         /* DosAllocMem                          */
#include <OS2.H>

PSZ      pszBuffer;      /* Buffer in which the DRAGITEM         */
                         /* structure is stored                  */
BOOL     fSuccess;       /* Return value                         */
DRAGITEM Dragitem;       /* DRAGITEM structure whose true type   */
                         /* length is to be obtained             */
ULONG    rc;             /* Return code                          */
ULONG    ulLength;       /* String length of dragged object      */

ulLength = DrgQueryTrueTypeLen(&Dragitem) + 1;

rc = DosAllocMem((PVOID *) pszBuffer, ulLength, fPERM);

fSuccess = DrgQueryTrueType(&Dragitem, ulLength, pszBuffer);
```

DrgReallocDragInfo

This function releases the current DRAGINFO structure and reallocates a new one.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
PDRAGINFO DrgReallocDragInfo (PDRAGINFO pdinfoOld, ULONG cditem)
```

Parameters

pdinfoOld (PDRAGINFO) – input
Pointer to the current DRAGINFO structure.

cditem (ULONG) – input
Number of DRAGITEM structures to be allocated.

Returns

pdinfoCurrent (PDRAGINFO) – returns
Pointer to a newly allocated DRAGINFO structure.

Remarks

It is necessary to call DrgReallocDragInfo anytime objects are added or deleted from the current lazy drag set. This function unconditionally frees the old DRAGINFO structure, reallocates a new DRAGINFO structure, and returns a pointer to the new structure.

Note: This function does not check if the source and target window handles are different; it unconditionally frees the DRAGINFO structure passed to it.

Related Functions

- DrgAllocDraginfo

Example Code

This example uses DrgReallocDragInfo to reallocate the DRAGINFO structure when an object is picked up or added to the lazy drag set. It checks whether a lazy drag operation is already in progress, and if so, adds one to the number of objects currently being dragged and calls DrgReallocDragInfo to obtain a new DRAGINFO structure with the required number of DRAGITEM structures.

```
#define  INCL_WINSTDDRAG
#include <os2.h>

PDRAGINFO  pdinfoCurrent;   /* Pointer to the current DRAGINFO     */
PDRAGINFO  pdinfoOld;       /* Pointer to the DRAGINFO to be freed */
ULONG      cditem;          /* Number of DRAGITEMS                 */
  .
  .
  .
case WM_PICKUP:

   if (DrgQueryDragStatus() & DGS_LAZYDRAGINPROGRESS)
   {
      /* Get a pinter to the current DRAGINFO structure */
      pdinfoOld=DrgQueryDraginfoPtr(NULL);

      /* Add space for one more DRAGITEM */
      cditem=pdinfoOld->cditem+1;

      /* Reallocate the DRAGINFO */
      pdinfoCurrent=DrgReallocDraginfo(pdinfoOld, cditem);
      if(pdinfoCurrent)
      {
         /* Continue the lazy drag operation */
         DrgLazyDrag( ... )
      }
   }
```

DrgReleasePS

This function releases a presentation space obtained by using the DrgGetPS function.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgReleasePS  (HPS  Hps)
```

Parameters

Hps (HPS) – input
Handle of the presentation space to release.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INV_HPS (0x207F)	An invalid presentation-space handle was specified.
PMERR_NOT_DRAGGING (0x1f00)	A drag operation is not in progress at this time.

Remarks

Only presentation spaces created with DrgGetPS can be released using this function.

The presentation-space handle should not be used after this function.

Related Functions

Prerequisite Functions

- DrgGetPS

Example Code

In this example the presentation space handle is retrieved, a bit map is loaded, and the presentation space is released with the DrgReleasePS function.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#include <os2.h>
#define ID_BITMAP 255
HPS    hps;
HWND hwnd;

case DM_DRAGOVER:
    hps = DrgGetPS(hwnd);

    DrawTargetEmphasis(hps, hwnd);
    DrgReleasePS(hps);
```

DrgSendTransferMsg

This function sends a message to the other application involved in the direct manipulation operation.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
MRESULT DrgSendTransferMsg  (HWND hwndTo, ULONG ulMsgid,
                             MPARAM mpParam1, MPARAM mpParam2)
```

Parameters

hwndTo (HWND) – input
Window handle to which the message is to be sent.

Target *hwndItem* in the DRAGITEM structure.
Source *hwndClient* in the DRAGTRANSFER structure.

ulMsgid (ULONG) – input
Identifier of the message to be sent.

Valid messages are:

DM_ENDCONVERSATION
DM_RENDER
DM_RENDERPREPARE

mpParam1 (MPARAM) – input
First message parameter.

mpParam2 (MPARAM) – input
Second message parameter.

Returns

mresReply (MRESULT) – returns
Message-return data.

Remarks

If the message to be sent is DM_RENDER or DM_RENDERCOMPLETE, the *fsReply* field in DRAGTRANSFER is set to 0 before the message is sent. If the message cannot be sent, FALSE is returned.

When the message to be sent is DM_RENDER, DosGiveSeg is called. DosGiveSeg gives access to the DRAGTRANSFER structure to the process that owns the window indicated by

hwndTo. The use count for the segment in which the DRAGTRANSFER structure exists is incremented.

The process to which the message is being sent must call DrgFreeDragtransfer for the DRAGTRANSFER structure before the segment can be freed.

Related Functions

- DrgPostTransferMsg

Example Code

This function is used to send a message from one window to another when a direct manipulation is in progress. In this example, the function is used to inform the target that the operation is complete and successful.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#include <os2.h>

PDRAGINFO pdinfo;
MPARAM    mp1;
TID       tid;

case DM_DROP:
  pdinfo = (PDRAGINFO) mp1;

  /*************************************************************/
  /* If this is a copy operation, spawn a thread to do the copy  */
  /*************************************************************/
  if (pdinfo->usOperation == DO_COPY)
  {
    DosCreateThread (&tid, CopyThread, pdinfo, FALSE, 4096);
  }
  break;

void Copy Thread (PDRAGINFO pdinfo)
{
  PDRAGITEM pditem;
  USHORT    i;
  ULONG     flResult;
  HAB       hab;
  HMQ       hmq;
  char      szSource[CCH_MAXPATH];
  char      szTarget[CCH_MACPATH];
```

```
  /*********************************************************/
  /* DrgSendTransferMsg needs a message queue, so create one for */
  /* this thread                                           */
  /*********************************************************/
  hab = WinInitialize (0);
  hmq = WinCreateMsgQueue (hab, 0);

  /*********************************************************/
  /* Try to copy each item that was dragged                */
  /*********************************************************/
  for (i = 0; i < pdinfo->cditem; i++)
  {
    /*******************************************************/
    /* Get a pointer to the DRAGITEM                       */
    /*******************************************************/
    pditem = DrgQueryDragitemPtr (pdinfo, i);

    /*******************************************************/
    /* If we could query the source and target names, and the    */
    /* copy was successful, return success                 */
    /*******************************************************/
    if (DrgQueryStrName (pditem->hstrSourceName, sizeof (szSource),
                         szSource)
        DrgQueryStrName (pditem->hstrTargetName, sizeof (szTarget),
                         szTarget)
        !DosCopy (szSource, szTarget, 0))
    {
      flResult = DMFL_TARGETSUCCESSFUL;
    }

    /*******************************************************/
    /* Otherwise, return failure                           */
    /*******************************************************/
    else
    {
      flResult = DMFL_TARGETFAIL;
    }

    /*******************************************************/
    /* Let the source know we're done with this item       */
    /*******************************************************/
    DrgSendTransferMsg (pditem->hwndItem, DM_ENDCONVERSATION,
                        (MPARAM) pditem->ulItemID,
                        (MPARAM) flResult);
  }

  WinDestroyMsgQueue (hmq);
  WinTerminate (hab);
}
```

DrgSetDragImage

This function sets the image that is being dragged.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgSetDragImage (PDRAGINFO pDraginfo, PDRAGIMAGE pdimg,
                      ULONG cdimg, PVOID pReserved)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure. representing the drag operation for which the pointer is to be set.

pdimg (PDRAGIMAGE) – input
Pointer to an array of DRAGIMAGE structures.

These structures describe the images to be drawn under the pointer during the drag.

cdimg (ULONG) – input

Number of DRAGIMAGE structures in the *pdimg* array.

pReserved (PVOID) – input
Reserved value, must be NULL.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_ACCESS_DENIED (0x150D)	The memory block was not allocated properly.
PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32768 to +32767 cannot be converted to a SHORT, and a negative number

	cannot be converted to a ULONG or USHORT.
PMERR_INSUFFICIENT_MEMORY (0x203E)	The operation terminated through insufficient memory.

Remarks

The image that is set with DrgSetDragImage is used only while the pointer is over the target that made the call. If the pointer leaves the original target, the new target can specify an image by calling DrgSetDragImage.

If the new target does not call DrgSetDragImage, the original image that was supplied on the call to DrgDrag is used.

Related Functions

- DrgSetDragPointer

Example Code

This example sets the icon image that is displayed during a direct manipulation operation.

```
#define INCL_GPIBITMAPS  /* GPI Bit Map Functions                 */
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#include <os2.h>
#define ID_BITMAP 257    /* .rc file: "bitmap 257 drgimage.bmp"  */
HPS       hps;           /* Presentation space handle            */
BOOL      flResult;
HAB       hab;
PDRAGINFO pdinfo;
DRAGIMAGE dimg;
HBITMAP   hbm;           /* Bit-map handle                       */
HWND      hwnd;

/********************************************************************/
/* Load a bit map for use as a drag image                           */
/********************************************************************/

case WM_CREATE:
    hps = WinGetPS(hwnd);

    hbm = GpiLoadBitmap(hps,0L,ID_BITMAP,20L,20L);
    WinReleasePS(hps);
    break;

case DM_DRAGOVER:

/********************************************************************/
/* Initialize the DRAGIMAGE structure                               */
/********************************************************************/

      dimg.cb = sizeof(DRAGIMAGE);  /* Size control block       */
      dimg.cptl        = 0;
      dimg.hImage = hbm;            /* Image handle passed to   */
                                    /* DrgDrag                  */
      dimg.sizlStretch.cx = 20L;    /* Size to stretch ico or   */
      dimg.sizlStretch.cy = 20L;    /* bmp to                   */
      dimg.fl = DRG_BITMAP |
                DRG_STRETCH;        /* Stretch to size specified */
      dimg.cxOffset = 0;            /* Offset of the origin of  */
      dimg.cyOffset = 0;            /* the image from the pointer*/
                                    /* hotspot                  */

/********************************************************************/
/* Set the drag image                                               */
/********************************************************************/

flResult= DrgSetDragImage(pdinfo,&dimg,(ULONG)sizeof(dimg), NULL);
```

DrgSetDragitem

This function sets the values in a DRAGITEM structure.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgSetDragitem (PDRAGINFO pDraginfo, PDRAGITEM pDragitem,
                     ULONG cbBuffer, ULONG iItem)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure in which to place the DRAGITEM.

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure to place in DRAGINFO.

cbBuffer (ULONG) – input
Size of the DRAGITEM addressed by *pDragitem*.

iItem (ULONG) – input
Zero-based index of the DRAGITEM to be set.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

This function is used to initialize the DRAGINFO structure before calling DrgDrag.

This function is used only by the source of the drag, not by the target.

Related Functions

- DrgQueryDragitem

Example Code

This example shows a direct manipulation operation between two windows. The actual operation, copying the CONFIG.SYS file to C:\OS2\CONFIG.SYS, is visually represented by a drag and drop of an icon.

```
#define INCL_GPIBITMAPS  /* GPI Bit Map Functions                */
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#define INCL_DOSFILEMGR  /* File Management Functions            */
#define INCL_WININPUT    /* Window Input Functions               */
#include <os2.h>
#include <string.h>
#define ID_WINDOW 255
#define ID_ITEM 256
#define ID_BITMAP 257    /* .rc file: "bitmap 257 drgimage.bmp"  */
HPS       hps;           /* Presentation space handle            */
BOOL      flResult;
HAB       hab;
PDRAGINFO pdinfo;
DRAGITEM  ditem;
DRAGIMAGE dimg;
PDRAGITEM pditem;
HBITMAP   hbm;           /* Bit-map handle                       */
HPOINTER  hptr;          /* Pointer bit-map handle               */
HWND      hwndDrop;
HWND      hwnd;
MPARAM    mp1;
char szBuffer[32];       /* Buffer where intersection string     */
                         /* is returned                          */
char szSource[32];
char szTarget[32];

/********************************************************************/
/* Inside  ClientWindowProc of Source Window                        */
/********************************************************************/

case WM_BEGINDRAG:

/********************************************************************/
/* Initialize the DRAGITEM structure                                */
/********************************************************************/

    ditem.hwndItem = hwnd;      /* Conversation partner             */
    ditem.ulItemID = ID_ITEM;   /* Identifies item being dragged */
    ditem.hstrType = DrgAddStrHandle(DRT_TEXT);     /* Text item */
    ditem.hstrRMF = DrgAddStrHandle("<DRM_OS2FILE,DRF_TEXT>");
    ditem.hstrContainerName = DrgAddStrHandle("C:\\");
    ditem.hstrSourceName = DrgAddStrHandle("C:\\CONFIG.SYS");
    ditem.hstrTargetName = DrgAddStrHandle("C:\\OS2\\CONFIG.SYS");
    ditem.cxOffset = 0; ditem.cyOffset = 0;
    ditem.fsControl = 0; ditem.fsSupportedOps = 0;
```

```
/*********************************************************************/
/* Create the DRAGINFO structure                                     */
/*********************************************************************/

    pdinfo = DrgAllocDraginfo(1);

/*********************************************************************/
/* Initialize the DRAGIMAGE structure                                */
/*********************************************************************/

    dimg.cb = sizeof(DRAGIMAGE); /* Size control block           */
    dimg.cptl        = 0;
    dimg.hImage = hbm;           /* Image handle passed to       */
                                 /* DrgDrag                      */
    dimg.sizlStretch.cx = 20L;   /* Size to stretch ico or       */
    dimg.sizlStretch.cy = 20L;   /* bmp to                       */
    dimg.fl = DRG_BITMAP |
              DRG_STRETCH;       /* Stretch to size specified    */
    dimg.cxOffset = 0;           /* Offset of the origin of the  */
    dimg.cyOffset = 0;           /* image from the pointer       */
                                 /* hotspot                      */

flResult= DrgSetDragitem(pdinfo, &ditem, (ULONG)sizeof(ditem), 0);

/*********************************************************************/
/* Perform the drag operation:                                       */
/*********************************************************************/

      hwndDrop = DrgDrag(hwnd,  /* Source of the drag             */
                  pdinfo,       /* Pointer to DRAGINFO structure */
                 (PDRAGIMAGE)&dimg,  /* Drag image               */
                  1,           /* Size of the pdimg array        */
                  VK_ENGDRAG,  /* Release of drag button         */
                               /* terminates the drag            */
                  NULL);       /* Reserved                       */

/*********************************************************************/
/* Inside ClientWindowProc of Target Window                          */
/*********************************************************************/

  case DM_DRAGOVER:

      pdinfo = MPFROMP(mp1);
      pditem = DrgQueryDragitemPtr(pdinfo,0);

      flResult = DrgVerifyTrueType(pditem,"DRF_TEXT");

      if(!flResult)
```

```
/*****************************************************************/
/* Inform the application that you will accept the drop          */
/*****************************************************************/

      return(MRFROM2SHORT(DOR_DROP,DO_COPY));

  case DM_DROP:
      pdinfo = MPFROMP(mp1);
      pditem = DrgQueryDragitemPtr(pdinfo,0);

/*****************************************************************/
/* Perform the operation represented by the direct manipulation  */
/*****************************************************************/

DrgQueryStrName(pditem->hstrSourceName,sizeof(szSource),szSource);
DrgQueryStrName(pditem->hstrTargetName,sizeof(szTarget),szTarget);
flResult = DosCopy(szSource,szTarget,0L);

/*****************************************************************/
/* If operation is successful, return DMFL_TARGETSUCCESSFUL      */
/*****************************************************************/

if(!flResult)
{
      DrgSendTransferMsg(pditem->hwndItem,
                         DM_ENDCONVERSATION,
                         MPFROMLONG(pditem->ulItemID),
                         MPFROMLONG(DMFL_TARGETSUCCESSFUL));
}

/*****************************************************************/
/* Otherwise, return DMFL_TARGETFAIL                             */
/*****************************************************************/

else
{
      DrgSendTransferMsg(pditem->hwndItem,
                         DM_ENDCONVERSATION,
                         MPFROMLONG(pditem->ulItemID),
                         MPFROMLONG(DMFL_TARGETFAIL));
}
```

DrgSetDragPointer

This function sets the pointer to be used while over the current target.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgSetDragPointer (PDRAGINFO pDraginfo, HPOINTER hptrHandle)
```

Parameters

pDraginfo (PDRAGINFO) – input
Pointer to the DRAGINFO structure. to be used for this drag.

hptrHandle (HPOINTER) – input
Handle to the pointer to use.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HPTR (0x101B)	An invalid pointer handle was specified.

Remarks

This function sets the pointer to be used to indicate the hot spot while dragging over the current target.

The pointer that is set with DrgSetDragPointer is used only while it is over the current target. The pointer is reset to the default when a new target is dragged over.

This function can be used by an application to provide meaningful augmentation emphasis for an operation that is unknown to the system (for example, swap).

When the drag pointer is successfully set, TRUE is returned.

Related Functions

- DrgSetDragImage

Example Code

This example uses the DrgSetDragPointer function to set the image used for the pointer while the pointer is over the target during a direct manipulation operation.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#include <OS2.H>
BOOL      flResult;
PDRAGITEM pditem;
HPOINTER  hptrCrossHair;
MPARAM    mp1;
char szBuffer[32];

case DM_DRAGOVER:
    DrgSetDragPointer ((PDRAGINFO) mp1, hptrCrossHair);
```

DrgVerifyNativeRMF

This function determines if the native rendering mechanism and format of an object match any supplied by the application.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgVerifyNativeRMF  (PDRAGITEM pDragitem, PSZ pRMF)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure. whose native rendering mechanism and format are to be verified.

pRMF (PSZ) – input
A String specifying the rendering mechanism and format.

The string is of the form:

```
mechfmt[,mechfmt,mechfmt,...]
```

where `mechfmt` can be in either of these formats:

- <mechanism(1),format(1)>
- (mechanism(1)[, mechanism(n)...]) (format(1)[,format(n)...])

Returns

rc (BOOL) – returns
Validity indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

This function determines if the native rendering mechanism and format of a dragged object are understood by the target.

If TRUE is returned, the target may be able to carry out the action indicated by the direct manipulation itself, or it can select the native rendering mechanism and format as those to be used for the data exchange.

Related Functions

- DrgQueryNativeRMF

Example Code

This example determines if the native rendering mechanism and format of an object match any supplied by the application.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#include <OS2.H>

DRAGITEM Dragitem;       /* DRAGITEM structure whose native      */
                         /* rendering mechanism and format are   */
                         /* to be verified                       */

char pszRMF[25];         /* A string specifying the rendering    */
                         /* mechanism and format.  The string is */
                         /* of the form:                         */
                         /*                                      */
                         /* mechfmt[,mechfmt,mechfmt,...],       */
                         /*                                      */
                         /* where 'mechfmt' can be in either of  */
                         /* these formats:                       */
                         /*                                      */
                         /*  o <mechanism(1),format(1)>          */
                         /*  o (mechanism(1)[, mechanism(n)...]) */
                         /*    (format(1)[,format(n)...])        */


strcpy(pszRMF,"(DRM_OS2FILE,DRF_TEXT)");
                         /* Mechanism is an OS/2 file and format */
                         /* is a null-terminated string.  See    */
                         /* the DRAGITEM structure for valid     */
                         /* formats.                             */

if(DrgVerifyNativeRMF(&Dragitem, pszRMF))
{
                         /* Code block                           */
}
```

DrgVerifyRMF

This function determines if a given rendering mechanism and format are supported for a dragged object.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgVerifyRMF (PDRAGITEM pDragitem, PSZ pMech, PSZ pFormat)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose native rendering mechanism and format are to be validated.

pMech (PSZ) – input
String specifying the rendering mechanism to search for.

NULL will match any mechanism.

pFormat (PSZ) – input
String specifying the rendering format to search for.

NULL will match any format.

Returns

rc (BOOL) – returns
Validity indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

This function determines if a given rendering mechanism and format ordered pair are represented in the set of valid pairs specified by *hstrRMF* for the dragged object.

Related Functions

- DrgVerifyNativeRMF

Example Code

This example determines if a given rendering mechanism and format are supported for a dragged object.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#include <OS2.H>

DRAGITEM Dragitem;       /* DRAGITEM structure whose native     */
                         /* rendering mechanism and format are  */
                         /* to be validated                     */
char pszMech[] = "DRM_OS2FILE";
                         /* A string specifying the rendering   */
                         /* mechanism to search for             */
char pszFormat[] = "DRF_TEXT";
                         /* A string specifying the rendering   */
                         /* format to search for                */

if(DrgVerifyRMF(&Dragitem, pszMech, pszFormat))
                         /* Mechanism is an OS/2 file and format */
                         /* is a null-terminated string          */
{
                         /* Code block                          */
}
```

DrgVerifyTrueType

This function determines if the true type of a dragged object matches an application-supplied type string.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgVerifyTrueType (PDRAGITEM pDragitem, PSZ pType)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose true type is to be verified.

pType (PSZ) – input
String specifying a type.

This string is in the format:

```
type[,type...]
```

Returns

rc (BOOL) – returns
Validity indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

If an item in the string pointed to by *pType* matches the first type in the string associated with *hstrType* in the DRAGITEM structure, TRUE is returned.

A target application uses this function to determine if it supports the true type of a dragged object. If the application does not support the true type, it can either disallow a drop or change its default operation. If the default operation is a move, the drop should be disallowed, or the operation changed to a copy to prevent any loss of data for the object.

Related Functions

- DrgQueryTrueType
- DrgVerifyType
- DrgVerifyTypeSet

Example Code

This example verifies whether a given type is present in the list of types defined for a drag object.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#include <OS2.H>

BOOL      fValid;
DRAGITEM  Dragitem;      /* DRAGITEM structure whose hstrType is */
                         /* to be verified                       */

char pszType[8];         /* A string specifying the types to     */
                         /* search for                           */

strcpy(pszType,DRT_EXE); /* Executable file type. See the      */
                           /* DRAGINFO structure for valid       */
                           /* types.                             */

fValid = DrgVerifyTrueType(&Dragitem, pszType);
```

DrgVerifyType

This function verifies whether a given type is present in the list of types defined for a drag object.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgVerifyType (PDRAGITEM pDragitem, PSZ pType)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose *hstrType* is to be verified.

pType (PSZ) – input
String specifying the types to search for.

This string is in the format:

```
type[,type...]
```

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32768 to +32767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.
PMERR_INSUFFICIENT_MEMORY (0x203E)	The operation terminated through insufficient memory.

Remarks

If at least one of the types specified by *pType* is present in *hstrType* in the DRAGITEM structure, TRUE is returned.

Related Functions

- DrgVerifyTrueType
- DrgVerifyTypeSet

Example Code

This example verifies whether a given type is present in the list of types defined for a drag object.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#include <OS2.H>


BOOL     fValid;
DRAGITEM Dragitem;       /* DRAGITEM structure whose hstrType is */
                         /* to be verified                       */
char pszType[] = DRT_EXE;
                         /* A string specifying the types to     */
                         /* search for                           */

fValid = DrgVerifyType(&Dragitem, pszType);
```

DrgVerifyTypeSet

This function returns the intersection of the contents of the string associated with the type-string handle for an object and an application-specified type string.

Syntax

```
#define INCL_WINSTDDRAG
#include <os2.h>
BOOL DrgVerifyTypeSet (PDRAGITEM pDragitem, PSZ pType, ULONG cbBuflen,
                       PSZ pBuffer)
```

Parameters

pDragitem (PDRAGITEM) – input
Pointer to the DRAGITEM structure whose *hstrType* is to be verified.

pType (PSZ) – input
String specifying the types to search for.

This string is in the format:

```
type[,type...]
```

cbBuflen (ULONG) – input
Size of the return buffer.

The buffer should be at least one byte longer than the length of the string pointed to by *pType*. Must be > 0.

pBuffer (PSZ) – output
Buffer where the intersection string is returned.

Returns

rc (BOOL) – returns
Match indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

If at least one of the types specified by *pType* is present in *hstrType* in the DRAGITEM structure, TRUE is returned.

Related Functions

- DrgVerifyType
- DrgVerifyTrueType

Example Code

In this example, the DrgVerifyTypeSet function is used to determine whether DRT_TEXT is among the types associated with the object. If it is, the drop is accepted.

```
#define INCL_WINSTDDRAG  /* Direct Manipulation (Drag) Functions */
#include <OS2.H>
#include <stdio.h>
BOOL     flResult;
DRAGITEM pditem;
char     szBuffer[32];

case DM_DRAGOVER:

   flResult = DrgVerifyTypeSet(&pditem,
                               DRT_TEXT,
                               sizeof(szBuffer),
                               szBuffer);

   flResult = strcmp(szBuffer,DRT_TEXT);

   /*************************************************************/
   /* See if the object is an OS/2 file as well as being of text */
   /* format.  AND result flag with previous result flag to get  */
   /* the "effective" return code.                               */
   /*************************************************************/

   flResult = DrgVerifyRMF(&pditem,"DRM_OS2FILE","DRF_TEXT");

   /*************************************************************/
   /* See if DRT_TEXT is the true type of the object             */
   /*************************************************************/

   flResult = DrgVerifyTrueType(&pditem,"DRF_TEXT");

   if(!flResult)

   /*************************************************************/
   /* Inform the application that you will accept the drop       */
   /*************************************************************/

   return(MRFROM2SHORT(DOR_DROP, DO_COPY));
   break;
```

Chapter 4. Dynamic Data Formatting Functions

The Information Presentation Facility (IPF) manages online, context-sensitive help information. Dynamic Data Formatting (DDF) is used for displaying dynamic help information. This section contains the DDF functions used to accomplish Dynamic Data Formatting. The following table shows all the dynamic data formatting (Ddf) functions in alphabetic order.

C Name
DdfBeginList
DdfBitmap
DdfEndList
DdfHyperText
DdfInform
DdfInitialize
DdfListItem
DdfMetafile
DdfPara
DdfSetColor
DdfSetFont
DdfSetFontStyle
DdfSetFormat
DdfSetTextAlign
DdfText

DdfBeginList

This function begins a definition list in the DDF buffer; it corresponds to the **:dl.** (definition list) tag.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfBeginList (HDDF hddf, ULONG ulWidthDT, ULONG fBreakType,
                   ULONG fSpacing)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

ulWidthDT (ULONG) – input
Width of the definition term.

fBreakType (ULONG) – input
Type of line break to use.

The following constants may be specified:

HMBT_ALL	Start all definition descriptions on the next line, regardless of the actual lengths of definition terms.
HMBT_FIT	Start definition description on the next line only when the definition term is longer than the width specified.
HMBT_NONE	Do not start the definition description on the next line, even when the definition term is longer than the width specified.

fSpacing (ULONG) – input
Spacing between definitions.

Only the following constants may be specified:

HMLS_SINGLELINE	Do not insert a blank line between each definition description and the next definition term.
HMLS_DOUBLELINE	Insert a blank line between each definition description and the next definition term.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)	Not enough memory is available.
HMERR_DDF_LIST_UNCLOSED (0x3007)	An attempt was made to nest a list.
HMERR_DDF_LIST_BREAKTYPE (0x3009)	The value of BreakType is not valid.
HMERR_DDF_LIST_SPACING (0x300A)	The value for Spacing is not valid.

Remarks

Once this function has been called, use of any DDF function other than DdfListItem, DdfSetColor, and DdfEndList may produce unpredictable results.

Related Functions

- DdfBitmap
- DdfEndList
- DdfHyperText
- DdfInform
- DdfInitialize
- DdfListItem
- DdfMetafile
- DdfPara
- DdfSetColor
- DdfSetFont
- DdfSetFontStyle
- DdfSetFormat
- DdfSetTextAlign
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, the example uses DdfBeginList to indicate the beginning of a definition list in the DDF buffer (this corresponds to the IPF dl tag). For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_DDF                /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>

struct _LISTITEM        /* definition list                      */
{
    PSZ Term;
    PSZ Desc;
} Definition[2] = {{"MVS", "Multiple Virtual System"},
                   {"VM",  "Virtual Machine"}};

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2)
{

    HWND   hwndParent;
    HWND   hwndInstance;
    HDDF   hDdf;            /* DDF handle                       */
    SHORT  i;               /* loop index                       */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                  hwndInstance,  /* Handle of help instance */
                  0L,            /* Default buffer size     */
                  0L             /* Default increment       */
                  );

:figseg=5.
        if (hDdf == NULLHANDLE)       /* Check return code      */
        {
            return (MRESULT)FALSE;
        }

        /* begin definition list */
        if (!DdfBeginList(hDdf, 3L, HMBT_ALL, HMLS_SINGLELINE))
        {
            return (MRESULT)FALSE;
        }
```

```
        /* insert 2 entries into definition list */
        for (i=0; i < 2; i++)
        {
            if (!DdfListItem(hDdf, Definition[i].Term,
                                   Definition[i].Desc))
            {
               return (MRESULT)FALSE;
            }
        }

        /* terminate definition list */
        if (!DdfEndList(hDdf))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
}
```

DdfBitmap

This function places a reference to a bit map in the DDF buffer.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfBitmap (HDDF hddf, HBITMAP hbm, ULONG fAlign)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

hbm (HBITMAP) – input
Standard Presentation Manager bit map handle.

fAlign (ULONG) – input
Text justification flag.

Any of the following values can be specified:

ART_LEFT	Left-justify the bit map.
ART_RIGHT	Right-justify the bit map.
ART_CENTER	Center the bit map.
ART_RUNIN	Allow the bit map to be reflowed with text.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)	Not enough memory is available.
HMERR_DDF_ALIGN_TYPE (0x3002)	The alignment type is not valid.

Remarks

The handle to the presentation space in which the bit map was created cannot be freed by the application while the panel is displayed.

Note: There is a (3-byte + size of HBITMAP structure). ESC code overhead in the DDF internal buffer for this function. There is a 1-byte ESC code overhead required for the *fAlign* flag.

Related Functions

- DdfBeginList
- DdfEndList
- DdfHyperText
- DdfInform
- DdfInitialize
- DdfListItem
- DdfMetafile
- DdfPara
- DdfSetColor
- DdfSetFont
- DdfSetFontStyle
- DdfSetFormat
- DdfSetTextAlign
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, the example obtains a device context (DevOpenDC), creates a presentation space (GpiCreatePS), and loads a bit map (GpiLoadBitmap). It then uses DdfBitmap to place a reference to the bit map in the DDF buffer. For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management     */
#define INCL_WINMESSAGEMGR      /* Message management            */
#define INCL_GPICONTROL         /* Basic PS control              */
#define INCL_GPIBITMAPS         /* Bit maps and Pel Operations   */
#define INCL_GPIPRIMITIVES      /* Drawing Primitives/Attributes */
#define INCL_DDF                /* Dynamic Data Facility         */
#include <os2.h>
#include <pmhelp.h>

#define ACVP_HAB  12
#define BM_HPS    16
#define BM_HDC    20
#define BM_HWND   24
#define ID_LEFT   255
```

```
MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;     /* parent window                          */
    HWND    hwndInstance;   /* help instance window                   */
    HDDF    hDdf;           /* DDF handle                             */
    HDC     hdc;            /* device context handle                  */
    HPS     hps;            /* presentation space handle              */
    HAB     hab;            /* anchor block handle                    */
    SIZEL   sizel = {0L,0L};/* size of new PS                         */
    HBITMAP hBitmap;        /* bit map handle                         */
    HMODULE hModule;        /* module handle                          */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                        hwndInstance,  /* Handle of help instance */
                        0L,            /* Default buffer size     */
                        0L             /* Default increment       */
                        );

        if (hDdf == NULLHANDLE)        /* Check return code       */
        {
            return (MRESULT)FALSE;
        }

        /* get module handle for bit map */
        DosQueryModuleHandle("bitmap", &hModule);
        if (hModule == NULLHANDLE)
        {
            return (MRESULT)FALSE;
        }

        /* get hab for this window */
        if ((hab = (HAB)WinQueryWindowULong(hwnd, ACVP_HAB)) == NULLHANDLE)
        {
            return (MRESULT)FALSE;
        }
```

```
        /* create a device context */
        if ((hdc = DevOpenDC(hab, OD_MEMORY, "*", 0L,
                             (PDEVOPENDATA)NULL, (HDC)NULL)) == NULLHANDLE)
        {
           return (MRESULT)FALSE;
        }

        /* save hdc in reserved word */
        WinSetWindowULong(hwnd, BM_HDC, (ULONG)hdc);

        /* create a noncached micro presentation space */
        /* and associate it with the window */
        if ((hps = GpiCreatePS(hab, hdc, &sizel,
                  PU_PELS | GPIF_DEFAULT
                  | GPIT_MICRO | GPIA_ASSOC)) == NULLHANDLE)
        {
           return (MRESULT)FALSE;
        }

        /* save hps in reserved word */
        WinSetWindowULong(hwnd, BM_HPS, (ULONG)hps);

        /* Load the Bit map to display */
        if ((hBitmap = GpiLoadBitmap(hps, hModule, ID_LEFT, 300L,
                                     300L)) == NULLHANDLE)
        {
            return (MRESULT)FALSE;
        }

        /* save bit map hwnd in reserved word */
        WinSetWindowULong(hwnd, BM_HWND, (ULONG)hBitmap);

        /* Display the bit map align left */
        if (!DdfBitmap(hDdf, hBitmap, ART_LEFT))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;

    case WM_CLOSE:
        /* release PS, DC, and bit map */
        GpiDestroyPS((HPS)WinQueryWindowULong(hwnd, BM_HPS));
        DevCloseDC((HDC)WinQueryWindowULong(hwnd, BM_HDC));
        GpiDeleteBitmap((HBITMAP)WinQueryWindowULong(hwnd, BM_HWND));
        WinDestroyWindow(WinQueryWindow(hwnd, QW_PARENT));
        return (MRESULT)TRUE;
    }
}
```

DdfEndList

This function terminates the definition list initialized by DdfBeginList.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfEndList  (HDDF  hddf)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_LIST_UNINITIALIZED (0x3008)	No definition list has been initialized by DdfBeginList.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfHyperText
- DdfInform
- DdfInitialize
- DdfListItem
- DdfMetafile
- DdfPara
- DdfSetColor
- DdfSetFont
- DdfSetFontStyle
- DdfSetFormat
- DdfSetTextAlign
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, the example uses DdfBeginList to indicate the beginning of a definition list in the DDF buffer (this corresponds to the IPF dl tag). For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_DDF                /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>

struct _LISTITEM        /* definition list                      */
{
    PSZ Term;
    PSZ Desc;
} Definition[2] = {{"MVS", "Multiple Virtual System"},
                   {"VM",  "Virtual Machine"}};

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2)
{

    HWND   hwndParent;
    HWND   hwndInstance;
    HDDF   hDdf;            /* DDF handle                       */
    SHORT  i;               /* loop index                       */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                  hwndInstance,  /* Handle of help instance */
                  0L,            /* Default buffer size     */
                  0L             /* Default increment       */
                  );

        if (hDdf == NULLHANDLE)       /* Check return code       */
        {
            return (MRESULT)FALSE;
        }
```

```
        /* begin definition list */
        if (!DdfBeginList(hDdf, 3L, HMBT_ALL, HMLS_SINGLELINE))
        {
            return (MRESULT)FALSE;
        }

        /* insert 2 entries into definition list */
        for (i=0; i < 2; i++)
        {
            if (!DdfListItem(hDdf, Definition[i].Term,
                                   Definition[i].Desc))
            {
               return (MRESULT)FALSE;
            }
        }

        /* terminate definition list */
        if (!DdfEndList(hDdf))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
}
```

DdfHyperText

This function defines a hypertext link to another panel, which is equal to a link of reftype=hd. Links to footnotes, launch links and links to external databases are not supported.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfHyperText  (HDDF hddf, PSZ pText, PSZ pReference,
                    ULONG fReferenceType)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

pText (PSZ) – input
Hypertext phrase.

pReference (PSZ) – input
Pointer to a string containing the link reference.

The value of this parameter depends on the value of *fReferenceType*.

If *fReferenceType* is REFERENCE_BY_RES
pReference must contain a pointer to a numeric string containing the res number; otherwise it will default to a res number of zero. Valid values are 1 - 64000; all other values are reserved.

If *fReferenceType* is REFERENCE_BY_ID
pReference must contain a pointer to a string containing the alphanumeric identifier of the destination panel.

fReferenceType (ULONG) – input
Reference Type.

This parameter specifies whether you are linking via a resource identifier (res number) or via an alphanumeric identifier.

REFERENCE_BY_RES	To link via a resource identifier.
REFERENCE_BY_ID	To link via an alphanumeric identifier.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)	Not enough memory is available.
HMERR_DDF_REFTYPE (0x3006)	The reference type is not valid.

Remarks

There is a 3-byte ESC code overhead in the DDF internal buffer for each word in the text buffer. There is a 1-byte ESC code overhead for each blank and for each newline character. If *fReferenceType* is REFERENCE_BY_ID, then there is a (3-byte + Reference length) ESC code overhead. For a *fReferenceType* of REFERENCE_BY_RES, the overhead is 5 bytes. Finally, there is a 3-byte ESC code overhead that is required for ending the hypertext link.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfEndList
- DdfInform
- DdfInitialize
- DdfListItem
- DdfMetafile
- DdfPara
- DdfSetColor
- DdfSetFont
- DdfSetFontStyle
- DdfSetFormat
- DdfSetTextAlign
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, the example uses DdfHyperText to create a hypertext link with another resource. For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_DDF                /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>

PSZ    Text = "This text is a HYPERTEXT message.\n"; /* hypertext string  */
PSZ    ResID = "1";     /* Resource identifier                  */

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND   hwndParent;
    HWND   hwndInstance;
    HDDF   hDdf;            /* DDF handle                           */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                    hwndInstance,  /* Handle of help instance */
                    0L,            /* Default buffer size     */
                    0L             /* Default increment       */
                    );

        if (hDdf == NULLHANDLE)       /* Check return code       */
        {
            return (MRESULT)FALSE;
        }

        /* create hypertext link with resource 1 */
        if (!DdfHyperText(hDdf, (PSZ)Text, ResID, REFERENCE_BY_RES))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
}
```

DdfInform

This function defines a hypertext inform link, which sends an HM_INFORM message to the application with a **res=** attribute. It also corresponds to the link tag with reftype=inform.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfInform (HDDF hddf, PSZ pText, ULONG resInformNumber)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

pText (PSZ) – input
Hypertext phrase.

resInformNumber (ULONG) – input
Res number associated with this hypertext field.

Possible values are 1 to 64000; all other values are reserved.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)	Not enough memory is available.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfEndList
- DdfHyperText
- DdfInitialize
- DdfListItem
- DdfMetafile
- DdfPara
- DdfSetColor
- DdfSetFont
- DdfSetFontStyle

- DdfSetFormat
- DdfSetTextAlign
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, the example uses DdfInform to create a hypertext inform link with another resource (corresponds to the IPF :link. tag with reftype=inform). For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_DDF                /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>

PSZ    Text = "This text is a HYPERTEXT message.\n"; /* hypertext string  */
MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND   hwndParent;
    HWND   hwndInstance;
    HDDF   hDdf;            /* DDF handle                           */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                    MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                        hwndInstance,  /* Handle of help instance */
                        0L,            /* Default buffer size     */
                        0L             /* Default increment       */
                        );

        if (hDdf == NULLHANDLE)       /* Check return code       */
        {
            return (MRESULT)FALSE;
        }

        /* create hypertext inform link with resource 1 */
        if (!DdfInform(hDdf, (PSZ)Text, 1L))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
}
```

DdfInitialize

This function initializes the IPF internal structures for dynamic data formatting and returns a DDF handle. The application uses this handle to refer to a particular DDF panel.

Syntax

```
#define INCL_DDF
#include <os2.h>
HDDF DdfInitialize (HWND hwndHelpInstance, ULONG cbBuffer,
                    ULONG ulIncrement)
```

Parameters

hwndHelpInstance (HWND) – input
Handle of a help instance.

cbBuffer (ULONG) – input
Initial length of internal buffer.

Initial length of internal buffer where DDF information is to be stored. If this field is 0L, a default value of 1K is defined. The maximum value is 61 440 bytes (60KB).

ulIncrement (ULONG) – input
Amount by which to increment the buffer size, if necessary.

If this field is NULL, a default value of 256 bytes is defined. The maximum value is 60KB.

Returns

hddf (HDDF) – returns
Handle to DDF.

A handle to DDF is returned if initialization was successful. Otherwise, the value returned is NULL, indicating that an error has occurred because of insufficient memory or incorrect instance.

Remarks

At initialization, the default for dynamic data display is that text aligned on the left, and formatting is turned on.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfEndList
- DdfHyperText

- DdfInform
- DdfListItem
- DdfMetafile
- DdfPara
- DdfSetColor
- DdfSetFont
- DdfSetFontStyle
- DdfSetFormat
- DdfSetTextAlign
- DdfText

Example Code

This example shows how to initialize and use the Dynamic Data Facility for displaying an online document. Two functions are defined: the first, SampleObj, creates a window that will display the online information and specifies the second function, SampleWindowProc, as the corresponding window procedure. These two functions are compiled into a DLL and exported, so that IPF can invoke them when it encounters the :ddf and :acviewport tags during execution. The :acviewport tag will specify the DLL name and the SampleObj function; when IPF calls SampleObj, it initializes an application-controlled window with SampleWindowProc as the window procedure and returns the window handle. Later, when IPF encounters the :ddf tag, it will send SampleWindowProc an HM_QUERY_DDF_DATA message. At this point, before calling any of the DDF API, DdfInitialize must first be called to initiate a DDF buffer, after which the other DDF API can be called to display the online information.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_WINDIALOGS         /* Dialog boxes                 */
#define INCL_DDF                /* Dynamic Data Facility        */
#define INCL_32
#include <os2.h>
#include <pmhelp.h>

#define COM_HWND 4                /* window word offsets        */
#define PAGE_HWND 8
#define ACVP_HAB  12

USHORT DdfClass = FALSE;

MRESULT EXPENTRY SampleWindowProc(HWND hWnd, ULONG Message,
                                  MPARAM lParam1, MPARAM lParam2);
```

```
USHORT APIENTRY SampleObj(PACVP pACVP, PCH Parameter)
{
HWND DdfHwnd;           /* Client window handle                  */
HWND DdfCHwnd;          /* Child window handle                   */
HWND PreviousHwnd;      /* Handle for setting comm window active */

    /* register DDF Base class if not registered already */
    if (!DdfClass)
    {
        if (!WinRegisterClass(
                  pACVP->hAB,        /* Anchor block handle           */
                  "CLASS_Ddf",       /* Application window class name */
                  SampleWindowProc,  /* Address of window procedure   */
                  CS_SYNCPAINT |     /* Window class style            */
                  CS_SIZEREDRAW |
                  CS_MOVENOTIFY,
                  20))               /* Extra storage                 */
        {
            return TRUE;
        }
        DdfClass = TRUE;
    }

    /*  create standard window  */
    if (!(DdfHwnd = WinCreateStdWindow(
                     pACVP->hWndParent, /* ACVP is parent       */
                     0L,                /* No class style       */
                     NULL,              /* Frame control flag   */
                     "CLASS_Ddf",       /* Window class name    */
                     NULL,              /* No title bar         */
                     0L,                /* No special style     */
                     0L,                /* Resource in .EXE     */
                     0,                 /* No window identifier */
                     &DdfCHwnd )))      /* Client window handle */
    {
        return FALSE;
    }

    /* store the frame window handle in ACVP data structure */
    pACVP->hWndACVP = DdfHwnd;

    /* set this window as active communication window */
    PreviousHwnd = (HWND)WinSendMsg(pACVP->hWndParent,
                          HM_SET_OBJCOM_WINDOW,
                          MPFROMHWND(DdfHwnd), NULL);
```

```
    /* save returned communication hwnd in reserved word */
    WinSetWindowULong(DdfCHwnd, COM_HWND, (ULONG)PreviousHwnd);

    /* save anchor block handle in reserved word */
    WinSetWindowULong (DdfCHwnd, ACVP_HAB, (ULONG)pACVP->hAB);

    return FALSE;
} /* SampleObj */

MRESULT EXPENTRY SampleWindowProc(HWND hWnd, ULONG Message,
                                  MPARAM lParam1, MPARAM lParam2)
{
    HWND   hwndParent;       /* parent window                        */
    HWND   hwndHelpInstance; /* help instance window                 */
    HDDF   hDdf;             /* DDF handle                           */
    ULONG  DdfID;            /* DDF resource id                      */

    switch (Message)
    {
    case HM_QUERY_DDF_DATA:
        WinSetWindowULong(hWnd, PAGE_HWND, LONGFROMMP(lParam1));
        DdfID = LONGFROMMP(lParam2);
        hwndParent = WinQueryWindow(hWnd, QW_PARENT);
        hwndParent = WinQueryWindow(hwndParent, QW_PARENT);
        hwndHelpInstance = (HWND)WinSendMsg(hwndParent, HM_QUERY,
                                  MPFROMSHORT(HMQW_INSTANCE), NULL);

        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                        hwndHelpInstance, /* Handle of help instance */
                        0L,               /* Default buffer size     */
                        0L                /* Default increment       */
                        );

        if (hDdf == NULLHANDLE)       /* Check return code       */
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;

    default:
        return (WinDefWindowProc(hWnd, Message, lParam1, lParam2));
    }
} /* SampleWindowProc */
```

DdfListItem

This function inserts a definition list entry in the DDF buffer; it corresponds to a combination of the **:dt.** (definition term) and **:dd.** (definition define) tags.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfListItem  (HDDF hddf, PSZ pTerm, PSZ pDescription)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

pTerm (PSZ) – input
Term portion of the definition list entry.

pDescription (PSZ) – input
Description portion of the definition list entry.

Note: There is a 3-byte ESC code overhead in the DDF internal buffer for each word in both the term and the description. There is a 1-byte ESC code overhead for each blank and for each newline character. For each list item, there is a 5-byte ESC code overhead for the margin alignment.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)	Not enough memory is available.
HMERR_DDF_LIST_UNINITIALIZED (0x3008)	No definition list has been initialized by DdfBeginList.

Remarks

The handle to the presentation space in which the bit map was created cannot be freed by the application while the panel is displayed.

Note: There is a (3-byte + size of HBITMAP structure) ESC code overhead in the DDF internal buffer for this function. There is a 1-byte ESC code overhead required for the Align flag.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfEndList
- DdfHyperText
- DdfInform
- DdfInitialize
- DdfMetafile
- DdfPara
- DdfSetColor
- DdfSetFont
- DdfSetFontStyle
- DdfSetFormat
- DdfSetTextAlign
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, the example uses DdfBeginList to indicate the beginning of a definition list in the DDF buffer (this corresponds to the IPF dl tag). For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_DDF                /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>

struct _LISTITEM        /* definition list                      */
{
    PSZ Term;
    PSZ Desc;
} Definition[2] = {{"MVS", "Multiple Virtual System"},
                   {"VM",  "Virtual Machine"}};

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2)
{

    HWND   hwndParent;
    HWND   hwndInstance;
    HDDF   hDdf;            /* DDF handle                        */
    SHORT i;                /* loop index                       */
```

```
    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                  hwndInstance,  /* Handle of help instance */
                  0L,            /* Default buffer size     */
                  0L             /* Default increment       */
                  );

        if (hDdf == NULLHANDLE)       /* Check return code       */
        {
            return (MRESULT)FALSE;
        }

        /* begin definition list */
        if (!DdfBeginList(hDdf, 3L, HMBT_ALL, HMLS_SINGLELINE))
        {
            return (MRESULT)FALSE;
        }

        /* insert 2 entries into definition list */
        for (i=0; i < 2; i++)
        {
            if (!DdfListItem(hDdf, Definition[i].Term,
                                   Definition[i].Desc))
            {
               return (MRESULT)FALSE;
            }
        }

        /* terminate definition list */
        if (!DdfEndList(hDdf))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
}
```

DdfMetafile

This function places a reference to a metafile into the DDF buffer.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfMetafile  (HDDF hddf, HMF hmf, PRECTL prclRect)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

hmf (HMF) – input
The handle of the metafile to display.

prclRect (PRECTL) – input
Size of the rectangle.

If not NULL, contains the size of the rectangle in which the metafile will be displayed. The aspect ratio of the metafile is adjusted to fit this rectangle.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)	Not enough memory is available.

Remarks

There is a 3-byte ESC code overhead in the DDF internal buffer for this function. There is also a (MetaFilename length) overhead. Finally, the *prclRect* variable requires an additional 16 bytes of overhead in the DDF internal buffer.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfEndList
- DdfHyperText
- DdfInform

- DdfInitialize
- DdfListItem
- DdfPara
- DdfSetColor
- DdfSetFont
- DdfSetFontStyle
- DdfSetFormat
- DdfSetTextAlign
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, and loading a metafile with GpiLoadMetaFile, the example uses DdfMetafile to place a reference to the metafile in the DDF buffer. For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_DDF                /* Dynamic Data Facility        */
#define INCL_GPIMETAFILES       /* MetaFiles                    */
#include <os2.h>
#include <pmhelp.h>

#define MF_HWND   0
#define ACVP_HAB  4

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND   hwndParent;
    HAB    hab;
    HWND   hwndInstance;    /* help instance window                 */
    HDDF   hDdf;            /* DDF handle                           */
    HMF    hwndMetaFile;    /* metafile handle                      */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                MPFROMSHORT( HMQW_INSTANCE ), NULL );
```

```
        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                    hwndInstance,  /* Handle of help instance */
                    0L,            /* Default buffer size     */
                    0L             /* Default increment       */
                    );

        if (hDdf == NULLHANDLE)       /* Check return code       */
        {
            return (MRESULT)FALSE;
        }

        /* get hab for this window */
        if ((hab = (HAB)WinQueryWindowULong(hwnd, ACVP_HAB)) == NULLHANDLE)
        {
           return (MRESULT)FALSE;
        }

        /* Load the Metafile to display */
        if ((hwndMetaFile = GpiLoadMetaFile(hab, "SAMP.MET")) == NULLHANDLE)
        {
            return (MRESULT)FALSE;
        }

        /* Save MetaFile hwnd in reserved word */
        WinSetWindowULong(hwnd, MF_HWND, hwndMetaFile);

        if (!DdfMetafile(hDdf, hwndMetaFile, NULL))
        {
            return (MRESULT)FALSE;
        }

        return (hDdf);

    case WM_CLOSE:
        GpiDeleteMetaFile((HMF)WinQueryWindowULong(hwnd, MF_HWND));
        WinDestroyWindow(WinQueryWindow(hwnd, QW_PARENT));

        return (MRESULT)TRUE;
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}
```

DdfPara

This function creates a paragraph within the DDF buffer. It corresponds to the **:p.** tag. This function places a reference to a bit map in the DDF buffer.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfPara  (HDDF hddf)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)	Not enough memory is available.

Remarks

There is a 1-byte ESC code overhead in the DDF internal buffer for this function.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfEndList
- DdfHyperText
- DdfInform
- DdfInitialize
- DdfListItem
- DdfMetafile
- DdfSetColor
- DdfSetFont
- DdfSetFontStyle
- DdfSetFormat
- DdfSetTextAlign
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, the example uses DdfPara to start a new paragraph, DdfSetFont and DdfSetFontStyle to have the text displayed in a large, bold Courier font, DdfSetColor to change the text color, and DdfText to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the DdfInitialize DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_DDF                /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
   HWND   hwndParent;
   HWND   hwndInstance;    /* help instance window                */
   HDDF   hDdf;            /* DDF handle                          */
```

```
switch( ulMsg )
{
case HM_QUERY_DDF_DATA:
    /* get the help instance */
    hwndParent = WinQueryWindow( hwnd, QW_PARENT );
    hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
    hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                              MPFROMSHORT( HMQW_INSTANCE ), NULL );

    /* Allocate 1K Buffer (default)  */
    hDdf = DdfInitialize(
                hwndInstance,  /* Handle of help instance */
                0L,            /* Default buffer size     */
                0L             /* Default increment       */
                );

    if (hDdf == NULLHANDLE)       /* Check return code       */
    {
        return (MRESULT)FALSE;
    }

    /* create paragraph in DDF buffer */
    if( !DdfPara( hDdf ) )
    {
       return (MRESULT)FALSE;
    }

    /* Change to large (100 x 100 dimensions) Courier font */
    if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
    {
       return (MRESULT)FALSE;
    }

    /* make the font BOLDFACE */
    if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
    {
       return (MRESULT)FALSE;
    }

    /* make the text display as BLUE on a PALE GRAY background */
    if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
    {
       return (MRESULT)FALSE;
    }
```

```
        /* Write data into the buffer */
        if (!DdfText(hDdf, "Sample Text"))
        {
           return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}
```

DdfSetColor

This function sets the background and foreground colors of the displayed text.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfSetColor (HDDF hddf, COLOR BackColor, COLOR ForColor)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

BackColor (COLOR) – input
Specifies the desired background color.

ForColor (COLOR) – input
Specifies the desired foreground color.

The following color value constants may be used for the foreground and background colors:

CLR_DEFAULT - used to set IPF default text color
CLR_BLACK
CLR_BLUE
CLR_RED
CLR_PINK
CLR_GREEN
CLR_CYAN
CLR_YELLOW
CLR_BROWN
CLR_DARKGRAY
CLR_DARKBLUE
CLR_DARKRED
CLR_DARKPINK
CLR_DARKGREEN
CLR_DARKCYAN
CLR_PALEGRAY

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)	Not enough memory is available.
HMERR_DDF_BACKCOLOR (0x3003)	The background color is not valid.
HMERR_DDF_FORECOLOR (0x3004)	The foreground color is not valid.

Remarks

There is a 4-byte ESC code overhead in the DDF internal buffer for the foreground color, and a 4-byte overhead for the background color, with this function.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfEndList
- DdfHyperText
- DdfInform
- DdfInitialize
- DdfListItem
- DdfMetafile
- DdfPara
- DdfSetFont
- DdfSetFontStyle
- DdfSetFormat
- DdfSetTextAlign
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, the example uses DdfPara to start a new paragraph, DdfSetFont and DdfSetFontStyle to have the text displayed in a large, bold Courier font, DdfSetColor to change the text color, and DdfText to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_DDF                /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>
```

```
MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
   HWND   hwndParent;
   HWND   hwndInstance;    /* help instance window                  */
   HDDF   hDdf;            /* DDF handle                            */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                    hwndInstance,  /* Handle of help instance */
                    0L,            /* Default buffer size     */
                    0L             /* Default increment       */
                    );

        if (hDdf == NULLHANDLE)       /* Check return code       */
        {
            return (MRESULT)FALSE;
        }

        /* create paragraph in DDF buffer */
        if( !DdfPara( hDdf ) )
        {
           return (MRESULT)FALSE;
        }

        /* Change to large (100 x 100 dimensions) Courier font */
        if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
        {
           return (MRESULT)FALSE;
        }

        /* make the font BOLDFACE */
        if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
        {
           return (MRESULT)FALSE;
        }
```

```
        /* make the text display as BLUE on a PALE GRAY background */
        if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
        {
           return (MRESULT)FALSE;
        }

        /* Write data into the buffer */
        if (!DdfText(hDdf, "Sample Text"))
        {
           return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}
```

DdfSetFont

This function specifies a text font in the DDF buffer.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfSetFont  (HDDF hddf, PSZ pFaceName, ULONG ulWidth,
                  ULONG ulHeight)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

pFaceName (PSZ) – input
Pointer to font name.

This parameter can be specified in two ways:

- An ASCIIZ string specifying the font name.
- NULL or "default" to specify the default font.

ulWidth (ULONG) – input
Font width in points.

A point is approximately 1/72 of an inch.

ulHeight (ULONG) – input
Font height in points.

Note: There is a (3-byte + FaceName length) ESC code overhead in the DDF internal buffer for FaceName. There is a 2-byte ESC code overhead for the width and a 2-byte overhead for the height.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)	Not enough memory is available.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfEndList
- DdfHyperText
- DdfInform
- DdfInitialize
- DdfListItem
- DdfMetafile
- DdfPara
- DdfSetColor
- DdfSetFontStyle
- DdfSetFormat
- DdfSetTextAlign
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, the example uses DdfPara to start a new paragraph, DdfSetFont and DdfSetFontStyle to have the text displayed in a large, bold Courier font, DdfSetColor to change the text color, and DdfText to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_DDF                /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
   HWND   hwndParent;
   HWND   hwndInstance;    /* help instance window                */
   HDDF   hDdf;            /* DDF handle                          */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                MPFROMSHORT( HMQW_INSTANCE ), NULL );
```

```
        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                    hwndInstance,  /* Handle of help instance */
                    0L,            /* Default buffer size     */
                    0L             /* Default increment       */
                    );

        if (hDdf == NULLHANDLE)       /* Check return code       */
        {
            return (MRESULT)FALSE;
        }

        /* create paragraph in DDF buffer */
        if( !DdfPara( hDdf ) )
        {
           return (MRESULT)FALSE;
        }

        /* Change to large (100 x 100 dimensions) Courier font */
        if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
        {
           return (MRESULT)FALSE;
        }

        /* make the font BOLDFACE */
        if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
        {
           return (MRESULT)FALSE;
        }

        /* make the text display as BLUE on a PALE GRAY background */
        if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
        {
           return (MRESULT)FALSE;
        }

        /* Write data into the buffer */
        if (!DdfText(hDdf, "Sample Text"))
        {
           return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}
```

DdfSetFontStyle

This function specifies a text font style in the DDF buffer.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfSetFontStyle  (HDDF hddf, ULONG fFontStyle)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

fFontStyle (ULONG) – input
Font style flag.

A NULL value for this parameter will set the font-style back to the default. Any of the following values can be specified:

FM_SEL_ITALIC
FM_SEL_BOLD
FM_SEL_UNDERSCORE

These values can be "ORed" together to combine different font styles.

Note: There is a 4-byte ESC code overhead in the DDF internal buffer for *fFontStyle*.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)	Not enough memory is available.
HMERR_DDF_FONTSTYLE (0x3005)	The font style is not valid.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfEndList
- DdfHyperText
- DdfInform

- DdfInitialize
- DdfListItem
- DdfMetafile
- DdfPara
- DdfSetColor
- DdfSetFont
- DdfSetFormat
- DdfSetTextAlign
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, the example uses DdfPara to start a new paragraph, DdfSetFont and DdfSetFontStyle to have the text displayed in a large, bold Courier font, DdfSetColor to change the text color, and DdfText to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_DDF                /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
   HWND   hwndParent;
   HWND   hwndInstance;    /* help instance window                 */
   HDDF   hDdf;            /* DDF handle                           */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                    hwndInstance,  /* Handle of help instance */
                    0L,            /* Default buffer size     */
                    0L             /* Default increment       */
                    );
```

```
        if (hDdf == NULLHANDLE)       /* Check return code       */
        {
            return (MRESULT)FALSE;
        }

        /* create paragraph in DDF buffer */
        if( !DdfPara( hDdf ) )
        {
           return (MRESULT)FALSE;
        }

        /* Change to large (100 x 100 dimensions) Courier font */
        if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
        {
           return (MRESULT)FALSE;
        }

        /* make the font BOLDFACE */
        if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
        {
           return (MRESULT)FALSE;
        }

        /* make the text display as BLUE on a PALE GRAY background */
        if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
        {
           return (MRESULT)FALSE;
        }

        /* Write data into the buffer */
        if (!DdfText(hDdf, "Sample Text"))
        {
           return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}
```

DdfSetFormat

This function is used to turn formatting off or on. It corresponds to the **:lines.** tag.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfSetFormat  (HDDF hddf, ULONG fFormatType)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

fFormatType (ULONG) – input
Formatting-activation flag.

Only the following constants may be used in this parameter:

TRUE	Turn formatting on.
FALSE	Turn formatting off.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)	Not enough memory is available.

Remarks

If formatting is ON, there is a 3-byte ESC code overhead in the DDF internal buffer for this function. Otherwise, there is a 4-byte ESC code overhead.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfEndList
- DdfHyperText
- DdfInform
- DdfInitialize
- DdfListItem

- DdfMetafile
- DdfPara
- DdfSetColor
- DdfSetFont
- DdfSetFontStyle
- DdfSetTextAlign
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, the example uses DdfSetTextAlign to specify left justified test in the DDF buffer when formatting is OFF. The example then uses DdfSetFormat to turn off formatting for text in the DDF buffer (corresponds to the IPF lines tag). For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_GPIPRIMITIVES      /* Drawing Primitives/Attributes*/
#define INCL_DDF                /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
   HWND   hwndParent;
   HWND   hwndInstance;    /* help instance window                */
   HDDF   hDdf;            /* DDF handle                          */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                    hwndInstance,  /* Handle of help instance */
                    0L,            /* Default buffer size     */
                    0L             /* Default increment       */
                    );
```

```
        if (hDdf == NULLHANDLE)       /* Check return code       */
        {
           return (MRESULT)FALSE;
        }

        /* left justify text when formatting is OFF */
        if (!DdfSetTextAlign(hDdf, TA_LEFT))
        {
           return (MRESULT)FALSE;
        }

        /* turn formatting OFF */
        if (!DdfSetFormat(hDdf, FALSE))
        {
           return (MRESULT)FALSE;
        }

        if (!DdfText(hDdf,
              "Format OFF: This text should be Left Aligned!\n"))
        {
           return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
     }
     return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}
```

DdfSetTextAlign

This function defines whether left, center, or right text justification is to be used when text formatting is off.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfSetTextAlign  (HDDF hddf, ULONG fAlign)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

fAlign (ULONG) – input
Justification flag.

Only the following constants may be used:

TA_LEFT	Left-justify text.
TA_RIGHT	Right-justify text.
TA_CENTER	Center text.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

HMERR_DDF_ALIGN_TYPE (0x3002)	The alignment type is not valid.

Remarks

It should be called before DdfSetFormat is called to turn off text formatting, and should not be called again until formatting is turned back on. Note that leading and trailing spaces are not stripped from the text as a result of this alignment.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfEndList
- DdfHyperText

- DdfInform
- DdfInitialize
- DdfListItem
- DdfMetafile
- DdfPara
- DdfSetColor
- DdfSetFont
- DdfSetFontStyle
- DdfSetFormat
- DdfText

Example Code

After initializing a DDF buffer with DdfInitialize, the example uses DdfSetTextAlign to specify left justified test in the DDF buffer when formatting is OFF. The example then uses DdfSetFormat to turn off formatting for text in the DDF buffer (corresponds to the IPF lines tag). For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_GPIPRIMITIVES      /* Drawing Primitives/Attributes*/
#define INCL_DDF                /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
   HWND   hwndParent;
   HWND   hwndInstance;    /* help instance window                */
   HDDF   hDdf;            /* DDF handle                          */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                MPFROMSHORT( HMQW_INSTANCE ), NULL );
```

```
        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                    hwndInstance,  /* Handle of help instance */
                    0L,            /* Default buffer size     */
                    0L             /* Default increment       */
                    );

        if (hDdf == NULLHANDLE)       /* Check return code       */
        {
           return (MRESULT)FALSE;
        }

        /* left justify text when formatting is OFF */
        if (!DdfSetTextAlign(hDdf, TA_LEFT))
        {
           return (MRESULT)FALSE;
        }

        /* turn formatting OFF */
        if (!DdfSetFormat(hDdf, FALSE))
        {
           return (MRESULT)FALSE;
        }

        if (!DdfText(hDdf,
              "Format OFF: This text should be Left Aligned!\n"))
        {
           return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
     }
     return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}
```

DdfText

This function adds text to the DDF buffer.

Syntax

```
#define INCL_DDF
#include <os2.h>
BOOL DdfText (HDDF hddf, PSZ pText)
```

Parameters

hddf (HDDF) – input
Handle to DDF returned by DdfInitialize.

pText (PSZ) – input
Pointer to the text buffer to be formatted.

Note: There is a 3-byte ESC code overhead in the DDF internal buffer for each word in the text buffer. There is a 1-byte ESC code overhead for each blank and for each newline character.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Related Functions

- DdfBeginList
- DdfBitmap
- DdfEndList
- DdfHyperText
- DdfInform
- DdfInitialize
- DdfListItem
- DdfMetafile
- DdfPara
- DdfSetColor
- DdfSetFont
- DdfSetFontStyle
- DdfSetFormat
- DdfSetTextAlign

Example Code

After initializing a DDF buffer with DdfInitialize, the example uses DdfPara to start a new paragraph, DdfSetFont and DdfSetFontStyle to have the text displayed in a large, bold Courier font, DdfSetColor to change the text color, and DdfText to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the DdfInitialize sample.

```
#define INCL_WINWINDOWMGR       /* General window management    */
#define INCL_WINMESSAGEMGR      /* Message management           */
#define INCL_DDF                /* Dynamic Data Facility        */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
   HWND   hwndParent;
   HWND   hwndInstance;    /* help instance window                 */
   HDDF   hDdf;            /* DDF handle                           */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default)  */
        hDdf = DdfInitialize(
                    hwndInstance,  /* Handle of help instance */
                    0L,            /* Default buffer size     */
                    0L             /* Default increment       */
                    );

        if (hDdf == NULLHANDLE)       /* Check return code       */
        {
            return (MRESULT)FALSE;
        }

        /* create paragraph in DDF buffer */
        if( !DdfPara( hDdf ) )
        {
           return (MRESULT)FALSE;
        }
```

```
        /* Change to large (100 x 100 dimensions) Courier font */
        if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
        {
           return (MRESULT)FALSE;
        }

        /* make the font BOLDFACE */
        if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
        {
           return (MRESULT)FALSE;
        }

        /* make the text display as BLUE on a PALE GRAY background */
        if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
        {
           return (MRESULT)FALSE;
        }

        /* Write data into the buffer */
        if (!DdfText(hDdf, "Sample Text"))
        {
           return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}
```

Chapter 5. Profile Functions

The following table shows all the Profile (Prf) functions in alphabetic order.

C Name
PrfCloseProfile
PrfOpenProfile
PrfQueryProfile
PrfQueryProfileData
PrfQueryProfileInt
PrfQueryProfileSize
PrfQueryProfileString
PrfReset
PrfWriteProfileData
PrfWriteProfileString

PrfCloseProfile

This function indicates that a profile is no longer available for use.

Syntax

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL PrfCloseProfile (HINI hini)
```

Parameters

hini (HINI) – input
Initialization-file handle.

After this function, the handle is no longer valid.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INI_FILE_IS_SYS_OR_USER (0x1124)	User or system initialization file cannot be closed.
PMERR_INVALID_INI_FILE_HANDLE (0x1115)	An invalid initialization-file handle was specified.

Remarks

This function cannot be used to close the current user or system initialization files.

Related Functions

- PrfOpenProfile

Example Code

This example calls PrfCloseProfile to close a profile and makes it unavailable for use.

```
#define INCL_WINSHELLDATA       /* Window Shell functions       */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
HINI  hini;             /* initialization-file handle           */

fSuccess = PrfCloseProfile(hini);
```

PrfOpenProfile

This function indicates that a file is available for use as a profile.

Syntax

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HINI PrfOpenProfile  (HAB hab, PSZ pszFileName)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pszFileName (PSZ) – input
User-profile file name.

This must not be the same as the current user (OS2.INI) or system initialization (OS2SYS.INI) file name.

Returns

hini (HINI) – returns
Initialization-file handle.

This handle is used on other calls to manipulate the profile file.

NULLHANDLE	Error occurred
Other	Initialization-file handle.

Possible returns from WinGetLastError

PMERR_OPENING_INI_FILE (0x1301)	Unable to open initialization file (due to lack of disk space for example).
PMERR_MEMORY_ALLOC (0x1309)	An error occurred during memory management.
PMERR_INI_FILE_IS_SYS_OR_USER (0x1124)	User or system initialization file cannot be closed.

Remarks

A user profile and a system profile are opened by the system, either at start-up time, or (in the case of the user profile) as a result of a PrfReset function, and are always available. Their handles are HINI_USERPROFILE and HINI_SYSTEMPROFILE. Applications do not have to open or close the user profile or the system profile.

The handle returned is only valid for the process issuing the PrfOpenProfile function.

The PrfOpenProfile function can be used by an administrator's application that is creating or modifying a profile for a user.

It can also be used to create a back-up profile as follows:

- Use the enumerate form of PrfQueryProfileData to obtain a list of application names in the profile being backed up.
- Use the enumerate form of PrfQueryProfileData to obtain a list of key names for each of the application names.
- Use PrfQueryProfileData for each application-name or key-name pair to read the appropriate data.
- Use PrfWriteProfileData to write the data into the back-up profile.

Related Functions

- PrfCloseProfile
- PrfQueryProfileData

Example Code

This example uses PrfOpenProfile to open and make available a profile for the file "PROFILE.INI."

```
#define INCL_WINSHELLDATA       /* Window Shell functions       */
#include <os2.h>

HINI  hini;             /* initialization-file handle           */
HAB   hab;              /* anchor-block handle                  */
char  pszFileName[13];  /* user-profile file name               */

strcpy(pszFileName,"PROFILE.INI");

hini = PrfOpenProfile(hab,pszFileName);
```

PrfQueryProfile

This function returns a description of the current user and system profiles.

Syntax

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL PrfQueryProfile (HAB hab, PPRFPROFILE pprfproProfile)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pprfproProfile (PPRFPROFILE) – in/out
Profile names structure.

The *cchUserName* and the *cchSysLen* parameters of the PRFPROFILE data structure are set to the lengths of the respective file names, even if truncation occurs. If these fields are initialized to 0 by the application, then the *pszUserName* and *pszSysName* parameters are not inspected, and the application can then determine the sizes of the buffers required to hold the names on a second call. Otherwise, the *pszUserName* and *pszSysName* parameters must point to reserved areas of memory, and the *cchUserName* and *cchSysLen* parameters must indicate the sizes of those areas.

If the *pszUserName* or the *pszSysName* parameter is NULL, then there is no defined user or system profile, respectively.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or there was insufficient space to record the names, which have been truncated.

Related Functions

- PrfReset

Example Code

This example calls PrfQueryProfile to obtain a description of the current user and system profiles, in this case querying the lengths of the user and system profile file names and placing the values in variables.

```
#define INCL_WINSHELLDATA       /* Window Shell functions       */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
HAB   hab;              /* anchor-block handle                  */
PRFPROFILE  pprfproProfile; /* Profile names structure          */
ULONG ulUserNameLen;    /* length of user file name             */
ULONG ulSysNameLen;     /* length of system file name           */

/* initialize lengths so that query will return the buffer sizes*/
pprfproProfile.cchUserName = 0L;
pprfproProfile.cchSysName = 0L;

fSuccess = PrfQueryProfile(hab, &pprfproProfile);

if (fSuccess == TRUE)
   {
   ulUserNameLen = pprfproProfile.cchUserName;
   ulSysNameLen  = pprfproProfile.cchSysName;
   }
```

PrfQueryProfileData

This function returns a string of binary data from the specified profile.

Syntax

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL PrfQueryProfileData (HINI hini, PSZ pszApp, PSZ pszKey, PVOID pBuffer,
                          PULONG pulBufferMax)
```

Parameters

hini (HINI) – input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle.

pszApp (PSZ) – input
Application name.

The name of the application for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

If this parameter is NULL, this function enumerates all the application names present in the profile and returns the names as a list in the *pBuffer* parameter. Each application name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this case, the *pulBufferMax* parameter contains the total length of the list excluding the final NULL character.

pszKey (PSZ) – input
Key name.

The name of the key for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

If this parameter is NULL, and if *pszApp* is not equal to NULL, this call enumerates all key names associated with the named application and returns the key names, but not their values, as a list in the *pBuffer* parameter. Each key name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this case, the *pulBufferMax* parameter contains the total length of the list **excluding** the final NULL character.

pBuffer (PVOID) – output
Value data.

A buffer in which the value corresponding to the key name is returned. The returned data is not null terminated, unless the value data is explicitly null terminated within the file. This function handles binary data.

pulBufferMax (PULONG) – in/out
Size of value data.

This is the size of the buffer specified by the *pBuffer* parameter. If the call is successful, this is overwritten with the number of bytes copied into the buffer.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARM (0x1303)	A parameter to the function contained invalid data.
PMERR_NOT_IN_IDX (0x1304)	The application name, key-name or program handle was not found.
PMERR_CAN_NOT_CALL_SPOOLER (0x130D)	An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.

Remarks

This function returns a string of binary data from the profile. The call searches the file for a key matching the name specified by the *pszKey* parameter, under the application heading specified by the *pszApp* parameter.

Enumeration can be performed in exactly the same way as in the PrfQueryProfileString function. The enumeration returns application or key names irrespective of whether the data concerned is written with the PrfWriteProfileString function or the PrfWriteProfileData function.

This function returns data that is written to the file using either the PrfWriteProfileString function or the PrfWriteProfileData function.

If the *pszApp* parameter is NULL, this call enumerates all application names and constructs in the *pBuffer* parameter a list of application names. Each application name in the list is terminated with a null character. The last string in the list is terminated with two null characters. This function returns the length of the list, up to, but not including, the final null. If the enumerated application names exceed the available buffer space, the enumerated

names are truncated, the enumerated list is not terminated with 2 bytes of zeros, and the *rc* parameter is set to FALSE. In this case, *pszKey* is ignored.

If the *pszApp* parameter is valid and if the *pszKey* is NULL, this function enumerates all key names associated with the *pszApp* parameter by constructing in the *pBuffer* parameter a list of key names. Each key name in the list is terminated with a null character. The last string in the list is terminated with two null characters. This function returns the length of the list, up to, but not including, the final null. If the enumerated key names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with 2 bytes of zeros, and the *rc* parameter is set to FALSE.

Related Functions

- PrfQueryProfileSize
- PrfWriteProfileData

Example Code

This example calls PrfQueryProfileData to search the user and system profiles for the value of key "KEY" within the application "APP" and return the value if found.

```
#define INCL_WINSHELLDATA       /* Window Shell functions       */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
HINI  hini;             /* initialization-file handle           */
char  pszApp[10];       /* application name                     */
char  pszKey[10];       /* key name                             */
VOID  *pBuffer;         /* Value data                           */
ULONG pulBufferMax;     /* Size of value data                   */

/* Both the user profile and system profile are searched */
hini = HINI_PROFILE;

/* specify application and key names */
strcpy(pszApp,"APP");
strcpy(pszKey,"KEY");

fSuccess = PrfQueryProfileData(hini, pszApp, pszKey, pBuffer,
                               &pulBufferMax);
```

PrfQueryProfileInt

This function returns an integer value from the specified profile.

Syntax

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG PrfQueryProfileInt (HINI hini, PSZ pszApp, PSZ pszKey, LONG lDefault)
```

Parameters

hini (HINI) – input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle returned by PrfOpenProfile.

pszApp (PSZ) – input
Application name.

The name of the application for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

pszKey (PSZ) – input
Key name.

The name of the key for which the profile data is required. The name must match exactly with the name stored in the profile. There is no case-independent searching.

lDefault (LONG) – input
Default value.

This value is returned in *lResult*, if the key defined by *pszKey* cannot be found in the initialization file.

Returns

lResult (LONG) – returns
Key value specified in the initialization file.

The value of the key specified by *pszKey* in the initialization file.

If the value corresponding to the key is not an integer, *lResult* is 0.

If the key-name value is a series of digits followed by non-numeric characters, *lResult* contains the value of the digits only. For example, "KeyName=102abc" causes the value 102 to appear in *lResult*.

Possible returns from WinGetLastError

PMERR_INVALID_PARM (0x1303)	A parameter to the function contained invalid data.
PMERR_NOT_IN_IDX (0x1304)	The application name, key-name or program handle was not found.
PMERR_CAN_NOT_CALL_SPOOLER (0x130D)	An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.

Remarks

This function returns an integer value from the profile. The call searches the file for a key matching the name specified by the *pszKey* parameter, under the application heading specified by the *pszApp* parameter. When an integer is stored as a text string using the PrfWriteProfileString function, for example, "123," the returned value is the number, 123. The call returns *lDefault* if the application-name or key-name pair cannot be found.

Note: The search is case-dependent.

Related Functions

- PrfQueryProfileData
- PrfWriteProfileString

Example Code

This example calls to search the user and system profiles for the integer value of key "KEY" within the application "APP" and return the value if found; if not found, 0 is returned.

```
#define INCL_WINSHELLDATA       /* Window Shell functions       */
#include <os2.h>

LONG  lResult;          /* key value                            */
HINI  hini;             /* initialization-file handle           */
char  pszApp[10];       /* application name                     */
char  pszKey[10];       /* key name                             */
LONG  lDefault;         /* default return value                 */

/* Both the user profile and system profile are searched */
hini = HINI_PROFILE;

/* specify application and key names */
strcpy(pszApp,"APP");
strcpy(pszKey,"KEY");

/* set default to 0 */
lDefault = 0;

lResult = PrfQueryProfileInt(hini, pszApp, pszKey, lDefault);
```

PrfQueryProfileSize

This function obtains the size in bytes of the value of a specified key for a specified application in the profile.

Syntax

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL PrfQueryProfileSize (HINI hini, PSZ pszApp, PSZ pszKey,
                          PULONG pDataLen)
```

Parameters

hini (HINI) – input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle returned by PrfOpenProfile.

pszApp (PSZ) – input
Application name.

The name of the application for which the profile data is required.

If the *pszApp* parameter is NULL, then the *pDataLen* parameter returns the length of the buffer required to hold the enumerated list of application names, as returned by the PrfQueryProfileString function when its *pszApp* parameter is NULL. In this case, the *pszKey* parameter is ignored.

pszKey (PSZ) – input
Key name.

The name of the key for which the size of the data is to be returned.

If the *pszKey* parameter is NULL, and if the *pszApp* parameter is not NULL, the *pDataLen* returns the length of the buffer required to hold the enumerated list of key names for that application name, as returned by the PrfQueryProfileString function when its *pszKey* parameter is NULL, and its *pszApp* parameter is not NULL.

pDataLen (PULONG) – output
Data length.

This parameter is the length of the value data related to the *pszKey* parameter. If an error occurs, this parameter is undefined.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARM (0x1303)	A parameter to the function contained invalid data.
PMERR_NOT_IN_IDX (0x1304)	The application name, key-name or program handle was not found.
PMERR_CAN_NOT_CALL_SPOOLER (0x130D)	An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.

Remarks

The *pszApp* parameter and *pszKey* parameter are case sensitive and must match the names stored in the file exactly. There is no case-independent searching.

This function can be used before using the PrfQueryProfileString call or the PrfQueryProfileData call, to allocate space for the returned data.

No distinction is made between data that is written using the PrfWriteProfileData function and the PrfWriteProfileString function.

Related Functions

- PrfQueryProfileData
- PrfQueryProfileString

Example Code

This example calls PrfQueryProfileSize to search the user and system profiles for the value of key "KEY" within the application "APP" and return the byte size of the value if found.

```
#define INCL_WINSHELLDATA       /* Window Shell functions       */
#include <os2.h>

BOOL  fSuccess;          /* success indicator                   */
HINI  hini;              /* initialization-file handle          */
char  pszApp[10];        /* application name                    */
char  pszKey[10];        /* key name                            */
ULONG pDataLen;          /* data length                         */

/* Both the user profile and system profile are searched */
hini = HINI_PROFILE;

/* specify application and key names */
strcpy(pszApp,"APP");
strcpy(pszKey,"KEY");

fSuccess = PrfQueryProfileSize(hini, pszApp, pszKey, &pDataLen);
```

PrfQueryProfileString

This function retrieves a string from the specified profile.

Syntax

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG PrfQueryProfileString (HINI hini, PSZ pszApp, PSZ pszKey,
                             PSZ pszDefault, PVOID pBuffer,
                             ULONG cchBufferMax)
```

Parameters

hini (HINI) – input
Initialization-file handle.

HINI_PROFILE	Both the user profile and system profile are searched
HINI_USERPROFILE	The user profile is searched
HINI_SYSTEMPROFILE	The system profile is searched
Other	Initialization-file handle returned by the PrfOpenProfile function.

pszApp (PSZ) – input
Application name.

The name of the application for which the profile data is required.

The search performed on the application name is always case-dependent. Names starting with the characters "PM_" are reserved for system use.

If this parameter is NULL, this function enumerates all the application names present in the profile and returns the names as a list in the *pBuffer* parameter. Each application name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this instance, the *pulLength* parameter contains the total length of the list **excluding** the final NULL character.

pszKey (PSZ) – input
Key name.

The name of the key for which the profile data is returned.

The search on key name is always case-dependent.

If this parameter equals NULL, and if the *pszApp* parameter is not equal to NULL, this function enumerates all key names associated with the named application and returns the key names (not their values) as a list in the *pBuffer* parameter. Each key name is terminated with a NULL character and the last name is terminated with two successive NULL characters. In this instance, the *pulLength* parameter contains the total length of the list **excluding** the final NULL character.

pszDefault (PSZ) – input
Default string.

The string that is returned in the *pBuffer* parameter, if the key defined by the *pszKey* parameter cannot be found in the profile.

If the pointer to this parameter is passed as NULL, then nothing is copied into the *pszKey* parameter if the key cannot be found. *pulLength* is returned as 0 in this case.

pBuffer (PVOID) – output
Profile string.

The text string obtained from the profile for the key defined by the *pszKey* parameter.

cchBufferMax (ULONG) – input
Maximum string length.

The maximum number of characters that can be put into the *pBuffer* parameter, in bytes. If the data from the profile is longer than this, it is truncated.

Returns

pulLength (ULONG) – returns
String length returned.

The actual number of characters (including the null termination character) returned in the *pBuffer* parameter, in bytes.

Possible returns from WinGetLastError

PMERR_INVALID_PARM (0x1303)	A parameter to the function contained invalid data.
PMERR_BUFFER_TOO_SMALL (0x110B)	The supplied buffer was not large enough for the data to be returned.
PMERR_NOT_IN_IDX (0x1304)	The application name, key-name or program handle was not found.
PMERR_CAN_NOT_CALL_SPOOLER (0x130D)	An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.
PMERR_INVALID_ASCIIZ (0x130C)	The profile string is not a valid zero-terminated string.

Remarks

The call searches the profile for a key matching the name specified by the *pszKey* parameter under the application heading specified by the *pszApp* parameter. If the key is found, the corresponding string is copied. If the key does not exist, the default character string, specified by the *pszDefault* parameter, is copied.

If the enumerated application names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with 2 bytes of zeros, and the

pulLength parameter is set to the number of bytes copied into the *pBuffer* parameter. In this instance, the *pszKey* parameter is ignored.

Note: If the enumeration cannot be performed for any reason, the default character string is *not* copied.

This function returns the length of the list, up to, but not including, the final null. If the enumerated key names exceed the available buffer space, the enumerated names are truncated, the enumerated list is not terminated with 2 bytes of zeros, and the *pulLength* parameter is set to the number of bytes copied into the *pBuffer* parameter.

This function is case-dependent; thus the strings in the *pszApp* parameter and the *pszKey* parameter must match exactly. This avoids any code-page dependency. The application storing the data must do any case-independent matching.

The enumeration call does not distinguish between data written with the PrfWriteProfileString function and the PrfWriteProfileData function.

Related Functions

- PrfWriteProfileString

Example Code

PrfQueryProfileString is issued twice to obtain the names of the default printer, the default presentation driver, and the queue associated with the printer. If any of these requests fails, the default values already defined in DEVOPENSTRUC are used.

```
#define INCL_WINSHELLDATA
#include <OS2.H>
char szTemp[80];
char szBuff[257];
PCH ptscan;

DEVOPENSTRUC dopPrinter = {"LPT1Q",
                           (PSZ)"IBM4201",
                            0L,
                           (PSZ)"PM_Q_STD",
                            0L, 0L, 0L, 0L, 0L};

  if (PrfQueryProfileString(HINI_PROFILE,
                            (PSZ)"PM_SPOOLER",
                            (PSZ)"PRINTER",
                             NULL,
                            (PVOID)szTemp,
                            (LONG)sizeof(szTemp)
                            )){
      szTemp[strlen(szTemp)-1] = 0;
      if (PrfQueryProfileString(HINI_PROFILE,
                                (PSZ)"PM_SPOOLER_PRINTER",
                                (PSZ)szTemp,
                                 NULL,
                                (PVOID)szBuff,
                                (LONG)sizeof(szBuff)
                                )){

          ptscan = (PCH)strchr(szBuff, ';');
          ptscan++;
          ptscan = (PCH)strchr(ptscan, (INT)';');
          ptscan++;
          *(ptscan + strcspn(ptscan, ".,;")) = 0;
          dopPrinter.pszLogAddress =  ptscan;

          ptscan = (PCH)strchr(szBuff, (INT)';');
          ptscan++;
          *(ptscan + strcspn(ptscan, ".,;")) = 0;
          dopPrinter.pszDriverName = ptscan;

      }
  }
```

PrfReset

This function defines which files are to be used as the user and system profiles.

Syntax

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL PrfReset (HAB hab, PRFPROFILE prfproProfile)
```

Parameters

hab (HAB) – input
Anchor-block handle.

prfproProfile (PRFPROFILE) – input
Profile-names structure.

This contains the names of the files to be used as the new Presentation Manager* (PM*) profile files. Any valid file names can be used. A name that is not already fully qualified is taken to refer to the current directory.

If the user profile file does not exist, a new file is created.

The name of the system profile cannot be changed. It must be the name of the current system profile as returned by PrfQueryProfile.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_OPENING_INI_FILE (0x1301)	Unable to open initialization file (due to lack of disk space for example).

Remarks

This function causes the workstation to use different profiles. When the workstation is initialized, the names of the user and system profiles are taken from the PROTSHELL statement specified in CONFIG.SYS. PrfReset allows the profiles to be changed during operation of the workstation, for example by a logon application controlling multiple consecutive users of the system.

After the PrfReset function completes, the system has a new set of preferences (for example screen colors), a new start-up list, and new spooler parameters.

The PrfReset function broadcasts the PL_ALTERED message, which must be processed by all applications that read their default settings from the user or system profiles.

Note: This will only change the default system values in the ini file. It is up to the applications to read the new default settings and reset them to their new values.

For example, consider logon applications. On receipt of a PL_ALTERED message, they should carry out the following:

- Read the new color settings from the new profiles, and set the new screen colors (and palettes) which should be refreshed.
- Set the country information, for example the date and time format, which is read from the new profiles.
- Other preferences, for example, those that affect the operations of the alarm and the mouse, should also update with the new settings held in the new profiles.

This function requires the existence of a message queue.

Related Functions

- PrfQueryProfile

Related Messages

- PL_ALTERED

Example Code

This function defines which files are to be used as the user and system profiles.

```
#define INCL_WINSHELLDATA
#include <OS2.H>
HAB hab;
char userpro[] = "profile.ini";
PRFPROFILE profile;

PrfQueryProfile(hab, &profile); /* get the system profile name */

profile.pszUserName = userpro;
profile.cchUserName = sizeof(userpro);

PrfReset (hab, &profile);
```

PrfWriteProfileData

This function writes a string of binary data into the specified profile.

Syntax

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL PrfWriteProfileData (HINI hini, PSZ pszApp, PSZ pszKey, PVOID pData,
                          ULONG cchDataLen)
```

Parameters

hini (HINI) – input
Initialization-file handle.

HINI_PROFILE	User profile
HINI_USERPROFILE	User profile
HINI_SYSTEMPROFILE	System profile
Other	Initialization-file handle returned by PrfOpenProfile.

pszApp (PSZ) – input
Application name.

The case-dependent name of the application for which profile data is to be written. Names starting with the characters "PM_" are reserved for system use.

pszKey (PSZ) – input
Key name.

The case-dependent name of the key for which profile data is to be written.

This parameter can be NULL in which case *all* the *pszKey* or *pData* pairs associated with *pszApp* are deleted.

pData (PVOID) – input
Value data.

This is the value of the *pszKey* or *pData* pair that is written to the profile. It is *not* zero-terminated, and its length is given by the *cchDataLen* parameter.

If this parameter is NULL, the string associated with the *pszKey* parameter is deleted.

cchDataLen (ULONG) – input
Size of value data.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARM (0x1303)	A parameter to the function contained invalid data.
PMERR_CAN_NOT_CALL_SPOOLER (0x130D)	An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.

Remarks

Because of the binary nature of the data, the input data is not zero-terminated. The length provided is the only way to identify the length of the data.

Related Functions

- PrfQueryProfileSize

Example Code

This function deletes the profile data associated with application sample.exe

```
#define INCL_WINSHELLDATA
#include <OS2.H>
HAB hab;


PrfWriteProfileData(HINI_USERPROFILE,
                    "sample",        /* application. */
                    NULL,
                    NULL,
                    0L);
```

PrfWriteProfileString

This function writes a string of character data into the specified profile.

Syntax

```
#define INCL_WINSHELLDATA /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL PrfWriteProfileString (HINI hini, PSZ pszApp, PSZ pszKey, PSZ pszData)
```

Parameters

hini (HINI) – input
Initialization-file handle.

HINI_PROFILE	User profile
HINI_USERPROFILE	User profile
HINI_SYSTEMPROFILE	System profile
Other	Initialization-file handle returned by PrfOpenProfile.

pszApp (PSZ) – input
Application name.

The case-dependent name of the application for which profile data is to be written. Names starting with the characters "PM_" are reserved for system use.

pszKey (PSZ) – input
Key name.

The case-dependent name of the key for which profile data is to be written.

This parameter can be NULL, in which case *all* the *pszKey* or *pszData* pairs associated with the *pszApp* parameter are deleted.

pszData (PSZ) – input
Text string.

This is the value of the *pszKey* or *pszData* pair that is written to the profile.

If this parameter is NULL, the string associated with the *pszKey* is deleted (that is, the entry is deleted).

If this parameter is not NULL, the string is used as the value of the *pszKey* or *pszData* pair, even if the string has zero length.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARM (0x1303)	A parameter to the function contained invalid data.
PMERR_CAN_NOT_CALL_SPOOLER (0x130D)	An error occurred attempting to call the spooler validation routine. This error is not raised if the spooler is not installed.

Remarks

If there is no application field in the file that matches the *pszApp*, a new application field is created before the *pszKey* or *pszData* entry is made.

If the key name does not exist for the application, a new *pszKey* or *pszData* entry is created for that application. If the *pszKey* already exists in the file, the existing value is overwritten.

Related Functions

- PrfQueryProfileString

Example Code

This function deletes the profile string associated with application sample.exe

```
#define INCL_WINSHELLDATA
#include <OS2.H>
HAB hab;


PrfWriteProfileString(HINI_USERPROFILE,
                      "sample",         /* application. */
                      NULL,
                      NULL);
```

Chapter 6. Spooler Functions

The following table shows how all of the Spooler functions are related within functional areas. The functions are in alphabetic order within these areas.

C Name	C Name
Control	
SplControlDevice	SplEnumQueueProcessor
SplCopyJob	SplHoldJob
SplCreateDevice	SplHoldQueue
SplCreateQueue	SplMessageBox
SplDeleteDevice	SplPurgeQueue
SplDeleteJob	SplQueryDevice
SplDeleteQueue	SplQueryJob
SplEnumDevice	SplQueryQueue
SplEnumDriver	SplReleaseJob
SplEnumJob	SplReleaseQueue
SplEnumPort	SplSetDevice
SplEnumPrinter	SplSetJob
SplEnumQueue	SplSetQueue
Job Submission	
SplQmAbort	SplQmOpen
SplQmAbortDoc	SplQmStartDoc
SplQmClose	SplQmWrite
SplQmEndDoc	

SplControlDevice

This function cancels, holds, continues, or restarts a print device.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplControlDevice  (PSZ pszComputerName, PSZ pszPortName,
                          ULONG ulControl)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where print device is to be controlled.

A NULL string specifies the local workstation.

pszPortName (PSZ) – input
Port name.

ulControl (ULONG) – input
Operation to perform.

PRD_DELETE	Delete current print job
PRD_PAUSE	Pause printing
PRD_CONT	Continue paused print job
PRD_RESTART	Restart print job.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_DestNotFound (2152)	The print device cannot be found.
NERR_DestIdle (2158)	This print device is idle and cannot accept control operations.
NERR_DestInvalidOp (2159)	This print device request contains an invalid control function.
NERR_ProcNoRespond (2160)	The queue processor is not responding.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

A paused print device cannot accept new print jobs.

If PRD_DELETE is attempted when there is no current print job, NERR_DestIdle (2158) is returned.

To control jobs on a remote server requires administrator privilege.

Related Functions

- SplEnumDevice
- SplQueryDevice

Example Code

This sample code demonstrates the result of various actions that can be performed on the print device by this function call. At the command line, a print device name is entered along with an action code.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>       /* for printf function */

INT main (argc, argv)
   INT argc;
   CHAR *argv[];
{
   SPLERR splerr ;
   ULONG  ulControl=0L ;
   PSZ    pszComputerName = NULL ;
   PSZ    pszPrintDeviceName ;

   /* Input a Print Device Name and an Action Code on the command line       */
   if (argc != 3)
   {
      printf("Syntax is:  qcontrol  PrintDeviceName  ActionCode \n");
      printf("Action codes are: D-Delete, P-Pause, C-Continue, R-Restart\n\n");
      DosExit( EXIT_PROCESS , 0 ) ;
   }

   /* Get the print device name from the first input parameter.              */
   pszPrintDeviceName = argv[1];
```

```
   /* Get the action code from the second input parameter.                */
   switch (argv[2][0])
   {
      case 'D':
        ulControl = PRD_DELETE ;
        break;
      case 'P':
        ulControl = PRD_PAUSE ;
        break;
      case 'C':
        ulControl = PRD_CONT ;
        break;
      case 'R':
        ulControl = PRD_RESTART ;
        break;
      default:
        printf("Invalid code\n");
        DosExit( EXIT_PROCESS , 0 ) ;
   }
   /* Call the function with the parameters obtained from the command line.  */
   splerr = SplControlDevice(pszComputerName, pszPrintDeviceName, ulControl);

   /* If there is an error returned, print it.                           */
   if (splerr != 0L)
   {
      switch (splerr)
      {
         case  NERR_DestNotFound :
            printf("Destination does not exist.\n");
            break;
         case  NERR_DestIdle:
            printf("This print device is idle - can't do control ops. \n");
            break;
         default:
            printf("Errorcode = %ld\n",splerr);
         }
   } else {
        printf("The print job operation was performed.\n\n");
   }
   DosExit( EXIT_PROCESS , 0 ) ;
   return (splerr) ;
}
```

SplCopyJob

This function copies a job in a print queue.

Currently there is a restriction that a job can only be copied onto the same queue (and computer) as the original job.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplCopyJob (PSZ pszSrcComputerName, PSZ pszSrcQueueName,
                   ULONG ulSrcJob, PSZ pszTrgComputerName,
                   PSZ pszTrgQueueName, PULONG pulTrgJob)
```

Parameters

pszSrcComputerName (PSZ) – input
Name of computer where job is to be copied from.

A NULL string specifies the local workstation.

pszSrcQueueName (PSZ) – input
Name of queue where job is to be copied from.

ulSrcJob (ULONG) – input
Source Job identification number.

pszTrgComputerName (PSZ) – input
Name of computer where job is to be copied to.

A NULL string specifies the local workstation.

pszTrgQueueName (PSZ) – input
Name of queue where job is to be copied to.

A NULL string specifies the same queue as the original job.

pulTrgJob (PULONG) – output
Job identification number of new job.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.

ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_JobNotFound (2151)	The print job does not exist.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_InvalidComputer (2351)	The computer name is invalid.

Related Functions

- SplEnumJob
- SplEnumQueue
- SplQueryJob
- SplQueryQueue

Example Code

This sample code will make a duplicate copy of the jobid that is entered at the prompt. Presently, there is a restriction that the job can only be duplicated on the same computer/queue; i.e. a local job.

```
#define INCL_SPL
#include <os2.h>
#include <stdio.h>       /* for printf function */
#include <stdlib.h>      /* for atoi function   */

INT main (argc, argv)
   INT argc;
   CHAR *argv[];
{
   SPLERR splerr ;
   ULONG  ulSrcJob, ulTrgJob ;
   PSZ    pszSrcComputerName,pszTrgComputerName ;
   PSZ    pszSrcQueueName,pszTrgQueueName ;

   if (argc != 2)
   {
      printf("Command is:  copyjob JOBID\n");
      DosExit( EXIT_PROCESS , 0 ) ;
   }
   pszSrcComputerName = (PSZ)NULL ;

   /* The only valid values at present for these three parameters is  NULL    */
   pszSrcQueueName = (PSZ)NULL;
   pszTrgComputerName = (PSZ)NULL ;
   pszTrgQueueName = (PSZ)NULL ;
```

```
   /* Convert input parameter to a ULONG                                    */
   ulSrcJob = atoi ( argv[1] );

   if (splerr = SplCopyJob(pszSrcComputerName,pszSrcQueueName,ulSrcJob,
                           pszTrgComputerName,pszTrgQueueName,&ulTrgJob))
   {
      printf("Return code SplCopyJob = %d\n",splerr);
   }
   else
   {
      printf("New job ID is %d\n",ulTrgJob);
   }
   DosExit( EXIT_PROCESS , 0 ) ;
   return (splerr);
}   /* end main */
```

SplCreateDevice

This function establishes a print device on the local workstation or a remote server.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplCreateDevice (PSZ pszComputerName, ULONG ulLevel, PVOID pBuf,
                        ULONG cbBuf)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where print device is to be added.

A NULL string specifies the local workstation.

ulLevel (ULONG) – input
Level of detail provided.

This must be 3.

pBuf (PVOID) – input
Data structure.

It points to a PRDINFO3 data structure.

cbBuf (ULONG) – input
Size, in bytes, of data structure.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_NAME (123)	The computer name is invalid.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_DestExists (2153)	The print device already exists.
NERR_DestNoRoom (2157)	The maximum number of print devices has been reached.

NERR_DestInvalidState (2162)	This operation cannot be performed on the print device.
NERR_SpoolNoMemory (2165)	A spooler memory allocation failure occurred.
NERR_DriverNotFound (2166)	The device driver does not exist.
NERR_BadDev (2341)	The device is already in use as a communications device.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

The result of this function is the creation of a new print device definition.

The printer is set up to print on the logical address (port) defined by *pszLogAddr* in PRDINFO3. If *pszLogAddr* is NULL, the print device definition is created but is not connected to any logical address. In this case no printing can occur on that print device or from any print queue connected only to that print device. If a logical address is specified, it must already be defined in the PM_SPOOLER_PORTS section of the initialization file.

Note: To change the connection between a print device and a port, use SplSetDevice.

The maximum length for a print device name is 32 characters. The use of a longer name results in ERROR_INVALID_NAME (123).

All device drivers and queues specified with the print device must already be defined to the spooler.

To add a remote print device requires administrator privilege.

Related Functions

- SplDeleteDevice
- SplEnumDevice
- SplEnumDriver
- SplEnumPort

Example Code

This sample code creates a PRDINFO3 structure with dummy parameters. This structure is then used to call SplCreateDevice to establish a print device on a local workstation.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT

#include <os2.h>
#include <stdio.h>    /* for printf function */
#include <string.h>   /* for strcpy function */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    ULONG  splerr  ;
    ULONG  cbBuf;
    ULONG  ulLevel ;
    PSZ    pszComputerName ;
    PSZ    pszPrintDeviceName ;
    PRDINFO3 prd3  ;

    if (argc != 2)
    {
       printf("Syntax:  sdcrt  DeviceName \n");
       DosExit( EXIT_PROCESS , 0 ) ;
    }
    /* We are going to create a print device on the local workstation. */
    pszComputerName = (PSZ)NULL ;

    /* Get the name from the command line.                              */
    pszPrintDeviceName = argv[1];

    /* Level 3 is valid. We will use level 3.                           */
    ulLevel = 3;

    /* Get size of buffer needed for a PRDINFO3 structure.              */
    cbBuf = sizeof(PRDINFO3);

    /* Set up the structure with dummy parameters.                      */
    prd3.pszPrinterName = pszPrintDeviceName;
    prd3.pszUserName = NULL;
    prd3.pszLogAddr = "LPT1";
    prd3.uJobId=0;
    prd3.pszComment= "Test comment";
    prd3.pszDrivers = "IBMNULL";
    prd3.usTimeOut = 777;

    /* Make the call. */
    splerr = SplCreateDevice(pszComputerName, ulLevel,
                             &prd3, cbBuf);
```

```
    /* Print out the results.                                          */
    if (splerr == NO_ERROR)
       printf("The device was successfully created.");
    else
       printf("SplCreateDevice Error=%ld, cbNeeded=%ld\n",
                               splerr, cbBuf) ;

    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}
```

SplCreateQueue

This function creates a new print queue on the local workstation or on a remote server. A remote server setup requires the LAN Requester and Server software.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplCreateQueue (PSZ pszComputerName, ULONG ulLevel,
                       PVOID pbBuf, ULONG cbBuf)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where queue is to be created.

A NULL string specifies a local workstation.

ulLevel (ULONG) – input
Level of detail provided.

This must be 3 or 6.

pbBuf (PVOID) – input
Data structure.

It points to a data structure depending on the value specified in *ulLevel* as follows:

ulLevel	**Buffer Content.**
3	a PRQINFO3 data structure
6	a PRQINFO6 data structure

cbBuf (ULONG) – input
Size, in bytes, of data structure.

It must be greater than 0.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_NAME (123)	The computer name is invalid.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
NERR_NetNotStarted (2102)	The network program is not started.

NERR_RedirectedPath (2117)	The operation is invalid on a redirected resource.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_DestNotFound (2152)	The printer destination cannot be found.
NERR_QExists (2154)	The printer queue already exists.
NERR_DestInvalidState (2162)	This operation cannot be performed on the print destination in its current state.
NERR_SpoolNoMemory (2165)	A spooler memory allocation failure occurred.
NERR_DriverNotFound (2166)	The device driver does not exist.
NERR_DataTypeInvalid (2167)	The data type is not supported by the queue processor.
NERR_ProcNotFound (2168)	The queue processor is not installed.
NERR_BadDev (2341)	The requested device is invalid.
NERR_CommDevInUse (2343)	This device is already in use as a communications device.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

To create a queue on a remote server requires administrator privilege. The following fields are required in PRQINFO3 or PRQINFO6:

- *uPriority*
- *uStartTime*
- *uUntilTime*
- *pszSepFile*
- *pszParms*

If a queue of the name specified in *pszName* already exists on *pszComputerName*, the call fails unless the queue is marked for deletion. In this case, the queue is not deleted, and the creation fields are used to perform a SplSetQueue function on the queue.

If *pszPrinters* is NULL, the queue is created but not connected to any printer.

pszDriverName can be a NULL string, in which case *pDriverData*. is ignored. Otherwise, *pszDriverName* must refer to the name of a device driver that is already defined in the initialization file (for example, "IBM4019").

Related Functions

- SplDeleteQueue
- SplEnumDevice
- SplEnumDriver
- SplEnumQueueProcessor

Example Code

This sample code creates a queue on the local workstation. The queue is created with dummy parameters. The name is entered at the command line.

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT

#include <os2.h>
#include <stdio.h>
#include <string.h>

INT main (argc, argv )
    INT argc;
    CHAR *argv[];
{

    ULONG  splerr  ;
    ULONG  cbBuf;
    ULONG  ulLevel ;
    PSZ    pszComputerName ;
    PSZ    pszQueueName ;
    PRQINFO3 prq3  ;

    if (argc != 2)
    {
       printf("Syntax:  sqcrt  QueueName \n");
       DosExit( EXIT_PROCESS , 0 ) ;
    }

    pszComputerName = (PSZ)NULL ;
    ulLevel = 3L;

    /* Get the queue name from the argument entered at    */
    /* the command line.                                  */
    pszQueueName = argv[1];

    /* Determine the size of the needed buffer.           */
    cbBuf = sizeof(PRQINFO3);

    /* Set up the structure with some dummy parameters.   */
    prq3.pszName = pszQueueName;
    prq3.uPriority=5;
    prq3.uStartTime=0;
    prq3.uUntilTime=0;
    prq3.pszSepFile="c:\\os2\\sample.sep";
    prq3.pszParms=NULL;
    prq3.pszPrinters=NULL;
    prq3.pszDriverName=NULL;
    prq3.pDriverData="IBMNULL";      /* Set to Driver.Device name */
```

```
    /* Make the call with the proper parameters.                    */
    splerr = SplCreateQueue(pszComputerName, ulLevel,
                            &prq3, cbBuf);

    /* Print out the error return code and some other information. */
    printf("SplCreateQueue Error=%ld, cbNeeded=%ld\n",
           splerr, cbBuf) ;

    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}
```

SplDeleteDevice

This function deletes a print device.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplDeleteDevice  (PSZ pszComputerName, PSZ pszPrintDeviceName)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where print device is. to be deleted.

A NULL string specifies the local workstation.

pszPrintDeviceName (PSZ) – input
Name of Print Device.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_DestNotFound (2152)	The print device cannot be found.
NERR_DestInvalidState (2162)	This operation cannot be performed on the print device.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

If the print device is currently printing a job, SplDeleteDevice fails and returns NERR_DestInvalidState (2162).

To delete a print device on a remote server requires administrator privilege.

Related Functions

- SplCreateDevice
- SplEnumDevice

Example Code

This sample code will delete the print device whose name is entered at the prompt.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>

INT main (argc, argv)
   INT argc;
   CHAR *argv[];
{
   SPLERR splerr= 0L;
   PSZ    pszComputerName ;
   PSZ    pszPrintDeviceName ;

   /* Check that the parameters were entered at the command line.          */
   if (argc != 2)
   {
      printf("Syntax:  sddel  PrintDeviceName  \n");
      DosExit( EXIT_PROCESS , 0 ) ;
   }

   /* Computer name of NULL indicates the local computer.                  */
   pszComputerName = (PSZ)NULL ;

   /* Set the PrintDeviceName to the value entered at the command line.    */
   pszPrintDeviceName = argv[1];
```

```
   /* Make the call and print out the return code.                    */
   splerr=SplDeleteDevice(pszComputerName, pszPrintDeviceName);
   switch (splerr)
   {
      case  NO_ERROR:
         printf("Print Device %s was deleted.\n",pszPrintDeviceName);
         break;
      case  NERR_DestNotFound :
         printf("Destination does not exist.\n");
         break;
      case  NERR_DestInvalidState:
         printf("This operation can't be performed on the print device.\n");
         break;
      case  NERR_SpoolerNotLoaded:
         printf("The Spooler is not running.\n");
         break;
      default:
         printf("SplDeleteDevice Errorcode = %ld\n",splerr);
   } /* endswitch */
   DosExit( EXIT_PROCESS , 0 ) ;
   return (splerr) ;
}
```

SplDeleteJob

This function deletes a job from a print queue.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplDeleteJob  (PSZ pszComputerName, PSZ pszQueueName,
                      ULONG ulJob)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where job is to be deleted.

A NULL string specifies the local workstation.

pszQueueName (PSZ) – input
Queue Name.

ulJob (ULONG) – input
Job identification number.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_JobNotFound (2151)	The print job does not exist.
NERR_ProcNoRespond (2160)	The queue processor is not responding.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

It is possible to delete a job that is currently printing.

If the print queue on which the print job is submitted is pending deletion (following a SplDeleteQueue call), and the print job is the last in the queue, this function has the additional effect of deleting the queue.

A user with administrator privilege can delete any job.

A job created locally can be deleted locally regardless of user privilege level, but can be deleted remotely only by an administrator.

A remote job can be deleted by a user without administrator privilege only if the username of the person initiating the request is the same as the username of the person who created the job.

Related Functions

- SplCopyJob
- SplEnumJob
- SplQueryJob

Example Code

This sample code will delete the job id that is entered at the prompt.

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>       /* for printf function */
#include <stdlib.h>      /* for atoi  function  */

INT main (argc, argv)
   INT argc;
   CHAR *argv[];
{
   SPLERR splerr ;
   ULONG  ulJob ;
   PSZ    pszComputerName = NULL ;
   PSZ    pszQueueName = NULL ;

   /* Get job id from the input argument. */
   ulJob = atoi(argv[1]);

   /* Call the function to do the delete. If an error is      */
   /*  returned, print it.                                    */
   splerr = SplDeleteJob( pszComputerName, pszQueueName, ulJob);
```

```
   if (splerr != NO_ERROR)
   {
      switch (splerr)
      {
      case  NERR_JobNotFound :
         printf("Job does not exist.\n");
         break;
      case  NERR_JobInvalidState:
         printf("This operation can't be performed on the print job.\n");
         break;
      default:
         printf("Errorcode = %ld\n",splerr);
      } /* endswitch */
   }
   else
   {
      printf("Job %d was deleted.\n",ulJob);
   } /* endif */
   DosExit( EXIT_PROCESS , 0 ) ;
   return (splerr);
}
```

SplDeleteQueue

This function deletes a print queue from the spooler.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplDeleteQueue  (PSZ pszComputerName, PSZ pszQueueName)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where queue is to be deleted.

A NULL string specifies the local workstation.

pszQueueName (PSZ) – input
Queue name.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_QInvalidState (2163)	This operation cannot be performed on the print queue.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

If there are print jobs in the queue, SplDeleteQueue marks the queue PRQ3_PENDING. No further jobs can then be added to the queue, which is deleted when all jobs are printed. A queue marked PRQ3_PENDING can be held, and jobs in the queue can be held, restarted, and repeated.

If a queue is held and there are jobs on the queue, a SplDeleteQueue function fails with NERR_QInvalidState (2163).

To delete a queue on a remote server requires administrator privilege on the remote server.

Related Functions

- SplCreateQueue
- SplEnumQueue
- SplQueryQueue

Example Code

This sample code will delete the queue name that is entered at the prompt.

```
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>       /* for printf function */

INT main (argc, argv)
   INT argc;
   CHAR *argv[];
{
   SPLERR splerr ;
   PSZ    pszComputerName = NULL ;
   PSZ    pszQueueName ;

   /* Get queue name from the input argument  */
   pszQueueName = argv[1];

   /* Call the function to do the delete. If an error is returned, print it.
*/
   splerr=SplDeleteQueue(pszComputerName, pszQueueName);

   if (splerr != 0L)
   {
      switch (splerr)
      {
         case  NERR_QNotFound :
            printf("Queue does not exist.\n");
            break;
         case  NERR_QInvalidState:
            printf("This operation can't be performed on the print queue.\n");
            break;
         default:
            printf("Errorcode = %ld\n",splerr);
      } /* endswitch */
   }
   else
   {
      printf("Queue %s was deleted.\n",pszQueueName);
   } /* endif */
   DosExit( EXIT_PROCESS , 0 ) ;
   return (splerr);
}
```

SplEnumDevice

This function lists print device on a server, optionally supplying status information.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplEnumDevice (PSZ pszComputerName, ULONG ulLevel, PPVOID pBuf,
                      ULONG cbBuf, PULONG pcReturned, PULONG pcTotal,
                      PULONG pcbNeeded, PPVOID pReserved)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where print devices are to. be listed.

A NULL string specifies the local workstation.

ulLevel (ULONG) – input
Level of detail required.

This must be 0, 2 or 3.

pBuf (PPVOID) – output
Buffer.

The returned contents of the buffer depend on the value indicated in *ulLevel* as follows:

ulLevel	**Buffer Contents**
0	An array of port names. Each name consists of 9 characters, including a null terminator.
2	An array of print device names of type PSZ.
3	An array of PRDINFO3 structures.

If no job is printing on the print device, bits 2-11 of *fsStatus* in the PRDINFO3 data structure are meaningless.

cbBuf (ULONG) – input
Size, in bytes, of Buffer.

It must be greater than 0.

pcReturned (PULONG) – output
Number of entries returned.

pcTotal (PULONG) – output
Number of entries available.

pcbNeeded (PULONG) – output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

pReserved (PPVOID) – output
Reserved value, must be NULL.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

If *ulLevel* is set to 0, each port name in *pBuf* is truncated to 8 characters.

Related Functions

- SplCreateDevice
- SplDeleteDevice

Example Code

This sample code enumerates all the devices on the local workstation. It then prints out the information.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>
```

```
INT main ()
{
   ULONG  cbBuf ;
   ULONG  cTotal;
   ULONG  cReturned ;
   ULONG  cbNeeded ;
   ULONG  ulLevel = 3L;
   ULONG  i ;
   SPLERR splerr ;
   PSZ    pszComputerName ;
   PBYTE  pBuf ;
   PPRDINFO3 pprd3 ;

   pszComputerName = (PSZ)NULL ;

   /* Make the call with cBuf = 0 so that you will get the size of the */
   /* buffer  needed returned in cbNeeded.                             */
   splerr = SplEnumDevice(pszComputerName, ulLevel, pBuf, 0L, /* cbBuf */
                          &cReturned, &cTotal, &cbNeeded,
                          NULL) ;

   /* Only continue if the error codes ERROR_MORE_DATA or       */
   /* NERR_BufTooSmall are returned.                              */
   if (splerr == ERROR_MORE_DATA || splerr == NERR_BufTooSmall)
   {
      /* Allocate memory for the buffer that will hold the returning info. */
      if (!DosAllocMem( &pBuf, cbNeeded,
                        PAG_READ|PAG_WRITE|PAG_COMMIT) )
      {
         cbBuf = cbNeeded ;

         /* Make call again with the proper buffer size.                */
         splerr = SplEnumDevice(pszComputerName, ulLevel, pBuf, cbBuf,
                                &cReturned, &cTotal,
                                &cbNeeded, NULL) ;

         /* If no errors, print out the buffer information.             */
         if (splerr == NO_ERROR)
         {
            for (i=0;i < cReturned ; i++)
            {
               /* Each time through the loop increase the pointer.      */
               pprd3 = (PPRDINFO3)pBuf+i ;
               printf("Device info:pszPrinterName   - %s\n",
                           pprd3->pszPrinterName) ;
               printf("  pszUserName - %s\n", pprd3->pszUserName);
               printf("  pszLogAddr  - %s\n", pprd3->pszLogAddr);
```

```
                printf("  uJobId      - %d  fsStatus - %X\n",
                           pprd3->uJobId ,  pprd3->fsStatus);
                printf("  pszStatus   - %s\n", pprd3->pszStatus);
                printf("  pszComment  - %s\n", pprd3->pszComment);
                printf("  pszDrivers  - %s\n", pprd3->pszDrivers);
                printf("  time        - %d usTimeOut - %X\n",
                           pprd3->time ,  pprd3->usTimeOut);
              }
          }
          DosFreeMem(pBuf) ;
       }
    } /* end if  */
    else
    {
        printf("SplEnumDevice splerr=%ld, cTotal=%ld, cReturned=%ld, cbNeeded=%ld\n",
               splerr, cTotal, cReturned, cbNeeded) ;
    }
    DosExit( EXIT_PROCESS , 0 ) ;
    return(splerr);
}   /* end main */
```

SplEnumDriver

This function lists printer presentation drivers on the local workstation or on a remote server.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplEnumDriver (PSZ pszComputerName, ULONG ulLevel, PPVOID pBuf,
                      ULONG cbBuf, PULONG pcReturned, PULONG pcTotal,
                      PULONG pcbNeeded, PPVOID pReserved)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where queues are to be listed.

A NULL string specifies the local workstation.

ulLevel (ULONG) – input
Level of detail.

The level of detail required. This must be 0.

pBuf (PPVOID) – output
Buffer.

The returned contents in the buffer are:

ulLevel	**Buffer Contents**
0	An array of PRDRIVINFO structures

cbBuf (ULONG) – input
Size, in bytes, of Buffer.

It must be greater than 0.

pcReturned (PULONG) – output
Number of entries returned.

pcTotal (PULONG) – output
Total number of entries available.

pcbNeeded (PULONG) – output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

pReserved (PPVOID) – output
Reserved value, must be NULL.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_InvalidComputer (2351)	The computer name is invalid.

Related Functions

- SplCreateDevice
- SplCreateQueue
- SplSetDevice
- SplSetQueue

Example Code

This sample code will enumerate all the drivers on a local computer.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>        /* for printf function   */
```

```
INT main ()
{
   SPLERR splerr ;
   ULONG  cbBuf ;
   ULONG  cTotal ;
   ULONG  cReturned ;
   ULONG  cbNeeded ;
   ULONG  i ;
   PSZ    pszComputerName = NULL ;
   PSZ    pszDriverName ;
   PBYTE  pbuf ;

   /* Call the function the first time with zero in cbBuf. The count of bytes */
   /* needed for the buffer to hold all the info will be returned in cbNeeded.*/
   splerr = SplEnumDriver(pszComputerName, 0L, NULL, 0L,
                          &cReturned, &cTotal, &cbNeeded,
                          NULL );

   /* If the return code is ERROR_MORE_DATA or NERR_BufTooSmall, then all the */
   /* parameters were correct; and we can continue.                           */
   if (splerr == ERROR_MORE_DATA || splerr == NERR_BufTooSmall)
   {
      /* Allocate memory for the buffer to hold the returned information. Use */
      /* the count of bytes that were returned by our first call.             */
      if (!DosAllocMem( &pbuf, cbNeeded,
                        PAG_READ|PAG_WRITE|PAG_COMMIT) )
      {
         /* Set count of bytes to the value returned by our first call.       */
         cbBuf= cbNeeded ;

         /* Now call the function a second time with the correct values, and  */
         /* the information will be returned in the buffer.                   */
         splerr= SplEnumDriver(pszComputerName, 0L, pbuf, cbBuf,
                               &cReturned ,&cTotal, &cbNeeded,
                               NULL ) ;
         if (splerr == NO_ERROR)
         {
            /* Set a pointer to point to the beginning of the buffer.         */
            pszDriverName = (PSZ)pbuf;
```

```
            /* Print the names that are in the buffer. The count of the number*/
            /* of names in pBuf have been returned in cReturned.              */
            for (i=0;i < cReturned ; i++)
            {
               printf("Driver name - %s\n", pszDriverName) ;
               /* Increment the pointer to point to the next name.            */
               pszDriverName += DRIV_NAME_SIZE + DRIV_DEVICENAME_SIZE + 2;
            }
         }
         /* Free the memory allocated for the buffer.                         */
         DosFreeMem(pbuf) ;
      }
   }
   else
   {
      /* If the first call to the function returned any error code other     */
      /* than ERROR_MORE_DATA or NERR_BufTooSmall, we print the following.   */
      printf("SplEnumDriver error=%ld \n",splerr) ;
   }
   DosExit( EXIT_PROCESS , 0 ) ;
   return (splerr);
}
```

SplEnumJob

This function lists the jobs in a print queue, optionally supplying status information on each job.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplEnumJob  (PSZ pszComputerName, PSZ pszQueueName,
                    ULONG ulLevel, PPVOID pBuf, PULONG cbBuf,
                    PULONG pcReturned, PULONG pcTotal,
                    PULONG pcbNeeded, PPVOID pReserved)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where jobs are to be listed.

A NULL string specifies the local workstation.

pszQueueName (PSZ) – input
Queue name.

ulLevel (ULONG) – input
Level of detail required.

This must be 0 or 2.

pBuf (PPVOID) – output
Buffer.

The returned contents in the buffer depend on the value specified in *ulLevel* as follows:

ulLevel	**Buffer Contents**
0	An array containing a *uJobId* for each of *pcReturned* jobs.
2	An array containing a PRJINFO2 structure for each of *pcReturned* jobs.

cbBuf (PULONG) – input
Size, in bytes, of Buffer.

It must be greater than 0.

pcReturned (PULONG) – output
Number of entries returned.

pcTotal (PULONG) – output
Number of entries available.

pcbNeeded (PULONG) – output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

pReserved (PPVOID) – output
Reserved vaue, must be NULL.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_InvalidComputer (2351)	The computer name is invalid.

Related Functions

- SplCopyJob
- SplDeleteJob
- SplQueryJob

Example Code

This sample code accepts a queue name from the command line, and then prints out all the information associated with each job in that queue. Level 0 and 2 are valid; we have chosen to print out level 2 information.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>      /* for printf function */
```

```
INT main (argc, argv)
  INT argc;
  CHAR *argv[];
{
  ULONG  splerr ;
  ULONG  cbBuf ;
  ULONG  cTotal ;
  ULONG  cReturned ;
  ULONG  cbNeeded ;
  ULONG  ulLevel;
  ULONG  i ;
  PSZ    pszComputerName ;
  PSZ    pszQueueName ;
  PVOID  pBuf = NULL;
  PPRJINFO2 pprj2 ;

  /* Check that the command line entry was two parameters.                  */
  if (argc != 2)
  {
    printf("Syntax:  enumjob  QueueName\n");
    DosExit( EXIT_PROCESS , 0 ) ;
  }
  /* Either a NULL or a pointer to a NULL specify the local workstation.    */
  pszComputerName = (PSZ)NULL ;

  /* Set queue name equal to the value entered at the command line.         */
  pszQueueName = argv[1];

  /* Valid level are 0 and2. Level 2 gives info for a PRJINFO2 structure.   */
  ulLevel = 2L;

  /* Make the call the first time with cbBuf = zero so that we can get a    */
  /* return of the number of bytes that are need for pBuf to hold all of    */
  /* the information. The bytes needed will be returned in cbNeeded.        */
  splerr = SplEnumJob(pszComputerName,pszQueueName, ulLevel, pBuf,0L,
                      &cReturned, &cTotal,
                      &cbNeeded, NULL) ;

  /* Check that the return code is one of the two valid errors at this time. */
  if (splerr == ERROR_MORE_DATA || splerr == NERR_BufTooSmall )
  {
     /* Allocate memory for pBuf. ( No error checking is done on DosAllocMem */
     /* call to keep this sample code simple.)                               */
     DosAllocMem( &pBuf, cbNeeded,
                  PAG_READ|PAG_WRITE|PAG_COMMIT );
```

```
    /* Set bytes needed for buffer to the value returned by the first call.  */
    cbBuf = cbNeeded ;

    /* Make the call with all the valid information.                         */
    SplEnumJob(pszComputerName,pszQueueName, ulLevel,
                        pBuf, cbBuf, &cReturned,&cTotal,
                        &cbNeeded,NULL );

    /* Set up a pointer to point to the beginning of the buffer in which we */
    /* have the returned information                                         */
    pprj2=(PPRJINFO2)pBuf;

    /* The number of structures in the buffer(pBuf) are returned in cReturned*/
    /* Implement a for loop to print out the information for each structure. */
    for (i=0; i<cReturned ;i++ )
    {
        printf("Job ID       = %d\n",  pprj2->uJobId);
        printf("Job Priority = %d\n",  pprj2->uPriority);
        printf("User Name    = %s\n",  pprj2->pszUserName);
        printf("Position     = %d\n",  pprj2->uPosition);
        printf("Status       = %d\n",  pprj2->fsStatus);
        printf("Submitted    = %ld\n", pprj2->ulSubmitted);
        printf("Size         = %ld\n", pprj2->ulSize);
        printf("Comment      = %s\n",  pprj2->pszComment);
        printf("Document     = %s\n\n",pprj2->pszDocument);

        /* Increment the pointer to point to the next structure in the buffer*/
        pprj2++;
    } /* endfor */
    /* Free the memory that we allocated to make the call.                   */
    DosFreeMem(pBuf) ;
  }
  else
  {
  /* If any other error other than ERROR_MORE_DATA or NERR_BufTooSmall, then  */
  /* print the returned information.                                          */
    printf("SplEnumJob Error=%ld, Total Jobs=%ld, Returned Jobs=%ld, Bytes Needed=%ld\n",
            splerr, cTotal, cReturned, cbNeeded) ;
  }
  DosExit( EXIT_PROCESS , 0 ) ;
  return (splerr);
}
```

SplEnumPort

This function lists printer ports on the local workstation or on a remote server.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplEnumPort (PSZ pszComputerName, ULONG ulLevel, PPVOID pBuf,
                    ULONG cbBuf, PULONG pcReturned, PULONG pcTotal,
                    PULONG pcbNeeded, PVOID pReserved)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where queues are to be listed.

A NULL string specifies the local workstation.

ulLevel (ULONG) – input
Level of detail.

The level of detail required. This must be 0 or 1.

pBuf (PPVOID) – output
Buffer.

The returned content in the buffer depends on the value specified in *ulLevel* as follows:

ulLevel	**Buffer Contents**
0	An array of PRPORTINFO structures
1	An array of PRPORTINFO1 structures

cbBuf (ULONG) – input
Size, in bytes, of Buffer.

pcReturned (PULONG) – output
Number of entries returned.

pcTotal (PULONG) – output
Total number of entries available.

pcbNeeded (PULONG) – output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

pReserved (PVOID) – output
Reserved.

This must be NULL.

Returns

rc (SPLERR) – returns

Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_InvalidComputer (2351)	The computer name is invalid.

Related Functions

- SplCreateDevice
- SplSetDevice

Example Code

This sample code will print out all the ports an associated information. This is done at level 1, and for the local workstation.

```
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>

INT main ()
{
   SPLERR splerr ;
   ULONG  cbBuf ;
   ULONG  cTotal;
   ULONG  cReturned ;
   ULONG  cbNeeded ;
   ULONG  ulLevel = 1;
   ULONG  i ;
   PSZ    pszComputerName = NULL;
   PVOID  pbuf ;
   PPRPORTINFO1 pPort1 ;
```

```
   splerr = SplEnumPort(pszComputerName, ulLevel, pbuf, 0L, /* cbBuf */
                        &cReturned, &cTotal,
                        &cbNeeded, NULL) ;

   if (splerr == ERROR_MORE_DATA || NERR_BufTooSmall )
   {
      if (!DosAllocMem( &pbuf, cbNeeded,
                        PAG_READ|PAG_WRITE|PAG_COMMIT) )
      {
          cbBuf = cbNeeded ;
          splerr = SplEnumPort(pszComputerName, ulLevel, pbuf, cbBuf,
                               &cReturned, &cTotal,
                               &cbNeeded, NULL) ;
          if (splerr == 0L)
          {
             pPort1 = (PPRPORTINFO1)pbuf ;
             printf("Port names: ");
             for (i=0; i < cReturned; i++)
             {
                printf("Port - %s, Driver - %s Path - %s\n              ",
                       pPort1->pszPortName, pPort1->pszPortDriverName,
                       pPort1->pszPortDriverPathName ) ;
                pPort1++ ;
             }
             printf("\n");
          }
          DosFreeMem(pbuf) ;
      }
   }
   else
   {
      printf("SplEnumPort splerr=%ld, \n",splerr) ;
   }
   DosExit( EXIT_PROCESS , 0 ) ;
   return (splerr);
}   /* end main */
```

SplEnumPrinter

This function lists print destinations in the system.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplEnumPrinter (PSZ PSZComputerName, ULONG ulLevel,
                       ULONG flType, PPVOID pBuf, PULONG cbBuf,
                       PULONG pcReturned, PULONG pcTotal,
                       PULONG pcbNeeded, PPVOID pReserved)
```

Parameters

PSZComputerName (PSZ) – input
Name of computer where queues are to be listed.

This must be NULL.

ulLevel (ULONG) – input
Level of detail required.

This must be 0.

flType (ULONG) – input
Type of print destinations required.

SPL_PR_QUEUE	Return only queues
SPL_PR_DIRECT_DEVICE	Return only direct print devices
SPL_PR_QUEUED_DEVICE	Return only queued print devices
SPL_PR_LOCAL_ONLY	Return only local print destinations

pBuf (PPVOID) – output
Buffer.

The returned contents in the buffer are as follows:

ulLevel	**Buffer Contents**
0	An array of PRINTERINFO structures.

When the names of print destinations are returned, calls can be made to SplQueryQueue or SplQueryDevice to find out further information about the print destination.

cbBuf (PULONG) – input
Size, in bytes, of Buffer.

It must be greater than 0.

pcReturned (PULONG) – output
Number of entries returned.

pcTotal (PULONG) – output
Number of entries available.

pcbNeeded (PULONG) – output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

pReserved (PPVOID) – output
Reserved value, must be NULL.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.

Related Functions

- SplQueryDevice
- SplQueryQueue

Example Code

This example code will print out all queues and printers for the local computer. It will print out both printers that are attached to a queue, and those that are direct printers.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>      /* for printf function */
```

```
INT main ()
{
    PVOID pBuf;
    ULONG fsType ;
    ULONG cbBuf ;
    ULONG cRes ;
    ULONG cTotal ;
    ULONG cbNeeded ;
    SPLERR splerr = 0 ;
    PPRINTERINFO pRes ;

    /* Set fsType to use all the flags. We will print out local device/queues.*/
    fsType = SPL_PR_QUEUE | SPL_PR_DIRECT_DEVICE |
             SPL_PR_QUEUED_DEVICE | SPL_PR_LOCAL_ONLY;

    /* Make function call with cbBuf equal to zero to get a return in cbNeeded*/
    /* of the number of bytes needed for  buffer to hold all the information  */

    splerr = SplEnumPrinter ( NULL,0 ,fsType ,NULL ,NULL ,&cRes ,
                              &cTotal,&cbNeeded ,NULL ) ;

    /* The error return code will be one of the two following codes if        */
    /* all the parameters were correct.  Otherwise it could be                */
    /* ERROR_INVALID_PARAMETER.                                          */

    if ( splerr == ERROR_MORE_DATA || splerr == NERR_BufTooSmall )
    {
       /* Allocate memory for the buffer using the count of bytes that were   */
       /* returned in cbNeeded.  For simplicity, no error checking is done.   */
       DosAllocMem( &pBuf, cbNeeded,
                    PAG_READ|PAG_WRITE|PAG_COMMIT);

       /* Set count of bytes in buffer to value used to allocate buffer.      */
       cbBuf = cbNeeded;

       /* Call function again with the correct buffer size.                   */
       splerr = SplEnumPrinter  ( NULL,0 ,fsType ,pBuf ,cbBuf ,&cRes ,
                                  &cTotal,&cbNeeded,NULL);
```

```
      /* If there are any returned structures in the buffer, then we will    */
      /* print out some of the information.                                   */
      if (cRes)
      {
         pRes = (PPRINTERINFO)pBuf ;
         while ( cRes-- )
         {
            /* Look at the flType element in the pRes structure to determine */
            /* what type of print destination the structure represents.      */
            switch (pRes[cRes].flType)
            {
               case  SPL_PR_QUEUE:
                  printf("Print destination %s is a queue.\n",
                  pRes[cRes].pszPrintDestinationName) ;
                  break;
               case SPL_PR_QUEUED_DEVICE:
                  printf("Print destination %s is a queued printer.\n",
                  pRes[cRes].pszPrintDestinationName) ;
                  break;
               case SPL_PR_DIRECT_DEVICE:
                  printf("Print destination %s is a direct printer.\n",
                  pRes[cRes].pszPrintDestinationName) ;
            }
            printf("Description -
                  %s\n\n",pRes[cRes].pszDescription) ;
         }
      }
      DosFreeMem(pBuf);
   }
   else
   {
      /* If we had any other return code other than ERROR_MORE_DATA or */
      /* NERR_BufTooSmall, we will print out the following information.   */
      printf("SplEnumPrinter error= %ld \n",splerr);
   }
   DosExit( EXIT_PROCESS , 0 ) ;
   return (splerr);
}
```

SplEnumQueue

This function lists print queues on the local workstation or on a remote server, optionally supplying additional information.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplEnumQueue (PSZ pszComputerName, ULONG ulLevel, PPVOID pBuf,
                     PULONG cbBuf, PULONG pcReturned,
                     PULONG pcTotal, PULONG pcbNeeded,
                     PPVOID pReserved)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where queues are to be listed.

A NULL string specifies the local workstation.

ulLevel (ULONG) – input
Level of detail.

The level of detail required. This must be 3, 4, 5 or 6.

pBuf (PPVOID) – output
Buffer.

The returned contents in the buffer depend on the value specified in *ulLevel* as follows:

ulLevel -	**Buffer Contents**
3	An array of PRQINFO3 structures. The *fsType* bit PRQ3_TYPE_APPDEFAULT is set for the application default queue only.
4	An array of *pcReturned* PRQINFO3 structures in which each PRQINFO3 structure is followed by an array of PRJINFO2 structures, one for each of job in the queue. *cJobs* in the PRQINFO3 or PRQINFO6 structure gives the number of jobs in the array.
5	A queue name of type PSZ.
6	An array of PRQINFO6 structures

cbBuf (PULONG) – input
Size, in bytes, of Buffer.

It must be greater than 0.

pcReturned (PULONG) – output
Number of entries returned.

pcTotal (PULONG) – output
Total number of entries available.

pcbNeeded (PULONG) – output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

pReserved (PPVOID) – output
Reserved value, must be NULL.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_InvalidComputer (2351)	The computer name is invalid.

Related Functions

- SplQueryJob
- SplQueryQueue
- SplSetJob
- SplSetQueue

Example Code

This sample code enumerates all the queues and the jobs in them that are on the local workstation.

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>

INT main ()
{
   SPLERR splerr;
   USHORT jobCount ;
   ULONG  cbBuf ;
   ULONG  cTotal;
   ULONG  cReturned ;
   ULONG  cbNeeded ;
   ULONG  ulLevel ;
   ULONG  i,j ;
   PSZ    pszComputerName ;
   PBYTE  pBuf ;
   PPRQINFO3 prq ;
   PPRJINFO2 prj2 ;

   ulLevel = 4L;
   pszComputerName = (PSZ)NULL ;
   splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, 0L, /* cbBuf */
                         &cReturned, &cTotal,
                         &cbNeeded, NULL)
   if ( splerr == ERROR_MORE_DATA || splerr == NERR_BufTooSmall )
   {
      if (!DosAllocMem( &pBuf, cbNeeded,
                        PAG_READ|PAG_WRITE|PAG_COMMIT) )
      {
         cbBuf = cbNeeded ;
         splerr = SplEnumQueue(pszComputerName, ulLevel, pBuf, cbBuf,
                               &cReturned, &cTotal,
                               &cbNeeded, NULL)
         if (splerr == NO_ERROR)
         {
            /* Set pointer to point to the beginning of the buffer.          */
            prq = (PPRQINFO3)pBuf ;
```

```
/* cReturned has the count of the number of PRQINFO3 structures.  */
for (i=0;i < cReturned ; i++)
{
   printf("Queue info: name - %s\n", prq->pszName) ;
if ( prq->fsType & PRQ3_TYPE_APPDEFAULT )
   printf("  This is the application default print queue\n");
   printf("  priority - %d  starttime - %d  endtime - %d
          fsType - %X\n",
          prq->uPriority ,  prq->uStartTime ,
          prq->uUntilTime , prq->fsType ) ;
   printf("  pszSepFile   - %s\n", prq->pszSepFile) ;
   printf("  pszPrProc    - %s\n", prq->pszPrProc) ;
   printf("  pszParms     - %s\n", prq->pszParms) ;
   printf("  pszComment   - %s\n", prq->pszComment) ;
   printf("  pszPrinters  - %s\n", prq->pszPrinters) ;
   printf("  pszDriverName- %s\n", prq->pszDriverName) ;
   if (prq->pDriverData)
   {
      printf("  pDriverData->cb          - %ld\n",
             (ULONG)prq->pDriverData->cb);
      printf("  pDriverData->lVersion    - %ld\n",
             (ULONG)prq->pDriverData->lVersion) ;
      printf("  pDriverData->szDeviceName- %s\n",
             prq->pDriverData->szDeviceName) ;
   }
   /* Save the count of jobs. There are this many PRJINFO2        */
   /* structures following the PRQINFO3 structure.                */
   jobCount = prq->cJobs;
   printf("Job count in this queue is %d\n\n",jobCount);

   /* Increment the pointer past the PRQINFO3  structure.         */
   prq++;

   /* Set a pointer to point to the first PRJINFO2 structure.     */
   prj2=(PPRJINFO2)prq;
   for (j=0;j < jobCount ; j++)
   {
      printf("Job ID      = %d\n", prj2->uJobId);
      printf("Job Priority= %d\n", prj2->uPriority);
      printf("User Name   = %s\n", prj2->pszUserName);
      printf("Position    = %d\n", prj2->uPosition);
      printf("Status      = %d\n", prj2->fsStatus);
      printf("Submitted   = %ld\n",prj2->ulSubmitted);
```

```
                printf("Size        = %ld\n",prj2->ulSize);
                printf("Comment     = %s\n", prj2->pszComment);
                printf("Document    = %s\n\n",prj2->pszDocument);

                /* Increment the pointer to point to the next structure.    */
                prj2++;
              } /* endfor jobCount */

              /* After doing all the job structures, prj2 points to the next */
              /* queue structure. Set the pointer for a PRQINFO3 structure.  */
              prq = (PPRQINFO3)prj2;
            }/*endfor cReturned */
          }
          DosFreeMem(pBuf) ;
        }
    } /* end if Q level given */
    else
    {
       /* If we are here we had a bad error code. Print it and some other info.*/
       printf("SplEnumQueue Error=%ld, Total=%ld, Returned=%ld, Needed=%ld\n",
              splerr, cTotal, cReturned, cbNeeded) ;
    }
    DosExit( EXIT_PROCESS , 0 ) ;
    return(splerr);
}   /* end main */
```

SplEnumQueueProcessor

This function lists printer queue processors on the local workstation or on a remote server.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplEnumQueueProcessor (PSZ pszComputerName, ULONG ulLevel,
                              PPVOID pBuf, ULONG cbBuf,
                              PULONG pcReturned, PULONG pcTotal,
                              PULONG pcbNeeded, PPVOID pReserved)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where queues are to be listed.

A NULL string specifies the local workstation.

ulLevel (ULONG) – input
Level of detail.

The level of detail required. This must be 0.

pBuf (PPVOID) – output
Buffer.

The returned contents of the buffer are as follows:

ulLevel	**Buffer Contents**
0	An array of PRQPROCINFO structures

cbBuf (ULONG) – input
Size, in bytes, of Buffer.

It must be greater than 0.

pcReturned (PULONG) – output
Number of entries returned.

pcTotal (PULONG) – output
Total number of entries available.

pcbNeeded (PULONG) – output
Size in bytes of available information.

A value of 0 specifies that the size is not known.

pReserved (PPVOID) – output
Reserved value, must be NULL.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_InvalidComputer (2351)	The computer name is invalid.

Related Functions

- SplSetQueue

Example Code

This sample code enumerates and prints all the queue processors on the local computer.

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>        /* for printf function   */

INT main ()
{
   SPLERR splerr ;
   ULONG  cbBuf ;
   ULONG  cTotal ;
   ULONG  cReturned ;
   ULONG  cbNeeded ;
   ULONG  i ;
   PSZ    pszComputerName = NULL ;
   PSZ    pszQProcName ;
   PBYTE  pBuf ;
```

```
/* Call the function the first time with zero in cbBuf. The count  */
/* of bytes needed for the buffer to hold all the info will be     */
/* returned in cbNeeded.                                           */
splerr = SplEnumQueueProcessor(pszComputerName, 0L, NULL, 0L,
                               &cReturned, &cTotal,
                               &cbNeeded,NULL );

/* If the return code is ERROR_MORE_DATA or NERR_BufTooSmall,     */
/* then all the parameters were correct; and we can continue.      */
if (splerr == ERROR_MORE_DATA || splerr == NERR_BufTooSmall)
{
   /* Allocate memory for the buffer to hold the returned information. Use */
   /* the count of bytes that were returned by our first call.             */
   if (!DosAllocMem( &pbuf, cbNeeded,
                     PAG_READ|PAG_WRITE|PAG_COMMIT) )
   {
      /* Set count of bytes to the value returned by our first call.       */
      cbBuf = cbNeeded ;

      /* Now call the function a second time with the correct values, and  */
      /* the information will be returned in the buffer.                   */
      splerr = SplEnumQueueProcessor(pszComputerName, 0L, pBuf, cbBuf,
                                     &cReturned, &cTotal,
                                     &cbNeeded,NULL ) ;

      /* If we have no errors, then print out the buffer information.      */
      if (splerr == NO_ERROR)
      {
         /* Set a pointer to point to the beginning of the buffer.         */
         pszQProcName = (PSZ)pBuf;

         /* Print the names that are in the buffer. The count of the number*/
         /* of names in pBuf have been returned in cReturned.              */
         for (i=0;i < cReturned ; i++)
         {
            printf("Queue Processor name - %s\n", pszQProcName) ;

            /* Increment the pointer to point to the next name.            */
            pszQProcName += DRIV_NAME_SIZE + 1;
         }
      }
      /* Free the memory allocated for the buffer.                         */
      DosFreeMem(pBuf) ;
   }
}
```

```
    else
    {
       /* If the first call to the function returned any other error code      */
       /* except ERROR_MORE_DATA or NERR_BufTooSmall, we print the    */
       /* following.                                                           */
       printf("SplEnumQueueProcessor error=%ld\n",splerr ) ;
    }
    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}
```

SplHoldJob

This function holds a job in a print queue.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplHoldJob  (PSZ pszComputerName, PSZ pszQueueName,
                    ULONG ulJob)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where job is to be paused.

A NULL string specifies the local workstation.

pszQueueName (PSZ) – input
Queue Name.

ulJob (ULONG) – input
Job identification number.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
NERR_NetNotStarted (2102)	The network program is not installed.
NERR_JobNotFound (2151)	The print job does not exist.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_JobInvalidState (2164)	This operation cannot be performed on the print job in its current state.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

If the job is already printing when the call is made, NERR_JobInvalidState (2164) is returned.

A user with administrator privilege can hold any job.

A job created locally can be held locally regardless of user privilege level, but can be held remotely only by an administrator.

A remote job can be held by a user without administrator privilege only if the username of the person initiating the request is the same as the username of the person who created the job.

Related Functions

- SplEnumJob
- SplQueryJob
- SplReleaseJob

Example Code

This sample code will hold the queue name that is entered at the prompt.

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>       /* for printf function */
#include <stdlib.h>      /* for atoi function   */

INT main (argc, argv)
   INT argc;
   CHAR *argv[];
{
   SPLERR splerr ;
   PSZ    pszComputerName = NULL ;
   PSZ    pszQueueName = NULL ;
   ULONG  ulJob ;

   /* Get job id from the input argument.  */
   ulJob = atoi(argv[1]);

   /* Call the function to do the hold. If an error is returned, print it. */
   splerr = SplHoldJob( pszComputerName, pszQueueName, ulJob);

   switch (splerr)
   {
      case NO_ERROR:
         printf("Job %d was held.\n",ulJob);
         break;
      case NERR_JobNotFound:
         printf("Job does not exist.\n");
         break;
      case NERR_JobInvalidState:
         printf("This operation can't be performed on the print Job.\n");
         break;
      default:
         printf("Errorcode = %ld\n",splerr);
   } /* endswitch */
   DosExit( EXIT_PROCESS , 0 ) ;
   argc;
   return (splerr);
}
```

SplHoldQueue

This function holds a print queue.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplHoldQueue  (PSZ pszComputerName, PSZ pszQueueName)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where queue is to be paused.

A NULL string specifies the local workstation.

pszQueueName (PSZ) – input
Queue name.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

This function suspends processing of all print jobs except for a job currently printing. Print jobs can be submitted to a held queue, but no jobs will be spooled to a print destination or print processor until the queue is released by a SplHoldQueue call.

To hold a remote queue requires administrator privilege on the remote server.

Related Functions

- SplCreateQueue
- SplEnumQueue
- SplQueryQueue
- SplReleaseQueue

Example Code

This sample code will hold the local queue name that is entered at the prompt.

```
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>       /* for printf function */

INT main (argc, argv)
   INT argc;
   CHAR *argv[];
{
   SPLERR splerr ;
   PSZ    pszComputerName = NULL ;
   PSZ    pszQueueName ;

   /* Get queue name from the input argument  */
   pszQueueName = argv[1];

   /* Call the function to do the hold. If an error is returned, print it.   */
   splerr = SplHoldQueue(pszComputerName, pszQueueName);
   if (splerr != 0L)
   {
      switch (splerr)
      {
         case NERR_QNotFound:
            printf("Queue does not exist.\n");
            break;
         case  NERR_SpoolerNotLoaded:
            printf("The Spooler is not running.\n");
            break;
         default:
            printf("Errorcode = %ld\n",splerr);
      } /* endswitch */
   }
   else
   {
      printf("Queue %s was held.\n",pszQueueName);
   } /* endif */
  DosExit( EXIT_PROCESS , 0 ) ;
  argc; /* keep the compiler quiet */
  return (splerr);
}
```

SplMessageBox

Related Methods

SplMessageBox creates and displays a message box.

The message queue will be created (and destroyed) if necessary. In OS/2 2.1, this API may return Retry automatically if the PM message box is displayed for more than three minutes.

Syntax

```
#include <os2.h>

ULONG SplMessageBox (PSZ pszAddress, ULONG flErrorInfo,
                     ULONG flErrorData, PSZ pszText, PSZ pszCaption,
                     USHORT idWindow, USHORT fsStyle)
```

Parameters

pszAddress (PSZ) – input
Pointer to a string containing the logical address of the device, such as LPT1.

flErrorInfo (ULONG) – input
Error information.

One of the following flags must be set to identify where the error occurred:

SPLINFO_QPERROR	Spooler queue processor error
SPLINFO_DDERROR	Presentation driver error
SPLINFO_SPLERROR	Spooler error
SPLINFO_OTHERERROR	Any other error

One of the following flags is also set to indicate the severity of the error:

SPLINFO_INFORMATION	Information only, no error.
SPLINFO_WARNING	Warning.
SPLINFO_ERROR	Recoverable error.
SPLINFO_SEVERE	Severe, unrecoverable error.
SPLINFO_USERINTREQD	This flag is optional. It shows that recovery requires action from the user.

flErrorData (ULONG) – input
Length of the data in bytes.

Error data:

SPLDATA_PRINTERJAM	Printer is jammed, offline, or not turned on
SPLDATA_FORMCHGREQD	Form change required
SPLDATA_CARTCHGREQD	Font cartridge change required
SPLDATA_PENCHGREQD	Pen change required
SPLDATA_DATAERROR	Data error, such as missing file
SPLDATA_UNEXPECTERROR	Unexpected DOS error
SPLDATA_OTHER	Any other error

pszText (PSZ) – input
Pointer to the text string for the message box.

pszCaption (PSZ) – input
Pointer to a string containing a meaningful title for the message box.

The text is centered in the title bar. If more than 40 characters are supplied, excess characters at the beginning and end of the string are not displayed.

idWindow (USHORT) – input
Window ID of the message box window.

fsStyle (USHORT) – input
Bit array specifying the contents and function of the message box.

Returns

Response (ULONG) – returns
Return codes.

This function returns a USHORT value (sResponse) that indicates the user's response.

Remarks

SplMessagBox creates a message queue if the current thread does not have one.

SplMessageBox is similar to WinMessageBox.

SplPurgeQueue

This function removes all jobs, except any currently printing, from a print queue.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplPurgeQueue  (PSZ pszComputerName, PSZ pszQueueName)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where queue is to be purged.

A NULL string specifies the local workstation.

pszQueueName (PSZ) – input
Queue name.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter was specified.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

A print job that is printing is not affected by this function.

If a print queue is pending deletion when this function is made, the queue is deleted when the job that is currently printing ends.

To purge a remote queue requires administrator privilege on the remote server.

Related Functions

- SplCreateQueue
- SplEnumQueue
- SplQueryQueue

Example Code

This code will purge a local queue, whose name is entered at the prompt.

```
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>       /* for printf function */

INT main (argc, argv)
   INT argc;
   CHAR *arg[];
{
   SPLERR splerr ;
   PSZ    pszComputerName = NULL ;
   PSZ    pszQueueName ;

   /* Get queue name from the input argument.                          */
   pszQueueName = arg[1];

   /* Call the function to do the purge. If an error is returned, print it.   */
   splerr=SplPurgeQueue(pszComputerName, pszQueueName);
   if (splerr != 0L)
   {
      switch (splerr)
      {
         case NERR_QNotFound:
            printf("Queue does not exist.\n");
            break;
         case  NERR_SpoolerNotLoaded:
            printf("The Spooler is not running.\n");
            break;
         default:
            printf("Errorcode = %ld\n",splerr);
      } /* endswitch */
   }
   else
   {
      printf("Queue %s was purged.\n",pszQueueName);
   } /* endif */

  DosExit( EXIT_PROCESS , 0 ) ;
  return (splerr);
}
```

SplQmAbort

Related Methods

SplQmAbort stops the generation of the spool file. It automatically closes the spool file (see "SplQmClose" on page 6-64).

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
BOOL SplQmAbort (HSPL hspl)
```

Parameters

hspl (HSPL) – input
Spooler handle.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_SPL_QUEUE_ERROR (0x4004)	No spooler queue supplied or found.
PMERR_SPL_INV_HSPL (0x4005)	The spooler handle is invalid.

Related Functions

Prerequisite Functions

- SplQmOpen

Related Functions

- DevEscape

Example Code

This function is used to stop the generation of spool files and automatically close the spool file.

```
#define INCL_SPL
#include <OS2.H>

HSPL hspl; /* spooler handle. */

SplQmAbort(hspl);
```

SplQmAbortDoc

Related Methods

SplQmAbortDoc ends a print job.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
BOOL SplQmAbortDoc  (HSPL hspl)
```

Parameters

hspl (HSPL) – input
Spooler handle.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_SPL_QUEUE_ERROR (0x4004)	No spooler queue supplied or found.
PMERR_STARTDOC_NOT_ISSUED (0x2104)	A request to write spooled output without first issuing a STARTDOC was attempted.
PMERR_SPL_INV_HSPL (0x4005)	The spooler handle is invalid.

Related Functions

Prerequisite Functions

- SplQmOpen
- SplQmStartDoc

Related Functions

- DevEscape

Example Code

This function is used to end a print job. Everything that has been written to the spool file for this job since the last SplQmStartDoc is erased, including the SplQmStartDoc.

```
#define INCL_SPL
#include <OS2.H>

HSPL hspl; /* spooler handle. */

SplQmAbortDoc(hspl);
```

SplQmClose

Related Methods

SplQmClose closes the spool file. It corresponds to the DevCloseDC function.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
BOOL SplQmClose (HSPL hspl)
```

Parameters

hspl (HSPL) – input
Spooler handle.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_SPL_QUEUE_ERROR (0x4004)	No spooler queue supplied or found.
PMERR_ENDDOC_NOT_ISSUED (0x210B)	A request to close the spooled output without first issuing a an ENDDOC was attempted.
PMERR_SPL_INV_HSPL (0x4005)	The spooler handle is invalid.

Related Functions

Prerequisite Functions

- SplQmOpen

Related Functions

- DevCloseDC

Example Code

This function is used to close a spool file that was opened with SplQmOpen.

```
#define INCL_SPL
#include <OS2.H>

HSPL hspl; /* spooler handle. */

SplQmClose(hspl);
```

SplQmEndDoc

Related Methods

SplQmEndDoc ends a print job, and returns ulJob, a unique number to identify the job. This function corresponds to the DevEscape (DEVESC_ENDDOC) call.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
ULONG SplQmEndDoc (HSPL hspl)
```

Parameters

hspl (HSPL) – input
Spooler handle.

Returns

ulJob (ULONG) – returns
Job identifier.

Nonzero	Jobid (1 through 65,535)
SPL_ERROR	Error

Possible returns from WinGetLastError

PMERR_SPL_QUEUE_ERROR (0x4004)	No spooler queue supplied or found.
PMERR_SPL_NO_DATA (0x400D)	No data supplied or found.
PMERR_SPL_INV_HSPL (0x4005)	The spooler handle is invalid.

Related Functions

Prerequisite Functions

- SplQmOpen
- SplQmStartDoc

Related Functions

- DevEscape

Example Code

This function is used to end a print job and return the job id. The print-job identifier is displayed to the user by the spooler while this job is on the queue, and while it is being printed.

```
#define INCL_SPL
#include <OS2.H>

HSPL hspl;  /* spooler handle. */
ULONG jobid;
CHAR szMsg[100];
HWND hwndClient;

jobid = SplQmEndDoc(hspl);

sprintf(szMsg, "ending job %d",jobid);
WinMessageBox(HWND_DESKTOP,
    hwndClient,                /* client-window handle  */
    szMsg,                     /* body of the message   */
    "Printing Information",    /* title of the message */
    0,                         /* message box id        */
    MB_NOICON | MB_OK);        /* icon and button flags */
```

SplQmOpen

Related Methods

SplQmOpen opens a spool file for generating a print job. It corresponds to the DevOpenDC call.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
HSPL SplQmOpen (PSZ pszToken, LONG lCount, PQMOPENDATA pqmdopData)
```

Parameters

pszToken (PSZ) – input
Reserved value.

lCount (LONG) – input
Number of items.

This is the number of items present in the *pqmdopData* supplied. This can be shorter than the full list, if omitted items are irrelevant.

This parameter must be greater than 0.

pqmdopData (PQMOPENDATA) – input
Pointer to DevOpenStruc.

Returns

hspl (HSPL) – returns
Spooler handle.

Nonzero	Spooler handle
SPL_ERROR	Error

Possible returns from WinGetLastError

PMERR_INVALID_PARM (0x1303)	A parameter to the function contained invalid data.
PMERR_SPL_INV_LENGTH_OR_COUNT (0x401A)	The length or count is invalid.
PMERR_SPL_QUEUE_NOT_FOUND (0x4024)	The spooler queue definition could not be found.

PMERR_BASE_ERROR (0x2006)	An OS/2 base error has occurred. The base error code can be accessed using the OffBinaryData field of the ERRINFO structure returned by WinGetErrorInfo.

Related Functions

- DevOpenDC

Example Code

This sample code will initialize a PDEVOPENSTRUC and use it to call the function.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_BASE
#define INCL_ERRORS

#include <os2.h>
#include <stdio.h>
#include <stdlib.h>

VOID main()
{
  HSPL hspl;
  PDEVOPENSTRUC pdata;        /* Pointer to a DEVOPENSTRUC structure */
  PSZ  pszToken = "*";        /* Spooler info identifier             */

  /* Allocate memory for pdata */
  if ( !DosAllocMem( &pdata,sizeof( DEVOPENSTRUC ),
     (PAG_READ|PAG_WRITE|PAG_COMMIT ) )
  {
    /* Initialize elements of pdata */
    pdata->pszLogAddress      = "LPT1Q1";
    pdata->pszDriverName      = "IBMNULL";
    pdata->pdriv              = NULL;
    pdata->pszDataType        = "PM_Q_STD";
    pdata->pszComment         = NULL;
    pdata->pszQueueProcName   = NULL;
    pdata->pszQueueProcParams = NULL;
    pdata->pszSpoolerParams   = NULL;
    pdata->pszNetworkParams   = NULL;

    hspl = SplQmOpen( pszToken,4L,( PQMOPENDATA )pdata );
```

```
    if ( hspl != SPL_ERROR )        /* Good spooler handle */
    {
       printf("SplQmOpen handle is %d\n",hspl);
    }
    else
    {
       printf("SplQmOpen failed.\n");
    }
  }
}
```

SplQmStartDoc

Related Methods

SplQmStartDoc starts a print job. It corresponds to the DevEscape (DEVESC_STARTDOC) call.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
BOOL SplQmStartDoc (HSPL hspl, PSZ pszDocName)
```

Parameters

hspl (HSPL) – input
Spooler handle.

pszDocName (PSZ) – input
Document name.

This is part of the job description that is displayed to the end user by the spooler.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARM (0x1303)	A parameter to the function contained invalid data.
PMERR_SPL_QUEUE_ERROR (0x4004)	No spooler queue supplied or found.
PMERR_ENDDOC_NOT_ISSUED (0x210B)	A request to close the spooled output without first issuing a an ENDDOC was attempted.
PMERR_SPL_INV_HSPL (0x4005)	The spooler handle is invalid.

Related Functions

Prerequisite Functions

- SplQmOpen

Related Functions

- DevEscape

Example Code

This function indicates the start of a print job. It allows the application to specify a document name to be associated with the print job.

Multiple print jobs can be generated, within a single queue manager open, by bracketing each job with SplQmStartDoc and SplQmEndDoc.

```
#define INCL_SPL
#include <OS2.H>

HSPL hspl;  /* spooler handle. */
CHAR szDocName[] = "Test Job";
CHAR szMsg[100];
HWND hwndClient;

sprintf(szMsg, "Starting job named: %s",szDocName);
WinMessageBox(HWND_DESKTOP,
    hwndClient,                 /* client-window handle  */
    szMsg,                      /* body of the message   */
    "Printing Information",     /* title of the message */
    0,                          /* message box id        */
    MB_NOICON | MB_OK);  /* icon and button flags */

SplQmStartDoc(hspl,szDocName);
```

SplQmWrite

Related Methods

SplQmWrite writes a buffer of data to the spool file for the print job.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
BOOL SplQmWrite  (HSPL hspl, LONG lCount, PBYTE pData)
```

Parameters

hspl (HSPL) – input
Spooler handle.

lCount (LONG) – input
Length in bytes.

This is the length of *pData*; it must not be less than 1 or greater than 65 535. Data that is longer than this must be written by two or more calls.

pData (PBYTE) – input
Buffer of data to be written to the spool file.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARM (0x1303)	A parameter to the function contained invalid data.
PMERR_BASE_ERROR (0x2006)	An OS/2 base error has occurred. The base error code can be accessed using the OffBinaryData field of the ERRINFO structure returned by WinGetErrorInfo.
PMERR_SPL_INV_LENGTH_OR_COUNT (0x401A)	The length or count is invalid.

PMERR_SPL_QUEUE_ERROR (0x4004)	No spooler queue supplied or found.
PMERR_SPL_PRINT_ABORT (0x4008)	The job has already been aborted.
PMERR_STARTDOC_NOT_ISSUED (0x2104)	A request to write spooled output without first issuing a STARTDOC was attempted.
PMERR_SPL_CANNOT_OPEN_FILE (0x401C)	Unable to open the file.
PMERR_SPL_INV_HSPL (0x4005)	The spooler handle is invalid.
PMERR_SPL_NO_DISK_SPACE (0x4006)	There is not enough free disk space.

Related Functions

Prerequisite Functions

- SplQmOpen
- SplQmStartDoc

Example Code

This function writes a buffer of data to the spool file for the print job. The size of the data buffer must not be greater than 64KB. Print jobs that exceed the maximum buffer size must be written by repeatedly calling to this function.

```
#define INCL_SPL
#include <OS2.H>

HSPL hspl; /* spooler handle. */

SplQmWrite(hspl,
           sizeof("DATA"),
           (PVOID)"DATA");
```

SplQueryDevice

This function retrieves information about a print device.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplQueryDevice  (PSZ pszComputerName, PSZ pszPrintDeviceName,
                        ULONG ulLevel, PPVOID pBuf, PULONG cbBuf,
                        PULONG pcbNeeded)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where print device is to be queried.

A NULL string specifies the local workstation.

pszPrintDeviceName (PSZ) – input
Name of Print Device.

This can specify a print device name or a port name. If *ulLevel* is 0, it must be a port name. If *ulLevel* is 2 or 3, it must be a print device name.

ulLevel (ULONG) – input
Level of detail required.

This must be 0, 2 or 3.

pBuf (PPVOID) – in/out
Buffer.

The returned contents of the buffer depend on the value specified in *ulLevel* as follows:

ulLevel	**Buffer Contents**
0	A port name consisting of 9 characters, including a null terminator.
2	A print device name of type PSZ.
3	A PRDINFO3 structure.

cbBuf (PULONG) – input
Size, in bytes, of Buffer.

It must be greater than 0.

pcbNeeded (PULONG) – output
Size in bytes of available information.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_DestNotFound (2152)	The print device cannot be found.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

If *ulLevel* is 0, the port name in *pBuf* is truncated to 8 characters.

If *ulLevel* is 3, and *pBuf* cannot hold the entire PRDINFO3 structure, SplQueryDevice returns NERR_BufTooSmall (2123).

To obtain the size of buffer required, call SplQueryDevice with the required value of *ulLevel* and a NULL buffer. The number of bytes required is returned in *pcbNeeded*.

If no job is printing on the print device, bits 2–11 of *fsStatus* in the PRDINFO3 data structure are meaningless.

Related Functions

- SplCreateDevice
- SplDeleteDevice
- SplEnumDevice

Example Code

This sample code returns information for the device name that is entered at the command line. The local workstation is selected. The query is done for level 3 information.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    SPLERR splerr  ;
    ULONG  cbBuf;
    ULONG  cbNeeded ;
    ULONG  ulLevel ;
    PSZ    pszComputerName ;
    PSZ    pszPrintDeviceName ;
    PVOID  pBuf ;
    PPRDINFO3 pprd3 ;

    if (argc != 2)
    {
       printf("Syntax:  sdqry   DeviceName  \n");
       DosExit( EXIT_PROCESS , 0 ) ;
    }

    pszComputerName = (PSZ)NULL ;
    pszPrintDeviceName = argv[1];
    ulLevel = 3;
    splerr = SplQueryDevice(pszComputerName, pszPrintDeviceName,
                            ulLevel, (PVOID)NULL, 0L, &cbNeeded );
    if (splerr != NERR_BufTooSmall)
    {
       printf("SplQueryDevice Err=%ld, cbNeeded=%ld\n",splerr, cbNeeded) ;
       DosExit( EXIT_PROCESS , 0 ) ;
    }
    if (!DosAllocMem( &pBuf, cbNeeded,
                      PAG_READ|PAG_WRITE|PAG_COMMIT) ){
       cbBuf= cbNeeded ;
       splerr = SplQueryDevice(pszComputerName, pszPrintDeviceName,
                               ulLevel, pBuf, cbBuf, &cbNeeded) ;

       printf("SplQueryDevice Error=%ld, Bytes Needed=%ld\n", splerr,
                               cbNeeded) ;

       pprd3=(PPRDINFO3)pBuf;
```

```
        printf("Print Device info: name - %s\n", pprd3->pszPrinterName) ;
        printf("User Name      = %s\n", pprd3->pszUserName) ;
        printf("Logical Address= %s\n", pprd3->pszLogAddr) ;
        printf("Job ID         = %d\n", pprd3->uJobId) ;
        printf("Status         = %d\n", pprd3->fsStatus) ;
        printf("Status Comment = %s\n", pprd3->pszStatus) ;
        printf("Comment        = %s\n", pprd3->pszComment) ;
        printf("Drivers        = %s\n", pprd3->pszDrivers) ;
        printf("Time           = %d\n", pprd3->time) ;
        printf("Time Out       = %d\n", pprd3->usTimeOut) ;
        DosFreeMem(pBuf) ;
    }
    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}
```

SplQueryJob

This function retrieves information about a print job.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplQueryJob  (PSZ pszComputerName, PSZ pszQueueName,
                     ULONG ulJob, ULONG ulLevel, PPVOID pBuf,
                     ULONG cbBuf, PULONG pcbNeeded)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where print job is to be queried.

A NULL string specifies the local workstation.

pszQueueName (PSZ) – input
Queue Name.

ulJob (ULONG) – input
Job identification number.

ulLevel (ULONG) – input
Level of detail required.

This must be 0, 2, or 3.

pBuf (PPVOID) – in/out
Buffer.

The returned contents of the buffer depend on the value specify in *ulLevel* as follow:

ulLevel	**Buffer Contents**
0	The job identifier
2	A PRJINFO2 structure, with variable information, up to the *cbBuf* of *pBuf*
3	A PRJINFO3 structure, with variable information, up to the *cbBuf* of *pBuf*.

cbBuf (ULONG) – input
Size, in bytes, of Buffer.

It must be greater than 0.

pcbNeeded (PULONG) – output
Size in bytes of available information.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_JobNotFound (2151)	The print job does not exist.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_InvalidComputer (2351)	The computer name is invalid.

Related Functions

- SplEnumJob
- SplEnumQueue
- SplQueryQueue
- SplSetJob

Example Code

The following sample code will print out the information contained in a PRJINFO3 structure that is returned from a SplQueryJob call. The parameters that are entered on the command line are the computer name, queue name, and the job id.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>    /* for printf call  */
#include <stdlib.h>   /* for atoi call    */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
```

```
{
    INT       splerr;
    ULONG     cbBuf ;
    ULONG     cbNeeded ;
    ULONG     ulLevel ;
    ULONG     ulJob ;
    PSZ       pszComputerName ;
    PSZ       pszQueueName ;
    PVOID     pBuf;
    PPRJINFO3 pprj3 ;

    /* Input the parameters Computer Name, Queue Name,and Job ID. Check that  */
    /* three parameters have been entered along with the program name.        */
    if (argc != 4)
    {
       /* Print a message and exit if wrong number of parameters entered      */
       printf("Syntax:  sjqry  ComputerName  QueueName  JobID    \n");
       DosExit( EXIT_PROCESS , 0 ) ;
    }
    /* Set the parameters to the values entered on the command line.          */
    pszComputerName = argv[1] ;
    pszQueueName = argv[2] ;
    ulJob = atoi (argv[3]);

    /* Valid levels are 0,2,and 3. Level 3 returns a PRJINFO3 structure.      */
    ulLevel = 3 ;

    /* Call the function with cbBuf equal to zero in order to get the number  */
    /* of bytes needed returned in cbNeeded.                                  */
    splerr = SplQueryJob(pszComputerName,pszQueueName,ulJob,
                         ulLevel, (PVOID)NULL, 0L, &cbNeeded );

    /* Only continue if the error return code is one of the two following.    */
    if (splerr == NERR_BufTooSmall || splerr == ERROR_MORE_DATA )
    {
       /* Allocate memory for the buffer(pBuf). Only continue if memory is    */
       /* successfully allocated.                                             */
       if (DosAllocMem( &pBuf, cbNeeded,
                        PAG_READ|PAG_WRITE|PAG_COMMIT) )
       {
          /* Set the count of bytes needed for the buffer to the value        */
          /* returned in cbNeeded from the first call.                        */
          cbBuf = cbNeeded ;
```

```
        /* Make the call again with all the correct values.                */
        SplQueryJob(pszComputerName,pszQueueName,ulJob,
                            ulLevel, pBuf, cbBuf, &cbNeeded) ;

        /* Set a pointer to point to the beginning of the buffer that holds */
        /* the returned structure.                                          */
        pprj3=(PPRJINFO3)pBuf;

        /* Print out the information for each element in the structure.     */
        printf("Job ID      = %d\n", pprj3->uJobId);
        printf("Job Priority= %d\n", pprj3->uPriority);
        printf("User Name   = %s\n", pprj3->pszUserName);
        printf("Position    = %d\n", pprj3->uPosition);
        printf("Status      = %d\n", pprj3->fsStatus);
        printf("Submitted   = %ld\n",pprj3->ulSubmitted);
        printf("Size        = %ld\n",pprj3->ulSize);
        printf("Comment     = %s\n", pprj3->pszComment);
        printf("Document    = %s\n", pprj3->pszDocument);
        printf("Notify Name = %s\n", pprj3->pszNotifyName);
        printf("Data Type   = %s\n", pprj3->pszDataType);
        printf("Parms       = %s\n", pprj3->pszParms);
        printf("Status      = %s\n", pprj3->pszStatus);
        printf("Queue       = %s\n", pprj3->pszQueue);
        printf("QProc Name  = %s\n", pprj3->pszQProcName);
        printf("QProc Parms = %s\n", pprj3->pszQProcParms);
        printf("Driver Name = %s\n", pprj3->pszDriverName);
        printf("Printer Name= %s\n", pprj3->pszPrinterName);

        /* If pDriverData is NULL, then we can not access any data.         */
        if (pprj3->pDriverData)
        {
           printf("  pDriverData->cb           - %ld\n",
                              (ULONG)pprj3->pDriverData->cb);
           printf("  pDriverData->lVersion     - %ld\n",
                              (ULONG)pprj3->pDriverData->lVersion) ;
           printf("  pDriverData->szDeviceName - %s\n",
                                 pprj3->pDriverData->szDeviceName) ;
        }
        printf("/n");

        /* Free memory that we allocated.                                   */
        DosFreeMem(pBuf) ;
    }
}
```

```
    else
    {
       /* If we are here than we have an error code. Print it out.          */
       printf("SplQueryJob Error=%ld,Bytes Needed=%ld\n",splerr, cbNeeded);
    }
    DosExit( EXIT_PROCESS , 0 ) ;
    return(splerr);
}
```

SplQueryQueue

This function supplies information about a print queue, and, optionally, about the jobs in it.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplQueryQueue (PSZ pszComputerName, PSZ pszQueueName,
                      ULONG ulLevel, PVOID pBuf, ULONG cbBuf,
                      PULONG pcbNeeded)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where queue is to be queried.

A NULL string specifies the local workstation.

pszQueueName (PSZ) – input
Queue name.

ulLevel (ULONG) – input
Level of detail required.

This must be 3, 4, 5 or 6.

pBuf (PVOID) – in/out
Buffer.

The returned contents of the buffer depend on the value specified in *ulLevel* as follows:

ulLevel	**Buffer Contents**
3	A PRQINFO3 structure, with associated variable information up to *cbBuf*. The *fsType* bit PRQ3_TYPE_APPDEFAULT is set for the application default queue only.
4	A PRQINFO3 structure, with associated variable information, and an array of PRJINFO2 structures, one for each job in the queue, up to *cbBuf*.
5	A queue name of type PSZ.
6	A PRQINFO6 structure, with associated variable information up to *cbBuf*.

cbBuf (ULONG) – input
Size, in bytes, of Buffer.

It must be greater than 0.

pcbNeeded (PULONG) – output
Size in bytes of available information.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
ERROR_MORE_DATA (234)	Additional data is available.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

If *ulLevel* is 3 or 4, and *pBuf* cannot hold the entire PRQINFO3 structure, SplQueryQueue returns NERR_BufTooSmall (2123).

If *ulLevel* is 6, and *pBuf* cannot hold the entire PRQINFO6 structure, SplQueryQueue returns NERR_BufTooSmall (2123).

If *ulLevel* is 4, and *pBuf* cannot hold all the available PRJINFO2 structures, SplQueryQueue returns ERROR_MORE_DATA (234).

To obtain the size of buffer required, call SplQueryQueue with the required value of *ulLevel* and a NULL buffer. The number of bytes required is returned in *pcbNeeded*.

Related Functions

- SplEnumQueue
- SplQueryJob
- SplSetJob
- SplSetQueue

Example Code

This sample code queries the local workstation for a queue name that is entered at the command prompt. The query is done at level 4 which returns returns in the buffer information in a PRQINFO3 structure and follows this with PRJINFO2 structures - one for each job in the queue.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    ULONG  splerr  ;
    ULONG  cbBuf;
    ULONG  cbNeeded ;
    ULONG  ulLevel ;
    ULONG  i ;
    USHORT uJobCount ;
    PSZ    pszComputerName ;
    PSZ    pszQueueName ;
    PVOID  pBuf;
    PPRJINFO2 prj2 ;
    PPRQINFO3 prq3 ;

    if (argc != 2)
    {
       printf("Syntax:  setqryq  QueueName \n");
       DosExit( EXIT_PROCESS , 0 ) ;
    }

    pszComputerName = (PSZ)NULL ;
    pszQueueName = argv[1];
    ulLevel = 4L;
    splerr = SplQueryQueue(pszComputerName, pszQueueName, ulLevel,
                           (PVOID)NULL, 0L, &cbNeeded );
    if (splerr != NERR_BufTooSmall || splerr != ERROR_MORE_DATA )
    {
       printf("SplQueryQueue Error=%ld, cbNeeded=%ld\n",splerr, cbNeeded) ;
       DosExit( EXIT_PROCESS , 0 ) ;
    }
    if (!DosAllocMem( &pBuf, cbNeeded,
                  PAG_READ|PAG_WRITE|PAG_COMMIT) )
    {
       cbBuf = cbNeeded ;
       splerr = SplQueryQueue(pszComputerName, pszQueueName, ulLevel,
                              pBuf, cbBuf, &cbNeeded) ;
       prq3=(PPRQINFO3)pBuf;
```

```
printf("Queue info: name- %s\n", prq3->pszName) ;
if ( prq3->fsType & PRQ3_TYPE_APPDEFAULT )
   printf("  This is the application default print queue\n");
printf("  priority - %d  starttime - %d  endtime - %d fsType - %X\n",
       prq3->uPriority ,  prq3->uStartTime , prq3->uUntilTime ,
       prq3->fsType ) ;
printf("  pszSepFile    - %s\n", prq3->pszSepFile) ;
printf("  pszPrProc     - %s\n", prq3->pszPrProc) ;
printf("  pszParms      - %s\n", prq3->pszParms) ;
printf("  pszComment    - %s\n", prq3->pszComment) ;
printf("  pszPrinters   - %s\n", prq3->pszPrinters) ;
printf("  pszDriverName - %s\n", prq3->pszDriverName) ;
if (prq3->pDriverData)
{
  printf("  pDriverData->cb           - %ld\n",
                        (ULONG)prq3->pDriverData->cb);
  printf("  pDriverData->lVersion     - %ld\n",
                        (ULONG)prq3->pDriverData->lVersion) ;
  printf("  pDriverData->szDeviceName - %s\n",
                        prq3->pDriverData->szDeviceName) ;
}
/* Store the job count for use later in the for loop.                */
uJobCount = prq3->cJobs;
printf("Job count in this queue is %d\n\n",uJobCount);

/* Increment the pointer to the PRQINFO3 structure so that it points to*/
/* the first structure after itself.                                 */
prq3++;

/* Cast the prq3 pointer to point to a PRJINFO2 structure, and set a   */
/* pointer to point to that place.                                   */
prj2=(PPRJINFO2)prq3;
for (i=0 ; i<uJobCount ;i++ ) {
   printf("Job ID    = %d\n",  prj2->uJobId);
   printf("Priority  = %d\n",  prj2->uPriority);
   printf("User Name = %s\n",  prj2->pszUserName);
   printf("Position  = %d\n",  prj2->uPosition);
   printf("Status    = %d\n",  prj2->fsStatus);
   printf("Submitted = %ld\n", prj2->ulSubmitted);
```

```
            printf("Size      = %ld\n", prj2->ulSize);
            printf("Comment   = %s\n",  prj2->pszComment);
            printf("Document  = %s\n\n",prj2->pszDocument);

            /* Increment the pointer to point to the next structure.          */
            prj2++;
          } /* endfor */
          DosFreeMem(pBuf) ;
      }
      DosExit( EXIT_PROCESS , 0 ) ;
      return (splerr);
}
```

SplReleaseJob

This function releases a held print job.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplReleaseJob (PSZ pszComputerName, PSZ pszQueueName,
                      ULONG ulJob)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where job is to be continued.

A NULL string specifies the local workstation.

pszQueueName (PSZ) – input
Queue Name.

ulJob (ULONG) – input
Job identification number.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_JobNotFound (2151)	The print job does not exist.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_JobInvalidState (2164)	This operation cannot be performed on the print job in its current state.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

Any job can be released by a user with administrator privilege.

A job created locally can be released locally regardless of user privilege level, but it can be released remotely only by a user with administrator privilege.

A remote job can be released by a user without administrator privilege only if the user identification of the person initiating the request is the same as the user identification of the person who created the job.

Related Functions

- SplEnumJob
- SplHoldJob
- SplQueryJob

Example Code

This sample code will release the job id that is entered at the prompt.

```
#define INCL_BASE
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>       /* for printf function */
#include <stdlib.h>      /* for atoi function   */

INT main (argc, argv)
   INT argc;
   CHAR *argv[];
{
   SPLERR splerr ;
   ULONG  ulJob ;
   PSZ    pszComputerName = NULL ;
   PSZ    pszQueueName = NULL ;

   /* Get job id from the input argument  */
   ulJob = atoi(argv[1]);

   /* Call the function to do the release. If an error is returned, print it. */
   splerr=SplReleaseJob( pszComputerName, pszQueueName, ulJob);
   switch (splerr)
   {
   case NO_ERROR:
      printf("Job %d was released.\n",ulJob);
      break;
   case NERR_JobNotFound:
      printf("Job does not exist.\n");
      break;
   case  NERR_JobInvalidState:
      printf("This operation can't be performed on the print Job.\n");
      break;
   default:
      printf("Errorcode = %ld\n",splerr);
   } /* endswitch */
   DosExit( EXIT_PROCESS , 0 ) ;
   argc;
   return (splerr);
}
```

SplReleaseQueue

This function releases a held print queue.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplReleaseQueue  (PSZ pszComputerName, PSZ pszQueueName)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where queue is to be continued.

A NULL string specifies the local workstation.

pszQueueName (PSZ) – input
Queue name.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

This function releases a queue that has been held by a SplHoldQueue function, or disabled by an error on the queue. It does not affect an active print queue.

To release a queue on a remote server requires administrator privilege on the remote server.

Related Functions

- SplEnumQueue
- SplHoldQueue
- SplQueryQueue

Example Code

This sample code will release the local queue that is entered at the prompt.

```
#define INCL_SPL
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>     /* for printf function */

INT main (argc, argv)
   INT argc;
   CHAR *argv[];
{
   SPLERR splerr ;
   PSZ    pszComputerName = NULL ;
   PSZ    pszQueueName ;

   /* Get queue name from the input argument.                          */
   pszQueueName = argv[1];

   /* Call the function to do the release. If an error is returned, print it. */
   splerr=SplReleaseQueue(pszComputerName, pszQueueName);
   if (splerr != 0L)
   {
      switch (splerr)
      {
         case NERR_QNotFound:
            printf("Queue does not exist.\n");
            break;
         case  NERR_SpoolerNotLoaded:
            printf("The Spooler is not running.\n");
            break;
         default:
            printf("Errorcode = %ld\n",splerr);
      } /* endswitch */
   }
   else
   {
      printf("Queue %s was released.\n",pszQueueName);
   } /* endif */
   DosExit( EXIT_PROCESS , 0 ) ;
   return (splerr);
}
```

SplSetDevice

This function modifies the configuration of a print device.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplSetDevice  (PSZ pszComputerName, PSZ pszPrintDeviceName,
                      ULONG ulLevel, PVOID pBuf, ULONG cbBuf,
                      ULONG ulParmNum)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where print device is to be modified.

A NULL string specifies the local workstation.

pszPrintDeviceName (PSZ) – input
Name of Print Device.

ulLevel (ULONG) – input
Level of detail required.

This must be 3.

pBuf (PVOID) – input
Buffer.

If *ulParmNum* is 0, this parameter must contain a complete PRJINFO3 structure Otherwise,it must contain a valid new value for the parameter of the PRJINFO3 structure indicated in *ulParmNum*

cbBuf (ULONG) – input
Size, in bytes, of Buffer.

It must be grater than 0.

ulParmNum (ULONG) – input
Parameter number.

Specifies either that the entire PRDINFO3 structure is to be modified, or only one specific parameter is to be modified.

If *ulParmNum* is 0, *pBuf* must contain a complete PRDINFO3 structure. Otherwise, *pBuf* must contain a new valid value for the parameter to be modified.

The following are the possible values for this parameter:

Parameter	Constant (Value)
pszLogAddr	PRD_LOGADDR_PARMNUM (3)
pszComment	PRD_COMMENT_PARMNUM (7)
pszDrivers	PRD_DRIVERS_PARMNUM (8)
usTimeOut	PRD_TIMEOUT_PARMNUM (10)

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_DestNotFound (2152)	The print device cannot be found.
NERR_DestInvalidState (2162)	This operation cannot be performed on the print device.
NERR_SpoolNoMemory (2165)	A spooler memory allocation failure occurred.
NERR_DriverNotFound (2166)	The device driver does not exist.
NERR_BadDev (2341)	The requested device is invalid.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

This function allows modification of a print device and its connection to a logical address. To disconnect a print device from a port, use *ulLevel*=3, *ulParmNum*=3, and *pBuf* is a NULL string.

To modify a print device on a remote server requires administrator privilege.

Related Functions

- SplEnumDevice
- SplEnumDriver
- SplEnumPort
- SplEnumPrinter
- SplQueryDevice

Example Code

This sample code first gets a device name from the command line. It then prompts the user for a parameter number and a value associated with it.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>            /* for printf function */
#include <string.h>           /* for strlen function */
#include <stdlib.h>           /* for atoi function   */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    CHAR   bufValue[2]={0};
    CHAR   bufInput[128]={0};
    ULONG  splerr  ;
    ULONG  cbBuf ;
    ULONG  ulParmNum ;
    USHORT usParm;
    PSZ    pszComputerName ;
    PSZ    pszPrintDeviceName ;
    PVOID  pBuf;

    if (argc != 2)
    {
       printf("Syntax:  sdset  DeviceName  \n");
       DosExit( EXIT_PROCESS , 0 ) ;
    }
    pszComputerName = (PSZ)NULL ;

    /* Set the print device name to the value from the command line. */
    pszPrintDeviceName = argv[1];

    /* Get the parameter and the value. Store them in buffers. */
    printf("Enter Parameter number to be modified\n");
    gets(&bufValue[0]);
    printf("Enter new parameter value \n");
    gets(&bufInput[0]);

    /* Convert the input parmnum  to a ULONG. */
    ulParmNum = atoi(&bufValue[0]);
```

```
    switch (ulParmNum)
    {
       case 10:
          /* Determine the size of the buffer. */
          cbBuf = sizeof(PUSHORT);

          /* Convert the input parameter to a USHORT. */
          usParm =(USHORT)atoi(&bufInput[0]);

          /* Point the buffer to the value. */
          pBuf = &usParm;
          break;
       case 3:
       case 7:
       case 8:
          /* Determine the size of the buffer. */
          cbBuf = strlen(&bufInput[0])+1;

          /* Point the buffer to the value. */
          pBuf = (PSZ)&bufInput;
          break;
       default:
          printf("Invalid number\n");
          DosExit( EXIT_PROCESS , 0 ) ;
          break;
    }

    /* Make the call. */
    splerr = SplSetDevice(pszComputerName,pszPrintDeviceName,3L,
                          pBuf,cbBuf,ulParmNum) ;

    /* Print out the result. */
    printf("SplSetDevice Err= %ld, Parameter= %d, cbBuf= %ld ,ulParmNum= %ld\n",
                          splerr, usParm, cbBuf, ulParmNum);

    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}
```

SplSetJob

This function modifies the instructions for a print job.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplSetJob  (PSZ pszComputerName, PSZ pszQueueName, ULONG ulJob,
                   ULONG ulLevel, PVOID pBuf, ULONG cbBuf,
                   ULONG ulParmNum)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where job is to be modified.

A NULL string specifies the local workstation.

pszQueueName (PSZ) – input
Queue Name.

ulJob (ULONG) – input
Job identification number.

ulLevel (ULONG) – input
Level of detail required.

This must be 3.

pBuf (PVOID) – input
Buffer.

If *ulParmNum* is 0, this parameter must contain a complete PRJINFO3 structure. Otherwise,it must contain a valid new value for the parameter of the PRJINFO3 structure. indicated in *ulParmNum*

cbBuf (ULONG) – input
Size, in bytes, of Buffer.

It must be greater than 0.

ulParmNum (ULONG) – input
Parameter number.

Specifies either that the entire PRJINFO3 structure is to be modified, or that only one specific parameter is to be modified.

If *ulParmNum* is 0, *pBuf* must contain a complete PRJINFO3 structure. Otherwise, *pBuf* must contain a new valid value for the parameter to be modified. The following are the possible values for this parameter:

Parameter	Value
pszNotifyName	PRJ_NOTIFYNAME_PARMNUM (3)
pszDataType	PRJ_DATATYPE_PARMNUM (4)
pszParms	PRJ_PARMS_PARMNUM (5)
uPosition	PRJ_POSITION_PARMNUM (6)
pszComment	PRJ_COMMENT_PARMNUM (11)
pszDocument	PRJ_DOCUMENT_PARMNUM (12)
pszStatus	PRJ_STATUSCOMMENT_PARMNUM (13)
uPriority	PRJ_PRIORITY_PARMNUM (14)
pszQProcParms	PRJ_PROCPARMS_PARMNUM (16)
pDriverData	PRJ_DRIVERDATA_PARMNUM (18)

uPosition must be given the appropriate value, as follows:

Value	Change
0	No change
1	Move to first place
>1	Move to this position, or if the specified value is greater than the number of jobs in the queue, move to the end of the queue.

Returns

rc (SPLERR) – returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
NERR_NetNotStarted (2102)	The network program is not started.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_JobNotFound (2151)	The print job does not exist.
NERR_SpoolerNotLoaded (2161)	The spooler is not running.
NERR_JobInvalidState (2164)	This operation cannot be performed on the print job in its current state.
NERR_SpoolNoMemory (2165)	A spooler memory allocation failure occurred.
NERR_DriverNotFound (2166)	The device driver does not exist.
NERR_ProcNotFound (2168)	The queue processor is not installed.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

The job priority is changed, if necessary, to the priority of the next job after the new position. If the spooler is restarted, the order in which jobs are put on the queue depends on the priority and age of the job. This order may be different from the order following the SplSetJob call.

Users without administrator privilege can use SplSetJob only for jobs created when the same user name was logged on. They can move jobs backwards only, and cannot increase the job priority above the queue priority.

A job created locally has no associated user name, and any user can set information locally for the job. Only an administrator can set information for a job on a remote server.

Related Functions

- SplEnumJob
- SplQueryJob
- SplQueryQueue

Example Code

This sample code first gets a queue name and a jobid from the command prompt. It then prompts the user to enter a parameter number and a value for that number.

```
#define INCL_BASE
#define INCL_DOSMEMMGR
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS
#include <os2.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    CHAR   bufValue[2]={0};
    CHAR   bufInput[128]={0};
    ULONG  splerr  ;
    ULONG  cbBuf;
    ULONG  ulParmNum ;
    ULONG  ulJob ;
    USHORT usParm;
    PSZ    pszComputerName ;
    PSZ    pszQueueName ;
    PVOID  pBuf;
```

```
if (argc != 3)
{
   printf("Syntax:  sjset  QueueName  JobID  \n");
   DosExit( EXIT_PROCESS , 0 ) ;
}
pszComputerName = (PSZ)NULL ;

/* Set values to those entered at the prompt. */
pszQueueName = argv[1] ;
ulJob = atoi (argv[2]);

/* Request a parameter and the associated value. Store them in buffers. */
printf("Enter Parameter number to be modified\n");
gets(&bufValue[0]);
printf("Enter new parameter value \n");
gets(&bufInput[0]);

/* Convert the ParmNum to a ULONG. */
ulParmNum = atoi(&bufValue[0]);
switch (ulParmNum)
{
   case  6:
   case 14:
      /* Calculate size of buffer needed if this is the parameter.*/
      cbBuf = sizeof(PUSHORT);

      /* Convert input parameter into a USHORT. */
      usParm =(USHORT)atoi(&bufInput[0]);

      /* Point pBuf to the value. */
      pBuf = &usParm;
      break;
   case  3:
   case  4:
   case  5:
   case 11:
   case 12:
   case 16:
      /* Calculate size of buffer needed if this is the parameter.*/
      cbBuf = strlen(&bufInput[0])+1;

      /* Point pBuf to the value. */
      pBuf = (PSZ)&bufInput;
      break;
```

```
        case 18:
           printf("In order to keep code simple, this is not implemented.");
           break;
        default:
           printf("Invalid number\n");
     }

     splerr = SplSetJob(pszComputerName,pszQueueName,ulJob,3L,
                        pBuf,cbBuf,ulParmNum) ;
     if ( !splerr)
        printf("Parameter was set.");
     else
        printf("SplSetJob Error= %ld, Parameter= %d, Buf= %ld ,ParmNum= %ld\n",
                       splerr, usParm, cbBuf, ulParmNum);
     DosExit( EXIT_PROCESS , 0 ) ;
     return (splerr) ;
}
```

SplSetQueue

This function modifies the configuration of a print queue.

Syntax

```
#define INCL_SPL /* Or use INCL_PM, */
#include <os2.h>
SPLERR SplSetQueue  (PSZ pszComputerName, PSZ pszQueueName,
                     ULONG ulLevel, PVOID pBuf, ULONG cbBuf,
                     ULONG ulParmNum)
```

Parameters

pszComputerName (PSZ) – input
Name of computer where queue is to be modified.

A NULL string specifies the local workstation.

pszQueueName (PSZ) – input
Queue name.

ulLevel (ULONG) – input
Level of detail required.

This must be 3 or 6.

pBuf (PVOID) – input
Buffer.

If *ulParmNum* is 0, this parameter must contain a complete PRQINFO3 or PRQINFO6 structure. Otherwise, it must contain a valid new value for the parameter of the PRQINFO3 or the PRQINFO6 structure indicated in *ulParmNum*.

cbBuf (ULONG) – input
Size, in bytes, of Buffer.

ulParmNum (ULONG) – input
Parameter number.

Specifies either that the entire PRQINFO3 or PRQINFO6 structure is to be modified, or that only one specific parameter is to be modified.

Possible values for this parameter are as follows:

Parameter	**Constant (Value)**
uPriority	PRQ_PRIORITY_PARMNUM (2)
uStartTime	PRQ_STARTTIME_PARMNUM (3)
uUntilTime	PRQ_UNTILTIME_PARMNUM (4)
pszSepFile	PRQ_SEPARATOR_PARMNUM (5)

pszPrProc	PRQ_PROCESSOR_PARMNUM (6)
pszParms	PRQ_PARMS_PARMNUM (8)
pszComment	PRQ_COMMENT_PARMNUM (9)
fsType	PRQ_TYPE_PARMNUM (10)
pszPrinters	PRQ_PRINTERS_PARMNUM (12)
pszDriverName	PRQ_DRIVERNAME_PARMNUM (13)
pDriverData	PRQ_DRIVERDATA_PARMNUM (14)
pszRemoteComputerName	PRQ_REMOTE_COMPUTER_PARMNUM (15)
pszRemoteQueueName	PRQ_REMOTE_QUEUE_PARMNUM (16)

Returns

rc (SPLERR) -- returns
Return code.

NO_ERROR (0)	No errors occurred.
ERROR_ACCESS_DENIED (5)	Access is denied.
ERROR_NOT_SUPPORTED (50)	This request is not supported by the network.
ERROR_BAD_NETPATH (53)	The network path cannot be located.
ERROR_INVALID_PARAMETER (87)	An invalid parameter is specified.
ERROR_INVALID_LEVEL (124)	The level parameter is invalid.
NERR_NetNotStarted (2102)	The network program is not installed.
NERR_RedirectedPath (2117)	The operation is invalid on a redirected resource.
NERR_BufTooSmall (2123)	The API return buffer is too small.
NERR_QNotFound (2150)	The printer queue does not exist.
NERR_DestNotFound (2152)	The printer destination cannot be found.
NERR_DestNoRoom (2157)	The maximum number of printer destinations has been reached.
NERR_DestInvalidState (2162)	This operation cannot be performed on the print destination.
NERR_SpoolNoMemory (2165)	A spooler memory allocation failure occurred.
NERR_DriverNotFound (2166)	The device driver does not exist.
NERR_DataTypeInvalid (2167)	The datatype is not supported by the processor.
NERR_ProcNotFound (2168)	The queue processor is not installed.
NERR_BadDev (2341)	The requested device is invalid.
NERR_CommDevInUse (2343)	The requested device is invalid.
NERR_InvalidComputer (2351)	The computer name is invalid.

Remarks

If the *uPriority* field in PRQINFO3 or PRQINFO6 is set to PRQ_NO_PRIORITY, the queue priority is not changed.

Related Functions

- SplCreateQueue
- SplEnumDevice
- SplEnumDriver
- SplEnumQueue

- SplEnumQueueProcessor
- SplQueryQueue

Example Code

This sample code prompts the user to enter a parameter number and a value at the prompt. This value is then put into a buffer for use by the function.

```
#define INCL_SPL
#define INCL_SPLDOSPRINT
#define INCL_SPLERRORS

#include <os2.h>
#include <stdio.h>     /* for printf function */
#include <stdlib.h>    /* for atoi function   */
#include <string.h>    /* for strlen function */

INT main (argc, argv)
    INT argc;
    CHAR *argv[];
{
    CHAR   bufValue[2] = {0};
    CHAR   bufInput[128]= {0};
    ULONG  splerr ;
    ULONG  cbBuf ;
    ULONG  ulParmNum ;
    USHORT usParm ;
    PSZ    pszComputerName ;
    PSZ    pszQueueName ;
    PVOID  pBuf;

    if (argc != 2)
    {
       printf("Syntax:  setqryq  QueueName \n");
       DosExit( EXIT_PROCESS , 0 ) ;
    }

    /* This function will be for the local workstation.
    pszComputerName = (PSZ)NULL ;

    /* Get the parameter from the command line.                    */
    pszQueueName = argv[1];

    /* Prompt the user for the parameter and values, and put them in buffers. */
    printf("Enter Parameter number to be modified\n");
    gets(&bufValue[0]);
    printf("Enter new parameter value \n");
    gets(&bufInput[0]);
```

```
    /* Convert the ParmNum to a ULONG.                                    */
    ulParmNum = atoi(&bufValue[0]);
    switch (ulParmNum){
       case 2:
       case 3:
       case 4:
       case 10:
          /* Determine the size of the buffer needed.                    */
          cbBuf = sizeof(PUSHORT);

          /* Convert the buffer input to a USHORT.                       */
          usParm =(USHORT)atoi(&bufInput[0]);

          /* Set the pBuf pointer to point to the value obtained.        */
          pBuf = &usParm;
          break;
       case 5:
       case 6:
       case 8:
       case 9:
       case 12:
       case 13:
          /* Determine the size of the buffer needed.                    */
          cbBuf = strlen(&bufInput[0])+1;

          /* Set the pBuf pointer to point to the value obtained from input.  */
          pBuf = (PSZ)&bufInput;
          break;
       case 14:
          printf("For simplicity this is not implemented.");
          break;
       default:
          printf("Invalid number\n");
          DosExit( EXIT_PROCESS , 0 ) ;
          break;
    }

    /* Make the call with all the proper parameters.                      */
    splerr = SplSetQueue(pszComputerName, pszQueueName, 3L,
                         pBuf, cbBuf, ulParmNum) ;

    /* Print the resultant error code, and the parameters entered.        */
    printf("SplSetQueue Error= %ld, Parameter= %d, cbBuf= %ld,
                        ulParmNum= %ld\n",
                             splerr, usParm, cbBuf, ulParmNum);
    DosExit( EXIT_PROCESS , 0 ) ;
    return (splerr);
}
```

Chapter 7. Window Functions

Window Functions by Functional Area

The following table shows how all of the Window (WIN) functions are related within functional areas. The functions are in alphabetic order within these areas.

C Name	C Name
Accelerators	
WinCopyAccelTable	WinQueryAccelTable
WinCreateAccelTable	WinSetAccelTable
WinDestroyAccelTable	WinTranslateAccel
WinLoadAccelTable	
Alarms	
WinAlarm	WinMessageBox
WinFlashWindow	WinMessageBox2
Atom Manager	
WinAddAtom	WinQueryAtomLength
WinCreateAtomTable	WinQueryAtomName
WinDeleteAtom	WinQueryAtomUsage
WinDestroyAtomTable	WinQuerySystemAtomTable
WinFindAtom	
Button	
WinCheckButton	WinQueryButtonCheckstate
Clipboard	
WinCloseClipbrd	WinQueryClipbrdOwner
WinEmptyClipbrd	WinQueryClipbrdViewer
WinEnumClipbrdFmts	WinSetClipbrdData
WinOpenClipbrd	WinSetClipbrdOwner
WinQueryClipbrdData	WinSetClipbrdViewer
WinQueryClipbrdFmtInfo	
Coordinate Mapping	
WinMakePoints	WinMapWindowPoints
WinMapDlgPoints	
Cursor	
WinCreateCursor	WinQueryCursorInfo
WinDestroyCursor	WinShowCursor

C Name	C Name
Dialog Boxes	
WinCreateDlg	WinFileDlg
WinDefDlgProc	WinFontDlg
WinDefFileDlgProc	WinFreeFileDlgList
WinDefFontDlgProc	WinGetDlgMsg
WinDismissDlg	WinIsControlEnabled
WinDlgBox	WinLoadDlg
WinEnableControl	WinProcessDlg
Drawing Management	
Drawing	
WinBeginPaint	WinLockVisRegions
WinEnableWindowUpdate	WinOpenWindowDC
WinEndPaint	WinQueryUpdateRect
WinExcludeUpdateRegion	WinQueryUpdateRegion
WinGetClipPS	WinRealizePalette
WinGetPS	WinReleasePS
WinGetScreenPS	WinShowWindow
WinInvalidateRect	WinUpdateWindow
WinInvalidateRegion	WinValidateRect
WinIsWindowShowing	WinValidateRegion
WinIsWindowVisible	
Drawing Helpers	
WinDrawBitmap	WinInvertRect
WinDrawBorder	WinQueryPresParam
WinDrawPointer	WinRemovePresParam
WinDrawText	WinScrollWindow
WinFillRect	WinSetPresParam
WinGetSysBitmap	
DSOM Support	
WinIsSOMDDReady	WinRestartSOMDD
WinIsWPDServerReady	WinRestartWPSDServer
Dynamic Data Exchange (DDE) Support	
WinDdeInitiate	WinDdeRespond
WinDdePostMsg	

C Name	C Name
Error Processing	
WinFreeErrorInfo	WinGetLastError
WinGetErrorInfo	
File Icons	
WinFreeFileIcon	WinSetFileIcon
WinLoadFileIcon	
Help Manager	
WinAssociateHelpInstance	WinDestroyHelpInstance
WinCreateHelpInstance	WinLoadHelpTable
WinCreateHelpTable	WinQueryHelpInstance
Initialization and Termination	
WinInitialize	WinQueryVersion
WinQueryAnchorBlock	WinTerminate
Keyboard	
WinCheckInput	WinIsPhysInputEnabled
WinEnablePhysInput	WinQueryFocus
WinFocusChange	WinSetFocus
WinGetKeyState	WinSetKeyboardStateTable
WinGetPhysKeyState	
Library Support	
WinDeleteLibrary	WinLoadLibrary
WinDeleteProcedure	WinLoadProcedure
List Box	
WinDeleteLboxItem	WinQueryLboxItemTextLength
WinInsertLboxItem	WinQueryLboxSelectedItem
WinQueryLboxCount	WinSetLboxItemText
WinQueryLboxItemText	
Menus	
WinCheckMenuItem	WinIsMenuItemValid
WinCreateMenu	WinLoadMenu
WinEnableMenuItem	WinPopupMenu
WinIsMenuItemChecked	WinSetMenuItemText
WinIsMenuItemEnabled	
Message Management	
WinBroadcastMsg	WinQueryQueueStatus

C Name	C Name
WinCreateMsgQueue	WinRegisterUserDatatype
WinDestroyMsgQueue	WinRegisterUserMsg
WinDispatchMsg	WinRequestMutexSem
WinGetMsg	WinSendDlgItemMsg
WinInSendMsg	WinSendMsg
WinLoadMessage	WinSetClassMsgInterest
WinPeekMsg	WinSetMsgInterest
WinPostMsg	WinSetMsgMode
WinPostQueueMsg	WinSetSynchroMode
WinQueryMsgPos	WinWaitEventSem
WinQueryMsgTime	WinWaitMsg
WinQueryQueueInfo	WinWaitMuxWaitSem
Mouse Capture	
WinQueryCapture	WinSetCapture
Mouse Tracking	
WinShowTrackRect	WinTrackRect
Object Management	
WinCreateObject	WinQueryObject
WinDeregisterObjectClass	WinRegisterObjectClass
WinDestroyObject	WinReplaceObjectClass
WinEnumObjectClass	WinSetObjectData
Pointer	
WinCreatePointer	WinQuerySysPointer
WinCreatePointerIndirect	WinQuerySysPointerData
WinDestroyPointer	WinSetPointer
WinLoadPointer	WinSetPointerOwner
WinLockPointerUpdate	WinSetPointerPos
WinQueryPointer	WinSetSysPointerData
WinQueryPointerInfo	WinShowPointer
WinQueryPointerPos	
Rectangles	
WinCopyRect	WinOffsetRect
WinEqualRect	WinPtInRect
WinInflateRect	WinSetRect
WinIntersectRect	WinSetRectEmpty

C Name	C Name
WinIsRectEmpty	WinSubtractRect
WinMakeRect	WinUnionRect
Standard Window	
WinCalcFrameRect	WinCreateStdWindow
WinCreateFrameControls	
String/Character and Code Pages	
WinCompareStrings	WinQueryCp
WinCpTranslateChar	WinQueryCpList
WinCpTranslateString	WinSetCp
WinLoadString	WinSubstituteStrings
WinNextChar	WinUpper
WinPrevChar	WinUpperChar
Task List	
WinAddSwitchEntry	WinQueryTaskSizePos
WinChangeSwitchEntry	WinQueryTaskTitle
WinCreateSwitchEntry	WinRemoveSwitchEntry
WinQuerySessionTitle	WinStartApp
WinQuerySwitchEntry	WinSwitchToProgram
WinQuerySwitchHandle	WinTerminateApp
WinQuerySwitchList	
System and Queue Hooks	
WinCallMsgFilter	WinSetHook
WinReleaseHook	
System Management	
Locking	
WinLockupSystem	WinUnlockupSystem
System Colors	
WinQuerySysColor	WinSetSysColors
System Modal Management	
WinQuerySysModalWindow	WinSetSysModalWindow
System Shutdown	
WinCancelShutdown	WinShutdownSystem
System Values	
WinQuerySysValue	WinSetSysValue
Timers	

C Name	C Name
WinGetCurrentTime	WinStopTimer
WinStartTimer	
Window Management	
Activation, Size and Position	
WinGetMaxPosition	WinSetActiveWindow
WinGetMinPosition	WinSetMultWindowPos
WinQueryActiveWindow	WinSetWindowBits
WinQueryWindowPos	WinSetWindowPos
WinRestoreWindowPos	WinStoreWindowPos
WinSaveWindowPos	
Creation and Class Information	
WinCreateWindow	WinQueryClassName
WinDefWindowProc	WinRegisterClass
WinDestroyWindow	WinSubclassWindow
WinQueryClassInfo	
General Window Information	
WinEnableWindow	WinQueryWindowDC
WinIsThreadActive	WinQueryWindowProcess
WinIsWindow	WinQueryWindowRect
WinIsWindowEnabled	WinSetDesktopBkgnd
WinQueryDesktopBkgnd	WinWindowFromDC
WinQueryDesktopWindow	WinWindowFromID
WinQueryObjectWindow	WinWindowFromPoint
Locking	
WinLockWindowUpdate	WinUnlockWindow
Window Hierarchies	
WinBeginEnumWindows	WinMultWindowFromIDs
WinEndEnumWindows	WinQueryWindow
WinEnumDlgItem	WinSetOwner
WinGetNextWindow	WinSetParent
WinIsChild	
Mixed Memory Model Support	
WinQueryClassThunkProc	WinSetClassThunkProc
WinQueryWindowModel	WinSetWindowThunkProc
WinQueryWindowThunkProc	

C Name	C Name
Window Text	
WinQueryDlgItemShort	WinQueryWindowTextLength
WinQueryDlgItemText	WinSetDlgItemShort
WinQueryDlgItemTextLength	WinSetDlgItemText
WinQueryWindowText	WinSetWindowText
Window Words	
WinQueryWindowPtr	WinSetWindowPtr
WinQueryWindowULong	WinSetWindowULong
WinQueryWindowUShort	WinSetWindowUShort
WinSetWindowBits	

WinAddAtom

This function adds an atom to an atom table.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ATOM WinAddAtom  (HATOMTBL hatomtblAtomTbl, PSZ AtomName)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

This is the handle returned by a previous call to WinCreateAtomTable or WinQuerySystemAtomTable.

AtomName (PSZ) – input
Atom name.

This is a character string to be added to the table.

If the string begins with an "#" character, the five ASCII digits that follow are converted into an integer atom. If this integer is a valid integer atom, this function returns that atom, without modifying the atom table.

If the string begins with an "!" character, the next two bytes are interpreted as an atom. If it is an integer atom, that atom is returned. If it is not an integer atom and it is a valid atom for the given atom table (that is, it has an atom name and use count associated with it) the use count of that atom is incremented by one and the atom is returned. Otherwise 0 is returned.

If the high order word of the string is minus one, the low order word is an atom. If it is an integer atom, that atom is returned. If it is not an integer atom and it is a valid atom for the given atom table (that is, it has an atom name and use count associated with it) the use count of that atom is incremented by one and the atom is returned. Otherwise 0 is returned.

Returns

atom (ATOM) – returns
Atom value.

Atom	The atom associated with the passed string
0	Invalid atom-table handle or invalid atom name specified.

Possible returns from WinGetLastError

PMERR_INVALID_HATOMTBL (0x1013)	An invalid atom-table handle was specified.
PMERR_INVALID_INTEGER_ATOM (0x1016)	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME (0x1015)	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND (0x1017)	The specified atom name is not in the atom table.

Remarks

If the atom name represents an integer atom, this function returns the atom represented by the passed atom name.

If the atom name does not represent an integer atom and if the atom name already exists in the atom table, this function increments the use count of that atom by one. Otherwise, the atom is added to the table and its use count is set to one. In either case this function returns the atom represented by the passed atom name.

Related Functions

- WinCreateAtomTable
- WinDeleteAtom
- WinDestroyAtomTable
- WinFindAtom
- WinQueryAtomLength
- WinQueryAtomName
- WinQueryAtomUsage
- WinQuerySystemAtomTable

Example Code

This example creates an Atom Table and then adds the atom "newatom" to the new table; it then checks the count for this new atom to verify that it is 1.

```
#define INCL_WINATOM            /* Window Atom Functions        */
#include <os2.h>

ATOM  atom;             /* new atom value                       */
HATOMTBL  hatomtblAtomTbl; /* atom-table handle                 */
char   pszAtomName[10]; /* atom name                            */
ULONG   ulInitial = 0;  /* initial atom table size (use default)*/
ULONG   ulBuckets = 0;  /* size of hash table (use default)     */
ULONG   ulCount;        /* atom usage count                     */
BOOL atomCount1 = FALSE;/* indicates atom count != 1            */

/* create atom table of default size */
hatomtblAtomTbl = WinCreateAtomTable(ulInitial, ulBuckets);

/* define name for new atom and add to table */
strcpy(pszAtomName,"newatom");
atom = WinAddAtom(hatomtblAtomTbl, pszAtomName);

ulCount = WindQueryAtomUsage(hatomtblAtomTbl, atom);

/* verify that usage count is 1 */
if (ulCount == 1)
   atomCount1 = TRUE;
```

WinAddSwitchEntry

This function adds an entry to the Window List. This is a list of running programs that is displayed to the user by the operating system.

Syntax

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
HSWITCH WinAddSwitchEntry  (SWCNTRL swctlSwitchData)
```

Parameters

swctlSwitchData (SWCNTRL) – input
Switch data.

Contains information about the newly created Window List entry.

If the *szSwtitle* field of the SWCNTRL structure is 0, the system uses the name under which the application is started. This only applies for the first call to this function since the program started. Otherwise, a NULL entry name is invalid.

The title is truncated, if necessary, to 60 characters.

If the *hprog* field of the SWCNTRL structure is NULLHANDLE, the value used by the system when the program was loaded (if it has been loaded) is substituted.

If the *idProcess* field of the SWCNTRL structure is 0, the current process ID is used.

If the *idSession* field of the SWCNTRL structure is 0, the current session ID is used.

If the *hwndIcon* field of the SWCNTRL structure is NULLHANDLE, the system supplies a default icon.

Returns

hswitchSwitch (HSWITCH) – returns
Handle to the newly created Window List entry.

There is a system limit to the number of Window List entries. However, this is a large number (several hundred) and is unlikely to be reached in practice because other system limits, such as memory size, restrict the number before this limit is reached.

NULLHANDLE	Error occurred
Other	Handle to the newly created Window List entry.

Possible returns from WinGetLastError

PMERR_NO_SPACE (0x1201)	The limit on the number of Window List entries has been reached with WinAddSwitchEntry.

PMERR_INVALID_WINDOW (0x1206)	The window specified with a Window List call is not a valid frame window.
PMERR_INVALID_SESSION_ID (0x120B)	The specified session identifier is invalid. Either zero (for the application's own session) or a valid identifier must be specified.

Remarks

Neither this function nor the WinRemoveSwitchEntry function are required if the main window is created with the frame creation flags FCF_TASKLIST or FCF_STANDARD, because these flags automatically update the Window List when the main window is created or destroyed.

Related Functions

- WinChangeSwitchEntry
- WinCreateSwitchEntry
- WinQuerySessionTitle
- WinQuerySwitchEntry
- WinQuerySwitchHandle
- WinQuerySwitchList
- WinQueryTaskSizePos
- WinQueryTaskTitle
- WinRemoveSwitchEntry
- WinSwitchToProgram

Example Code

This example calls WinQueryWindowProcess to get the current process identifier (needed for the SWCNTRL structure). It then sets up the swctl structure and calls WinAddSwitchEntry to add the name of the program 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 the application will be calling the WinChangeSwitchEntry function to avoid having to set up the structure again.

```
#define INCL_WINSWITCHLIST      /* Window Task Switch Functions */
#include <os2.h>

SWCNTRL swctl;          /* switch control data                  */
HSWITCH hswitch;        /* switch handle                        */
PID pid;                /* process id                           */
HWND  hwndFrame;        /* frame handle                         */

WinQueryWindowProcess(hwndFrame, &pid, NULL);

swctl.hwnd = hwndFrame;                 /* window handle      */
swctl.hwndIcon = NULLHANDLE;            /* icon handle        */
swctl.hprog = NULLHANDLE;               /* program handle     */
swctl.idProcess = pid;                  /* process identifier */
swctl.idSession = 0;                    /* session identifier */
swctl.uchVisibility = SWL_VISIBLE;      /* visibility         */
swctl.fbJump = SWL_JUMPABLE;            /* jump indicator     */
swctl.szSwtitle[0] = 0;                 /* program name       */
swctl.bProgType = PROG_DEFAULT;         /* program type       */

hswitch = WinAddSwitchEntry(&swctl);
```

WinAlarm

This function generates an audible alarm.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinAlarm  (HWND hwndDeskTop, ULONG flStyle)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop window
Other	Specified desktop window.

flStyle (ULONG) – input
Alarm style.

Used to signify different situations to the operator.

The duration and frequency of the alarms can be changed by the WinSetSysValue function. The alarm frequency is defined to be in the range 0x0025 through 0x7FFF. The alarm is not generated if system value SV_ALARM is set to FALSE. The alarms are dependent on the device capability. Different alarms are selected by use of these values:

WA_WARNING
WA_NOTE
WA_ERROR

Returns

rc (BOOL) – returns
Alarm-generated indicator.

TRUE	Alarm generated
FALSE	Alarm not generated.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_PARM (0x1303)	A parameter to the function contained invalid data.

Remarks

Although this function is in the INCL_WINDIALOGS section, it is part of the common subset and is also included if INCL_COMMON is defined.

Related Functions

- WinFlashWindow
- WinMessageBox

Example Code

This example calls an application-defined initialization function, and calls WinAlarm to generate an audible alarm to notify the user if the function fails.

```
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#include <os2.h>

if (!GenericInit())                 /* general initialization */
    WinAlarm(HWND_DESKTOP, WA_ERROR);
```

WinAssociateHelpInstance

This function associates the specified instance of the Help Manager with the window chain of the specified application window.

Syntax

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinAssociateHelpInstance (HWND hwndHelpInstance, HWND hwndApp)
```

Parameters

hwndHelpInstance (HWND) – input
Handle of an instance of the Help Manager.

This is the handle returned by the WinCreateHelpInstance call.

NULLHANDLE	Disassociates an instance of the Help Manager from a window chain when the instance has been destroyed.
Other	The handle of an instance of the Help Manager to be associated with the application window chain.

hwndApp (HWND) – input
Handle of an application window.

The handle of the application window with which the instance of the Help Manager will be associated. The instance of the Help Manager is associated with the application window and any of its children or owned windows.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

In order to provide help, the application must associate an instance of the Help Manager with a chain of application windows. This association lets the Help Manager know which instance should provide the help function.

The Help Manager traces the window chain, starting from the window where help is requested. The application window in the chain with the associated help instance will be the one with which the Help Manager communicates and next to which the help window is positioned, unless a HM_SET_ACTIVE_WINDOW message is sent to the Help Manager. If

the HM_SET_ACTIVE_WINDOW message is sent to the Help Manager, the active window parameter is the window with which the Help Manager communicates. The Help Manager positions the help window next to the window specified as the relative window.

Related Functions

- WinCreateHelpInstance
- WinCreateHelpTable
- WinDestroyHelpInstance
- WinLoadHelpTable
- WinQueryHelpInstance

Related Messages

- HM_SET_ACTIVE_WINDOW

Example Code

This example shows a typical main function for an application which uses help. Following creation of the main application window the Help Manager is initialized and associated with the window. The help table is defined in the application's resources. When the window is destroyed, terminating the application, the help instance is also destroyed.

```
#define INCL=_WIN
#include <os2.h>

#define IDHT_APPLICATION        100     /* id of HELP TABLE in resource file */

main( int argc, char *argv[], char *envp[] )
{
   HAB  hab = WinInitialize( 0 );
   HMQ  hmq = WinCreateMsgQueue( hab, 0 );
   HWND hwnd;
   HWND hwndClient;
   HWND hwndHelp;
   QMSG qmsg;
   ULONG flStyle;
   HELPINIT helpinit;

   /* Setup the help initialization structure */
   helpinit.cb = sizeof( HELPINIT );
   helpinit.ulReturnCode =  0L;
   helpinit.pszTutorialName =  (PSZ)NULL;
   /* Help table in application resource */
   helpinit.phtHelpTable = (PHELPTABLE)MAKEULONG( IDHT_APPLICATION, 0xffff );
   helpinit.hmodHelpTableModule = NULLHANDLE;
```

the HM_SET_ACTIVE_WINDOW message is sent to the Help Manager, the active window parameter is the window with which the Help Manager communicates. The Help Manager positions the help window next to the window specified as the relative window.

Related Functions

- WinCreateHelpInstance
- WinCreateHelpTable
- WinDestroyHelpInstance
- WinLoadHelpTable
- WinQueryHelpInstance

Related Messages

- HM_SET_ACTIVE_WINDOW

Example Code

This example shows a typical main function for an application which uses help. Following creation of the main application window the Help Manager is initialized and associated with the window. The help table is defined in the application's resources. When the window is destroyed, terminating the application, the help instance is also destroyed.

```
#define INCL=_WIN
#include <os2.h>

#define IDHT_APPLICATION        100     /* id of HELP TABLE in resource file */

main( int argc, char *argv[], char *envp[] )
{
   HAB  hab = WinInitialize( 0 );
   HMQ  hmq = WinCreateMsgQueue( hab, 0 );
   HWND hwnd;
   HWND hwndClient;
   HWND hwndHelp;
   QMSG qmsg;
   ULONG flStyle;
   HELPINIT helpinit;

   /* Setup the help initialization structure */
   helpinit.cb = sizeof( HELPINIT );
   helpinit.ulReturnCode =  0L;
   helpinit.pszTutorialName =  (PSZ)NULL;
   /* Help table in application resource */
   helpinit.phtHelpTable = (PHELPTABLE)MAKEULONG( IDHT_APPLICATION, 0xffff );
   helpinit.hmodHelpTableModule = NULLHANDLE;
```

```
   /* Default action bar and accelerators */
   helpinit.hmodAccelActionBarModule = NULLHANDLE;
   helpinit.idAccelTable = 0;
   helpinit.idActionBar = 0;
   helpinit.pszHelpWindowTitle = "APPNAME HELP";
   helpinit.fShowPanelId = CMIC_SHOW_PANEL_ID;
   helpinit.pszHelpLibraryName = "APPNAME.HLP";

   /* Register the class */
   if( WinRegisterClass( ... ) )
   {
      /* create the main window */
      flStyle = FCF_STANDARD;
      hwnd = WinCreateStdWindow( ... );

      if( hwnd )
      {
         /* Create and associate the help instance */
         hwndHelp = WinCreateHelpInstance( hab, &helpinit );

         if( hwndHelp && WinAssociateHelpInstance( hwndHelp, hwnd ) )
         {
            /* Process messages */
            while( WinGetMsg( hab, &qmsg, NULLHANDLE, 0, 0 ) )
            {
               WinDispatchMsg( hab, &qmsg );
            } /* endwhile */
         }

         /* Remove help instance - note: add                    */
         /*     WinAssociateHelpInstance( NULLHANDLE, hwnd );      */
         /* to WM_DESTROY processing to remove the association. */
         WinDestroyHelpInstance( hwndHelp );
      }
   }

   /* finish the cleanup and exit */
   WinDestroyMsgQueue( hmq );
   WinTerminate( hab );
}
```

WinBeginEnumWindows

This function begins the enumeration process for all of the immediate child windows of a specified window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HENUM WinBeginEnumWindows (HWND hwnd)
```

Parameters

hwnd (HWND) – input
Handle of the window whose child windows are to be enumerated.

HWND_DESKTOP	Enumerate all main windows
HWND_OBJECT	Enumerate all object windows
Other	Enumerate all immediate children of the specified window.

Returns

henumHenum (HENUM) – returns
Enumeration handle.

This is used in subsequent calls to the WinGetNextWindow function to return the immediate child-window handles in succession.

When the application has finished the enumeration, the enumeration handle must be destroyed with the WinEndEnumWindows call.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function remembers the window hierarchy at the time of invocation of the call. Thereafter the information is referenced by use of the *henumHenum* parameter and does not change during the enumeration by the WinGetNextWindow call. The windows are enumerated in the z-order at the time enumeration is begun, with the topmost child window enumerated first.

Only the immediate children of the specified window are enumerated; child windows of the child windows are excluded.

The enumerated windows are not locked by this function and can thus be destroyed between the time that it is called and the time that the WinGetNextWindow function is used to obtain the handle for the window.

Related Functions

- WinEndEnumWindows
- WinEnumDlgItem
- WinGetNextWindow
- WinIsChild
- WinMultWindowFromIDs
- WinQueryWindow
- WinSetOwner
- WinSetParent

Example Code

This example begins window enumeration of all main windows (i.e. all immediate children of the Desktop), after which WinGetNextWindow is called in a loop to enumerate all the children, until all children are found and the enumeration ends.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND  hwndParent;       /* Handle of the window whose child windows
                           are to be enumerated                 */
HWND  hwndNext;         /* current enumeration handle           */
HENUM  henum;           /* enumeration handle                   */
BOOL  fSuccess;         /* success indicator                    */
SHORT sRetLen;          /* returned string length               */
SHORT sLength = 10;     /* string buffer length                 */
char  pchBuffer[10];    /* string buffer                        */

hwndParent = HWND_DESKTOP;

henum = WinBeginEnumWindows(hwndParent);

while ((hwndNext = WinGetNextWindow(henum)) != NULLHANDLE) {
  .
  .
  .
}

fSuccess = WinEndEnumWindows (henum);
```

WinBeginPaint

This function obtains a presentation space whose associated update region is set ready for drawing in a specified window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
HPS WinBeginPaint (HWND hwnd, HPS hps, PRECTL prclPaint)
```

Parameters

hwnd (HWND) – input
Handle of window where drawing is going to occur.

HWND_DESKTOP	The desk top window.
Other	Specified window.

hps (HPS) – input
Presentation-space handle.

NULLHANDLE	Obtain a cache presentation space.
Other	Presentation-space handle. This function sets its clipping region to the update region of the *hwnd* parameter.

prclPaint (PRECTL) – output
Bounding rectangle.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT can also be used, if supported by the language.

NULL	No bounding rectangle; that is, there is no need of changing any point.
Other	Specifies the smallest rectangle bounding the update region, in window coordinates.

Returns

hpsPaintPS (HPS) – returns
Presentation-space handle.

NULLHANDLE	Error occurred
Other	Presentation-space handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

PMERR_INV_HPS (0x207F)	An invalid presentation-space handle was specified.

Remarks

This function is generally call during the processing of a WM_PAINT message when the application needs to update the content of the window.

If the presentation space already exists, its update region is set and the device context of the window is associated with the presentation space. Otherwise, a cache presentation space is obtained specifically for the window.

The update region associated with *hwnd* is reset to NULLHANDLE. It is assumed that any drawing following this function restores the content of the window to a fully correct state.

This function hides the pointer if it is in the window and the WinEndPaint function restores it.

This function hides the tracking rectangle if it is active and might hide part of the painting window; that is, if *hwnd* is a child of the window specified by the *hwnd* parameter in the WinTrackRect function. The WinEndPaint function shows it again.

WinEndPaint must be called after the application completes drawing, and can be nested for the same window.

Related Functions

- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Related Messages

- WM_PAINT

Example Code

This example uses WinBeginPaint to obtain and associate a presentation space with the update region of a window so that redrawing can take place.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwnd;           /* parent window                        */
RECTL  rcl;             /* update region                        */
HPS    hps;             /* presentation-space handle            */

 case WM_PAINT:
    hps = WinBeginPaint(hwnd,   /* handle of the window      */
        NULLHANDLE,             /* get a cache presentation space */
        &rcl);                  /* receives update rectangle  */
    WinFillRect(hps, &rcl, CLR_WHITE);
    WinEndPaint(hps);
```

WinBroadcastMsg

This function broadcasts a message to multiple windows.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinBroadcastMsg (HWND hwndParent, ULONG ulMsgId,
                      MPARAM mpParam1, MPARAM mpParam2,
                      ULONG flCmd)
```

Parameters

hwndParent (HWND) – input
Parent-window handle.

ulMsgId (ULONG) – input
Message identifier.

mpParam1 (MPARAM) – input
Parameter 1.

mpParam2 (MPARAM) – input
Parameter 2.

flCmd (ULONG) – input
Broadcast message command.

BMSG_POST	Post the message. This value is mutually exclusive with BMSG_SEND and BMSG_POSTQUEUE.
BMSG_SEND	Send the message. This value is mutually exclusive with BMSG_POST and BMSG_POSTQUEUE.
BMSG_POSTQUEUE	Post a message to all threads that have a message queue. This value is mutually exclusive with BMSG_POST and BMSG_SEND. The *hwnd* parameter of the QMSG structure is set to NULL.
BMSG_DESCENDANTS	Broadcast the message to all the descendants of the *hwndParent* parameter.
BMSG_FRAMEONLY	Broadcast the message only to windows with a style of CS_FRAME.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Message was sent or posted successfully to all applicable windows
FALSE	Error occurred.

Remarks

This function sends or posts a message to all the immediate child windows of *hwndParent*, except in the case when *flCmd* is BMSG_DESCENDANTS.

The *ulMsgId*, *mpParam1*, and *mpParam2* parameters make up the message sent or posted. The window handle of the receiving window is added to the message.

Related Functions

- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example broadcasts a WM_CLOSE message to all descendants of the specified window.

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#include <os2.h>

BOOL  fSuccess;          /* Success indicator                   */
HWND  hwndParent;        /* parent window handle                */
ULONG   ulMsgId;         /* Message identifier                  */
MPARAM  mpParam1;        /* Parameter 1                         */
MPARAM  mpParam2;        /* Parameter 2                         */
ULONG   flCmd;           /* message command                     */

/* set msg to close window, parameters to NULL */
ulMsgId = WM_CLOSE;
mpParam1 = MPVOID;
mpParam2 = MPVOID;

/* broadcast to all descendants */
flCmd = BMSG_DESCENDANTS;

fSuccess = WinBroadcastMsg(hwndParent, ulMsgId, mpParam1,
                           mpParam2, flCmd);
```

WinCalcFrameRect

This function calculates a client rectangle from a frame rectangle, or a frame rectangle from a client rectangle.

Syntax

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinCalcFrameRect (HWND hwndFrame, PRECTL prcl, BOOL fClient)
```

Parameters

hwndFrame (HWND) – input
Frame-window handle.

prcl (PRECTL) – in/out
Window rectangle.

Note: The value of each field in this structure must be in the range -32768 through 32767. The data type WRECT can also be used, if supported by the language.

fClient (BOOL) – input
Frame indicator.

TRUE	Frame rectangle provided
FALSE	Client-area rectangle provided.

Returns

rc (BOOL) – returns
Rectangle-calculated indicator.

TRUE	Rectangle successfully calculated
FALSE	Error occurred, or the calculated rectangle is empty.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function provides the size and position of the client area within the specified frame rectangle for the specified frame window, or conversely, the size and position of the frame window that would contain a client window of the specified size and position.

This function sends a WM_CALCFRAMERECT message to the frame window. This enables a subclassed frame control to implement the calculation correctly.

This function works if *hwndFrame* is hidden; *hwndFrame* should be hidden if it is required that the window shows a particular client rectangle when the window is first shown.

Related Functions

- WinCreateFrameControls
- WinCreateStdWindow
- WinCreateWindow
- WinDefWindowProc
- WinDestroyWindow
- WinQueryClassInfo
- WinQueryClassName
- WinRegisterClass
- WinSubclassWindow

Related Messages

- WM_CALCFRAMERECT

Example Code

This example converts a client window's boundaries into screen coordinates and calls WinCalcFrameRect to calculate an equivalent frame rectangle size.

```
#define INCL_WINFRAMEMGR        /* Window Frame Functions      */
#define INCL_WINWINDOWMGR       /* Window Manager Functions    */
#include <os2.h>

BOOL  fSuccess;         /* Success indicator                   */
HWND  hwndClient;       /* client window                       */
HWND  hwndFrame;        /* frame window                        */
RECTL  rclBoundary;     /* Boundary rectangle                  */

/* convert client boundary coordinates to screen coordinates */
WinMapWindowPoints(hwndClient, HWND_DESKTOP, (PPOINTL)&rclBoundary,
                   2);

/* calculate equivalent frame boundary from boundary data */
fSuccess = WinCalcFrameRect(hwndFrame, &rclBoundary, FALSE);
```

WinCallMsgFilter

This function calls a message-filter hook.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinCallMsgFilter (HAB hab, PQMSG pqmsg, ULONG ulFilter)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pqmsg (PQMSG) – input
Message to be passed to the message-filter hook.

ulFilter (ULONG) – input
Filter.

Message-filter code passed to the message-filter hook. This can be an application-specific value or one of the following standard MSGF_* values:

MSGF_DIALOGBOX	Dialog-box mode loop.
MSGF_TRACK	Window-movement and size tracking. When this hook is used the TRACKINFO structure specified the *ptiTrackinfo* parameter of the WinTrackRect function is updated to give the current state before the hook is called. Only the *rclTrack* and the *fs* parameters are updated.
MSGF_DRAG	Direct manipulation mode loop.
MSGF_DDEPOSTMSG	DDE post message mode loop.

Returns

rc (BOOL) – returns
Message-filter hook return indicator.

TRUE	A message-filter hook returns TRUE
FALSE	All message-filter hooks return FALSE, or no message-filter hooks are defined.

Remarks

This function allows an application to pass a message to the message-filter hook procedure(s).

Related Functions

- WinReleaseHook
- WinSetHook

Example Code

This example calls a message filter hook and passes a WM_CLOSE message in message box mode.

```
#define INCL_WINHOOKS           /* Window Hook Functions        */
#include <os2.h>

BOOL  fHookRet;         /* filter hook return indicator         */
HAB   hab;              /* Anchor-block handle                  */
QMSG  pqmsg;            /* Message to be passed to the
                           message-filter hook                  */
ULONG  ulFilter;        /* Filter                               */
POINTL ptrPos = {5L,5L};/* pointer position                     */

/* initialize message structure */
pqmsg.hwnd  = HWND_DESKTOP;
pqmsg.msg   = WM_CLOSE;
pqmsg.mp1   = MPVOID;
pqmsg.mp2   = MPVOID;
pqmsg.time  = 0L;
pqmsg.ptl   = ptrPos;

/* call hook in message box mode */
ulFilter = MSGF_MESSAGEBOX;

fHookRet = WinCallMsgFilter(hab, &pqmsg, ulFilter);
```

WinCancelShutdown

This function cancels a request for an application to shut down.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinCancelShutdown  (HMQ hmq, BOOL fCancelAlways)
```

Parameters

hmq (HMQ) – input
Handle of message queue for current thread.

fCancelAlways (BOOL) – input
Cancellation control.

TRUE	No WM_QUIT message should be placed on this queue during system shutdown.
FALSE	The applications ignore any outstanding WM_QUIT messages already sent to it, but a message should be sent during other system shutdowns.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

On a system shutdown, each message queue is normally posted a WM_QUIT message. An application can handle this message in one of two ways:

- Destroy its message queue using WinDestroyMsgQueue (*hmq*) or,
- Call WinCancelShutdown (*hmq*, FALSE)

Either way the system can proceed to the next queue.

If the application determines that it never wants a message queue to receive a WM_QUIT message as a result of a system shutdown, it should call WinCancelShutdown (*hmq*, TRUE), typically right after creating the message queue.

Related Functions

- WinCreateMsgQueue
- WinInitialize
- WinTerminate

Related Messages

- WM_QUIT

Example Code

This example cancels a shutdown request (WM_QUIT message) and specifies that the message queue will not accept any new WM_QUIT messages.

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#include <os2.h>
HAB   hab,
BOOL  fSuccess;         /* Success indicator                    */
HMQ   hmq;              /* queue handle                         */

hmq = WinCreateMsgQueue(hab, 0);
fSuccess = WinCancelShutdown(hmq, TRUE);
```

WinChangeSwitchEntry

This function changes the information in a Window List entry.

Syntax

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinChangeSwitchEntry  (HSWITCH hswitchSwitch,
                             SWCNTRL swctlSwitchData)
```

Parameters

hswitchSwitch (HSWITCH) – input
Handle to the Window List entry to be changed.

swctlSwitchData (SWCNTRL) – input
Switch-control data.

Contains the information to change the Window List entry.

If the *idProcess* field of the SWCNTRL structure is 0, the current process ID is used.

If the *idSession* field of the SWCNTRL structure is 0, the current session ID is used.

Returns

ulRetCode (ULONG) – returns
Return code.

0	Successful completion
Other	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_SWITCH_HANDLE (0x1202)	An invalid Window List entry handle was specified.
PMERR_INVALID_WINDOW (0x1206)	The window specified with a Window List call is not a valid frame window.

Remarks

Leading and trailing blanks are removed from the title and, if necessary, it is truncated to 60 characters.

Example Code

This example changes the program name of a task-list entry to "Generic: NEW.APP" using the handle returned by WinAddSwitchEntry.

```
#define INCL_WINSWITCHLIST      /* Window Switch List Functions */
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HSWITCH hswitch;        /* task-list entry handle               */
SWCNTRL swctl;          /* switch-control data                  */
PID pid;                /* process id                           */
HAB hab;                /* anchor-block handle                  */
HWND hwndFrame;         /* frame handle                         */

hab = WinQueryAnchorBlock(hwndFrame);      /* gets anchor block */
WinQueryWindowProcess(hwndFrame, &pid, NULL);/* gets process id */

/* initialize switch structure */
swctl.hwnd = hwndFrame;                  /* window handle      */
swctl.hwndIcon = NULLHANDLE;             /* icon handle        */
swctl.hprog = NULLHANDLE;                /* program handle     */
swctl.idProcess = pid;                   /* process identifier */
swctl.idSession = 0;                     /* session identifier */
swctl.uchVisibility = SWL_VISIBLE;       /* visibility         */
swctl.fbJump = SWL_JUMPABLE;             /* jump indicator     */
swctl.szSwtitle[0]= 0;                   /* program name       */

hswitch = WinCreateSwitchEntry(hab, &swctl);

/* set application name */
strcpy(swctl.szSwtitle, "Generic: NEW.APP");

WinChangeSwitchEntry(hswitch, &swctl);
```

WinCheckButton

This macro sets the checked state of the specified button control. It returns the previous check state.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
USHORT WinCheckButton (HWND hwndDlg, USHORT usId, USHORT usChkstate)
```

Parameters

hwndDlg (HWND) – input
Dialog window handle.

usId (USHORT) – input
Button control identity.

usChkstate (USHORT) – input
Indicates the current checked state of the button.

Returns

usRetChkst (USHORT) – returns
Returns the previous checkstate.

Remarks

This macro expands to:

```
#define WinCheckButton(hwndDlg, usId, usChkstate)
  ((USHORT)WinSendDlgItemMsg(hwndDlg,
                             usId,
                             BM_SETCHECK,
                             MPFROMSHORT(usChkstate),
                             (MPARAM)NULL))
```

This call requires the existence of a message queue.

Related Functions

- WinSendDlgItemMsg

Related Messages

- BM_SETCHECK

Example Code

This example responds to a button click (BN_CLICKED, WM_CONTROL message) on a check box by setting the checked state of the button.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINBUTTONS         /* Window Button definitions    */
#include <os2.h>

USHORT  usCheckId;      /* check box id                         */
HWND  hwndDlg;          /* dialog window handle                 */
USHORT  usChkstate;     /* new checked state                    */
USHORT  usOldstate;     /* old checked state                    */
MPARAM  mp1;            /* Parameter 1 (rectl structure)        */
MPARAM  mp2;            /* Parameter 2 (frame boolean)          */

case WM_CONTROL:
     /* switch on control code */
     switch(SHORT2FROMMP(mp1))
        {
        case BN_CLICKED:
             usCheckId = SHORT1FROMMP(mp1);

             /* query current check state */
             usChkstate = WinQueryButtonCheckstate(hwndDlg,
                          usCheckId);

             /* set box check state */
             usOldstate = WinCheckButton(hwndDlg, usCheckId,
                                         usChkstate);
             break;
        }
```

WinCheckInput

This function is specific to version 2.1, or higher, of the OS/2 operating system.

This function is used in conjunction with the MsgInputHook hook to input simulated events into the Presentation Manager input processing system.

Syntax

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinCheckInput (HAB hab)
```

Parameters

hab (HAB) – input
The application anchor block.

Returns

rc (BOOL) – returns
Return code.

TRUE	The system checked for mouse and keyboard input.
FALSE	An error occurred.

Remarks

This function must be called whenever an application injects simulated mouse or keyboard events using MsgInputHook, otherwise there is no guarantee when those simulated events will occur. It forces the Presentation Manager input processing system to wake up and take a look at all possible sources of input events: the MsgInputHook hook, the JournalPlaybackHook hook, and the Presentation Manager input queue that contains normal mouse and keyboard events.

Related Functions

- MsgInputHook

WinCheckMenuItem

This macro sets the check state of the specified menu item to the flag.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinCheckMenuItem  (HWND hwndMenu, USHORT usId, BOOL fCheck)
```

Parameters

hwndMenu (HWND) – input
Menu window handle.

usId (USHORT) – input
Item identifier.

fCheck (BOOL) – input
Check flag.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This macro expands to:

```
#define WinCheckMenuItem(hwndMenu, usId, fCheck)
   ((BOOL)WinSendMsg(hwndMenu,
                     MM_SETITEMATTR,
                     MPFROM2SHORT(usId, TRUE),
                     MPFROM2SHORT(MIA_CHECKED, (BOOL)(fCheck) ? MIA_CHECKED : 0)))
```

This function requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- MM_SETITEMATTR

Example Code

This example responds to a select menu message (WM_MENUSELECT) by querying (via WinIsMenuItemChecked) the check attribute and then setting the check state of the menu item that was selected.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

USHORT  usItemId;       /* menu item id                         */
HWND    hwndMenu;       /* menu handle                          */
BOOL    usChkstate;     /* new checked state                    */
BOOL    fSuccess;       /* success indicator                    */
MPARAM  mp1;            /* Parameter 1 (menu item id)           */
MPARAM  mp2;            /* Parameter 2 (menu handle)            */

case WM_MENUSELECT:
     usItemId = SHORT1FROMMP(mp1);
     hwndMenu = HWNDFROMMP(mp2);

     /* query current check state */
     usChkstate = WinIsMenuItemChecked(hwndMenu, usItemId);

     /* set menu item check state */
     fSuccess = WinCheckMenuItem(hwndMenu, usItemId, ! usChkstate);
```

WinCloseClipbrd

This function closes the clipboard, allowing other applications to open it with the WinOpenClipbrd function.

Syntax

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinCloseClipbrd (HAB hab)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function causes the contents of the clipboard to be drawn in the clipboard viewer window (if any), by sending it a WM_DRAWCLIPBOARD message. This action occurs only if the clipboard data has changed.

The clipboard must be open before this function is invoked.

Related Functions

- WinEmptyClipbrd
- WinEnumClipbrdFmts
- WinOpenClipbrd
- WinQueryClipbrdData
- WinQueryClipbrdFmtInfo
- WinQueryClipbrdOwner
- WinQueryClipbrdViewer
- WinSetClipbrdData
- WinSetClipbrdOwner
- WinSetClipbrdViewer

Related Messages

- WM_DRAWCLIPBOARD

Example Code

This example closes the clipboard, previously opened by WinOpenClipbrd, to allow other applications to open it for use.

```
#define INCL_WINCLIPBOARD       /* Window Clipboard Functions   */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
HAB   hab;              /* anchor-block handle                  */

fSuccess = WinCloseClipbrd(hab);
```

WinCompareStrings

Compares two null-terminated strings defined using the same code page.

Syntax

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinCompareStrings  (HAB hab, ULONG idCodepage,
                          ULONG idCountryCode, PSZ pszString1,
                          PSZ pszString2, ULONG ulReserved)
```

Parameters

hab (HAB) – input
Anchor-block handle.

idCodepage (ULONG) – input
Code page identity of both strings.

If *idCodepage* is set to 0, the current code page of the caller is used.

idCountryCode (ULONG) – input
Country code.

If *idCountryCode* is set to 0, the country information for the default system country code is used.

pszString1 (PSZ) – input
String 1.

pszString2 (PSZ) – input
String 2.

ulReserved (ULONG) – input
Reserved value, should be 0.

Returns

ulResult (ULONG) – returns
Comparison result.

WCS_EQ	Strings are equal
WCS_LT	String 1 is less than string 2
WCS_GT	String 1 is greater than string 2
WCS_ERROR	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_STRING_PARM (0x100B)	The specified string parameter is invalid.

Remarks

The first code page of either of the two ASCII code pages specified in CONFIG.SYS is used as the system default. The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865
Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft** Windows**.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data. The following is the list of valid country codes:

Country	Code
Arabic	785
Australian	61
Belgian	32
Canadian-French	2
Danish	45
Finnish	358
French	33
German	49
Hebrew	972
Italian	39
Japanese	81
Korean	82
Latin-American	3
Netherlands	31
Norwegian	47
Portuguese	351
Simpl. Chinese	86
Spanish	34
Swedish	46
Swiss	41
Trad. Chinese	88
UK-English	44
US-English	1
Other country	0

Related Functions

- WinLoadString
- WinNextChar
- WinPrevChar
- WinSubstituteStrings
- WinUpper
- WinUpperChar

Example Code

This example compares two strings using the same code page: the first string is loaded from a resource DLL, while the second is created by the application.

```
#define INCL_WINCOUNTRY         /* Window Country Functions     */
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_DOSMODULEMGR       /* Module Manager Functions     */
#include <os2.h>

ULONG   ulResult;       /* comparison result                    */
HAB     hab;            /* anchor-block handle                  */
ULONG   idCodepage=437; /* Code page identity of both strings   */
ULONG   idCountryCode=1;/* Country code                         */
char    pszString1[10]; /* first string                         */
char    pszString2[10]; /* second string                        */
LONG    lLength;        /* length of string                     */
ULONG   idString = STRING_ID; /* String identifier              */
LONG    lBufferMax = 10;/* Size of buffer                       */
HMODULE hmodDLL;        /* Handle of the module which contains
                           the help table and help subtable
                           resources.                           */
CHAR    LoadError[100]; /* object name buffer for DosLoad       */
ULONG   rc;             /* return code                          */

/* obtain resource handle */
rc = DosLoadModule(LoadError, sizeof(LoadError), "RES.DLL",
                   &hmodDLL);

/* load string from resource */
if (rc == 0)
   lLength = WinLoadString(hab, hmodDLL, idString, lBufferMax,
                           pszString1);

/* compare strings */
if (lLength > 0)
   {
   /* set second string */
   strcpy(pszString2,"Compare");

   ulResult = WinCompareStrings(hab, idCodepage, idCountryCode,
                                pszString1, pszString2, 0);
   }
```

WinCopyAccelTable

This function is used to get the accelerator-table data corresponding to an accelerator-table handle, or to determine the size of the accelerator-table data.

Syntax

```
#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinCopyAccelTable  (HACCEL hAccel, PACCELTABLE pacctAccelTable,
                          ULONG ulCopyMax)
```

Parameters

hAccel (HACCEL) – input
Accelerator-table handle.

pacctAccelTable (PACCELTABLE) – in/out
Accelerator-table data area.

NULL Return the size, in bytes, of the complete accelerator table, and ignore the *ulCopyMax* parameter.

Other Copy up to *ulCopyMax* bytes of the accelerator table into this data area.

ulCopyMax (ULONG) – input
Maximum data area size.

Returns

ulCopied (ULONG) – returns
Amount copied or size required.

Other Amount of data copied into the data area, or the size of data area required for the complete accelerator table.

0 Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HACCEL (0x101A) An invalid accelerator-table handle was specified.

Related Functions

- WinCreateAccelTable
- WinDestroyAccelTable
- WinLoadAccelTable
- WinQueryAccelTable
- WinSetAccelTable
- WinTranslateAccel

Example Code

This example gets the accelerator-table data corresponding to an accelerator-table handle returned by WinCreateAccelTable or WinLoadAccelTable and assigns the accelerator table code page to a variable.

```
#define INCL_WINACCELERATORS    /* Window Accelerator Functions */
#include <os2.h>

ULONG   ulCopied;       /* bytes copied                          */
HACCEL  hAccel;         /* Accelerator-table handle              */
ACCELTABLE  pacctAccelTable;/* Accelerator-table data area       */
ULONG   ulCopyMax;      /* Maximum data area size                */
ULONG   ulAccelCP;      /* code page                             */

ulCopyMax = sizeof(pacctAccelTable);
if (hAccel)
   ulCopied = WinCopyAccelTable(hAccel, &pacctAccelTable,
                                ulCopyMax);
if (ulCopied)
   ulAccelCP = pacctAccelTable.codepage;
```

WinCopyObject

This function is specific to version 3, or higher, of the OS/2 operating system.

This function copies an object from its existing folder to a specified new destination.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
HOBJECT WinCopyObject  (HOBJECT hObjectofObject, HOBJECT hObjectofDest,
                        ULONG ulFlags)
```

Parameters

hObjectofObject (HOBJECT) – input
Handle of the object being copied.

hObjectofDest (HOBJECT) – input
Handle of the folder into which *hObjectofObject* is to be copied.

ulFlags (ULONG) – input
Flags.

COPY_FAILIFEXSTS

Returns

hwndDlg (HOBJECT) – returns
Handle of the newly created object.

A return value of NULLHANDLE indicates that either *hObjectofDest* is NULLHANDLE or an object with the same name as *hObjectofObject* exists in the destination folder.

Possible returns from WinGetLastError

WPERR_INVALID_FLAGS (0x1719) An invalid flag was specified.

Remarks

Using HOBJECTs for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

Related Functions

- WinCreateObject.
- WinDestroyObject
- WinMoveObject
- WinQueryObjectWindow
- WinSaveObject

WinCopyRect

This function copies a rectangle from *prclSrc* to *prclDst*.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinCopyRect  (HAB hab, PRECTL prclDst, PRECTL prclSrc)
```

Parameters

hab (HAB) – input
Anchor-block handle.

prclDst (PRECTL) – output
Destination rectangle.

Note: The value of each field in this structure must be in the range -32768 through 32767. The data type WRECT can also be used, if supported by the language.

prclSrc (PRECTL) – input
Source rectangle.

Note: The value of each field in this structure must be in the range -32768 through 32767. The data type WRECT can also be used, if supported by the language.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Related Functions

- WinEqualRect
- WinFillRect
- WinInflateRect
- WinIntersectRect
- WinIsRectEmpty
- WinOffsetRect
- WinPtInRect
- WinSetRect
- WinSetRectEmpty
- WinSubtractRect
- WinUnionRect

Example Code

This example copies a rectangle using WinCopyRect.

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions   */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
HAB   hab;              /* anchor-block handle                  */
RECTL prclRect2 = {0,0,200,200}; /* source rectangle            */
RECTL prclRect1;        /* destination rectangle                */

fSuccess = WinCopyRect(hab, &prclRect1, &prclRect2);
```

WinCpTranslateChar

This function translates a character from one code page to another.

Syntax

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
UCHAR WinCpTranslateChar (HAB hab, ULONG idCpSource, UCHAR ucSource,
                          ULONG idCpDest)
```

Parameters

hab (HAB) – input
Anchor-block handle.

idCpSource (ULONG) – input
Source-character code page.

ucSource (UCHAR) – input
Character to be translated.

idCpDest (ULONG) – input
Code page of the resultant character.

Returns

ucDest (UCHAR) – returns
If nonzero, the translated or substitution (0xFF) character.

Possible returns from WinGetLastError

PMERR_INVALID_STRING_PARM (0x100B)	The specified string parameter is invalid.
PMERR_INVALID_SRC_CODEPAGE (0x101D)	The source code page parameter is invalid.
PMERR_INVALID_DST_CODEPAGE (0x101E)	The destination code page parameter is invalid.

Remarks

Successful invocation of this function indicates that either (1) the character was directly mapped into the destination code page or, (2) the substitution character (0xFF) was returned.

The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865
Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft Windows.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data.

Related Functions

- WinCpTranslateString
- WinQueryCp
- WinQueryCpList
- WinSetCp

Example Code

This example translates a character from US code page 437 to multilingual code page 850.

```
#define INCL_WINCOUNTRY          /* Window Country Functions     */
#include <os2.h>

HAB     hab;             /* anchor-block handle                  */
UCHAR   ucDest;          /* translated character                 */
UCHAR   ucSource = 'A'; /* source character                      */
ULONG   idCpSource=437; /* Code page of source character (US)    */
ULONG   idCpDest=850;   /* Code page of dest. character (multi) */

ucDest = WinCpTranslateChar(hab, idCpSource, ucSource, idCpDest);
```

WinCpTranslateString

This function translates a string from one code page to another.

Syntax

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinCpTranslateString  (HAB hab, ULONG idCpSource, PSZ pszSource,
                            ULONG idCpDest, ULONG cbLenDest, PSZ pszDest)
```

Parameters

hab (HAB) – input
Anchor-block handle.

idCpSource (ULONG) – input
Source-string code page.

pszSource (PSZ) – input
String to be translated.

This is a null-terminated string.

idCpDest (ULONG) – input
Code page of the resultant string.

cbLenDest (ULONG) – input
Maximum length of output string.

An error is raised if this is not large enough to contain the translated string.

pszDest (PSZ) – output
The translated string.

This is a null-terminated string.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_STRING_PARM (0x100B)	The specified string parameter is invalid.

PMERR_INVALID_SRC_CODEPAGE (0x101D)	The source code page parameter is invalid.
PMERR_INVALID_DST_CODEPAGE (0x101E)	The destination code page parameter is invalid.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Remarks

Successful invocation of this function indicates that either (1) the character was directly mapped into the destination code page or, (2) the substitution character (0xFF) was returned.

The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865
Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft Windows.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data.

Related Functions

- WinCpTranslateChar
- WinQueryCp
- WinQueryCpList
- WinSetCp

Example Code

This example translates a string from US code page 437 to multilingual code page 850.

```
#define INCL_WINCOUNTRY         /* Window Country Functions     */
#include <os2.h>

HAB     hab;            /* anchor-block handle                  */
ULONG   idCpSource=437; /* Code page of source character (US)   */
ULONG   idCpDest=850;   /* Code page of dest. character (multi) */
char    pszSource[10];  /* source string                        */
char    pszDest[10];    /* destination string                   */
ULONG   cbLenDest = 9L; /* max length of destination string     */
BOOL    fSuccess;
strcpy(pszSource,"TRANSLATE");
fSuccess = WinCpTranslateString(hab, idCpSource, pszSource,
                                idCpDest, cbLenDest, pszDest);
```

WinCreateAccelTable

This function creates an accelerator table from the accelerator definitions in memory.

Syntax

```
#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HACCEL WinCreateAccelTable  (HAB hab, PACCELTABLE pacctAccelTable)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pacctAccelTable (PACCELTABLE) – input
Accelerator table.

Returns

haccelhAccel (HACCEL) – returns
Accelerator-table handle.

Remarks

This function returns a different *haccelhAccel* value when called twice in succession with the same parameter values.

Related Functions

- WinCopyAccelTable
- WinDestroyAccelTable
- WinLoadAccelTable
- WinQueryAccelTable
- WinSetAccelTable
- WinTranslateAccel

Example Code

This example creates an accelerator-table handle for an in memory accelerator table consisting of 3 accelerator keys, using US codepage 437.

```
#define INCL_WINACCELERATORS    /* Window Accelerator Functions */
#define INCL_WININPUT           /* Key constants                */
#define INCL_WINFRAMEMGR        /* Frame control constants      */
#include <os2.h>

ULONG        ulAccelLen      = 0;          /* Accelerator-table length      */
HACCEL       hAccel          = NULLHANDLE; /* Accelerator-table handle      */
PACCELTABLE  pacctAccelTable = NULL;       /* Pointer to Accelerator-tab le */
HAB          hab             = NULLHANDLE; /* Anchor block handle           */

ACCEL   acctable[] = {
         AF_SYSCOMMAND | AF_ALT | AF_VIRTUALKEY,VK_F7,SC_MOVE,
         AF_SYSCOMMAND | AF_ALT | AF_VIRTUALKEY,VK_F8,SC_SIZE,
         AF_SYSCOMMAND | AF_ALT | AF_VIRTUALKEY,VK_F12,SC_CLOSE};

        /* Get memory for the Accelerator-table and initialize it. */

  ulAccelLen = sizeof( acctable ) + sizeof( ACCELTABLE );
  pacctAccelTable = (PACCELTABLE) malloc ( ulAccelLen );

  pacctAccelTable->cAccel = 3;                /* Number of ACCEL entries */
  pacctAccelTable->codepage = 437;            /* Code page */
  pacctAccelTable->aaccel[0] = acctable[0];   /* ACCEL entries... */
  pacctAccelTable->aaccel[1] = acctable[1];
  pacctAccelTable->aaccel[2] = acctable[2];

  hAccel = WinCreateAccelTable( hab, pacctAccelTable );
```

WinCreateAtomTable

This function creates a private empty atom table.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HATOMTBL WinCreateAtomTable  (ULONG ulInitial, ULONG ulBuckets)
```

Parameters

ulInitial (ULONG) – input
Initial bytes.

Initial number of bytes to be reserved for the atom table. This is a lower bound on the amount of memory reserved. The amount of memory actually used by an atom table depends upon the actual number of atoms stored in the table. If zero, the size of the atom table is the minimum size needed to store the atom hash table.

ulBuckets (ULONG) – input
Size of the hash table.

Used to access atoms. If zero, the default value of 37 is used. The best results are achieved if a prime number is used.

Returns

hatomtblAtomTbl (HATOMTBL) – returns
Atom-table handle.

NULLHANDLE	Call failed.
Other	Atom-table handle. This must be passed as a parameter in subsequent atom manager calls.

Remarks

The minimum size of atom table allocated is 16+(2**ulBuckets*).

The atom table is owned by the process from which this function is issued. It cannot be accessed directly from any other process. If it is still in existence when the process terminates, it will automatically be deleted by the system.

There is a system atom table which is created at boot time, which cannot be destroyed, and which can be accessed by any process in the system. The handle of the system atom table is queried with the WinQuerySystemAtomTable function.

Related Functions

- WinAddAtom
- WinDeleteAtom
- WinDestroyAtomTable
- WinFindAtom
- WinQueryAtomLength
- WinQueryAtomName
- WinQueryAtomUsage
- WinQuerySystemAtomTable

Example Code

This example creates an Atom Table of default size, adds the atom "newatom" to the new table, and then destroys the table.

```
#define INCL_WINATOM            /* Window Atom Functions        */
#include <os2.h>

ATOM  atom;             /* new atom value                       */
HATOMTBL  hatomtblAtomTbl; /* atom-table handle                 */
HATOMTBL  hatomtblDestroy; /* result of destroy table           */
char   pszAtomName[10]; /* atom name                            */
ULONG   ulInitial = 0;  /* initial atom table size (use default)*/
ULONG   ulBuckets = 0;  /* size of hash table (use default)     */

/* create atom table of default size */
hatomtblAtomTbl = WinCreateAtomTable(ulInitial, ulBuckets);

/* define name for new atom and add to table */
strcpy(pszAtomName,"newatom");
atom = WinAddAtom(hatomtblAtomTbl, pszAtomName);

hatomtblDestroy = WinDestroyAtomTable(hatomtblAtomTbl);
```

WinCreateCursor

This function creates or changes a cursor for a specified window.

Syntax

```
#define INCL_WINCURSORS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinCreateCursor  (HWND hwnd, LONG lx, LONG ly, LONG lcx, LONG lcy,
                       ULONG ulrgf, PRECTL prclClip)
```

Parameters

hwnd (HWND) – input
Handle of window in which cursor is displayed.

This must be the handle of a window for which the application can receive input.

lx (LONG) – input
x-position of cursor.

ly (LONG) – input
y-position of cursor.

lcx (LONG) – input
x-size of cursor.

If 0, the system nominal border width (SV_CXBORDER) is used.

lcy (LONG) – input
y-size of cursor.

If 0, the system nominal border height (SV_CYBORDER) is used.

ulrgf (ULONG) – input
Controls the appearance of the cursor.

CURSOR_SOLID	The cursor is solid.
CURSOR_HALFTONE	The cursor is halftone.
CURSOR_FRAME	The cursor is a rectangular frame.
CURSOR_FLASH	The cursor flashes.
CURSOR_SETPOS	Set a new cursor position. *lcx* and *lcy* are ignored. Used when a cursor has already been created. In this case, all other appearance flags are ignored.

prclClip (PRECTL) – input
Cursor rectangle.

A rectangle within which the cursor is visible. If the cursor goes outside this rectangle, it is clipped away and is invisible.

The rectangle is specified in window coordinates.

If *prclClip* is NULL, the drawing of the cursor is clipped to the window rectangle of *hwnd*.

Note: The cursor is always clipped to the window rectangle, even if part of *prclClip* is outside it.

The value of each field in this structure must be in the range -32768 through 32767. The data type WRECT can also be used, if supported by the language.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

The cursor is used to indicate the position of text input. It is initially hidden and must be made visible using the WinShowCursor function.

This function destroys any existing cursor, as it is confusing to the user if two cursors are visible at any one time. An application creates and displays a cursor when it has the input focus, or is the active window. Creating a cursor at any other time can stop the cursor from flashing in another window. Similarly, when the application loses the input focus or becomes inactive, it destroys its cursor using the WinDestroyCursor function.

The cursor width is generally specified as 0 (nominal border width is used). This is preferable to a value of 1, for example, as such a fine width is almost invisible on a high-resolution device.

When *ulrgf* is set to CURSOR_FLASH, TID_CURSOR timer is created.

Related Functions

- WinDestroyCursor
- WinQueryCursorInfo
- WinShowCursor

Example Code

This example creates a cursor of default height and width at (0,200) which will be visible within the entirety of the input window.

```
#define INCL_WINCURSORS         /* Window Cursor Functions      */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
HWND    hwnd;           /* cursor display window                */
LONG   lx = 0;          /* cursor x position                    */
LONG   ly = 200;        /* cursor y position                    */
ULONG   ulrgf;          /* cursor appearance                    */

fSuccess = WinCreateCursor(hwnd, lx, ly, 0, 0, ulrgf, NULL);
```

WinCreateDlg

This function creates a dialog window.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinCreateDlg  (HWND hwndParent, HWND hwndOwner,
                    PFNWP pfnDlgProc, PDLGTEMPLATE pdlgt,
                    PVOID pCreateParams)
```

Parameters

hwndParent (HWND) – input
Parent-window handle of the created dialog window.

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

hwndOwner (HWND) – input
Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the WinLoadDlg function.

pfnDlgProc (PFNWP) – input
Dialog procedure for the created dialog window.

pdlgt (PDLGTEMPLATE) – input
Dialog template.

pCreateParams (PVOID) – input
Pointer to application-defined data area.

This is passed to the dialog procedure in the WM_INITDLG message.

This parameter MUST be a pointer rather than a long.

Returns

hwndDlg (HWND) – returns
Dialog-window handle.

NULLHANDLE	Dialog window not created
Other	Dialog-window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_INTEGER_ATOM (0x1016)	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME (0x1015)	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND (0x1017)	The specified atom name is not in the atom table.

Remarks

This function is identical to the WinLoadDlg call except that it creates a dialog window from *pdlgt* in memory, rather than a dialog template in a resource file.

This function should not be used while the pointing device capture is set (see WinSetCapture).

Related Functions

- WinDefDlgProc
- WinDismissDlg
- WinDlgBox
- WinGetDlgMsg
- WinLoadDlg
- WinProcessDlg

Related Messages

- WM_INITDLG

Example Code

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.

```
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#define INCL_DOSRESOURCES       /* OS/2 Resource functions      */
#include <os2.h>

PFNWP    MyDlgProc;
#define  ID_DIALOG  1

PDLGTEMPLATE pdlgt;     /* dialog template                      */

DosGetResource(0L, RT_DIALOG, ID_DIALOG, (PVOID)pdlgt);

/* make any changes to dialog template here */

WinCreateDlg(HWND_DESKTOP,
    NULLHANDLE,  /* owner window               */
    MyDlgProc,   /* address of dialog procedure */
    pdlgt,       /* address of dialog structure */
    NULL);       /* application-specific data   */
```

WinCreateFrameControls

This function creates the standard frame controls for a specified window.

Syntax

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinCreateFrameControls (HWND hwndFrame, PFRAMECDATA pfcdata,
                             PSZ pszTitle)
```

Parameters

hwndFrame (HWND) – input
Frame-window handle.

Becomes the parent and owner window of all the frame controls that are created:

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

pfcdata (PFRAMECDATA) – input
Frame-control data.

This includes a combination of frame creation flags (FCF_*), that specifies which frame controls are to be created. For further information, see "Frame Creation Flags" in the *Presentation Manager Programming Reference Volume II*.

pszTitle (PSZ) – input
Title string.

This parameter contains a string that is displayed in the WC_TITLEBAR control only when FCF_TITLEBAR is specified in *pfcdata*.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

This function is typically used when the standard frame controls are needed for use with a nonstandard window (such as a non WC_FRAME class).

All of the controls are created with the standard FID_* window identifiers; see "Frame Control Window Processing," in the *Presentation Manager Programming Reference Volume II*.

The controls are created but not formatted. Formatting must be done by setting the required positions. All controls are created with position and size set to zero and WS_VISIBLE is not set.

Related Functions

- WinCalcFrameRect
- WinCreateStdWindow
- WinCreateWindow
- WinDefWindowProc
- WinDestroyWindow
- WinQueryClassInfo
- WinQueryClassName
- WinRegisterClass
- WinSubclassWindow

Example Code

This example creates frame controls (title bar, system menu, size border, and min/max buttons) for a button window created by WinCreateWindow. The new controls are owned by and children of the button window.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINLISTBOXES       /* List Box definitions         */
#define INCL_WINFRAMEMGR        /* Frame Manager Functions      */
#include <os2.h>

HWND    hwnd;           /* cursor display window                */
ULONG flStyle;          /* window style                         */
USHORT ButtonId;        /* window id  (app supplied)            */
BOOL  fSuccess;         /* success indicator                    */
FRAMECDATA  pFcdata;    /* Frame-control data                   */
USHORT  usFrameId;      /* frame resource id (app supplied)     */


flStyle = WS_VISIBLE;                   /* create window visible  */

/* create button window (no frame controls) */
hwnd = WinCreateWindow(HWND_DESKTOP,  /* parent window          */
                       WC_BUTTON,    /* class name             */
                       "new button",  /* window text           */
                       flStyle,       /* window style          */
                       0, 0,          /* position (x,y)        */
                       200, 100,      /* size (width,height)   */
                       0L,            /* owner window          */
                       HWND_TOP,      /* sibling window        */
                       ButtonId,      /* window id             */
                       NULL,          /* control data          */
                       NULL);         /* presentation parms    */

/*****************************
 * initialize frame structure *
 *****************************/
pFcdata.cb = sizeof(FRAMECDATA);  /* Length  */
/* Frame-creation flags  */
pFcdata.flCreateFlags = FCF_TITLEBAR |  FCF_SYSMENU |
                        FCF_SIZEBORDER | FCF_MINMAX;
pFcdata.hmodResources = 0L;       /* resource in EXE */
pFcdata.idResources = usFrameId;  /* resource id */

/* create frame controls; display 'button frame' on title bar */
fSuccess = WinCreateFrameControls(hwnd, &pFcdata, "button frame");
```

WinCreateHelpInstance

This function creates an instance of the Help Manager with which to request Help Manager functions.

Syntax

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinCreateHelpInstance  (HAB hab, PHELPINIT phinitHMInitStructure)
```

Parameters

hab (HAB) – input
Anchor-block handle.

The handle of the application anchor block returned from the WinInitialize function.

phinitHMInitStructure (PHELPINIT) – in/out
Help Manager initialization structure.

Returns

hwndhelp (HWND) – returns
Help Manager handle.

NULLHANDLE	Error occurred
Other	Help Manager handle.

Remarks

If an error occurs, it is in the *ulReturnCode* parameter of the HELPINIT structure.

Related Functions

- WinAssociateHelpInstance
- WinCreateHelpTable
- WinDestroyHelpInstance
- WinLoadHelpTable
- WinQueryHelpInstance

Example Code

This example shows a typical main function for an application which uses help. Following creation of the main application window the Help Manager is initialized and associated with the window. The help table is defined in the application's resources. When the window is destroyed, terminating the application, the help instance is also destroyed.

```
#define INCL=_WIN
#include <os2.h>

#define IDHT_APPLICATION        100     /* id of HELP TABLE in resource file
*/

main( int argc, char *argv[], char *envp[] )
{
   HAB  hab = WinInitialize( 0 );
   HMQ  hmq = WinCreateMsgQueue( hab, 0 );
   HWND hwnd;
   HWND hwndClient;
   HWND hwndHelp;
   QMSG qmsg;
   ULONG flStyle;
   HELPINIT helpinit;

   /* Setup the help initialization structure */
   helpinit.cb = sizeof( HELPINIT );
   helpinit.ulReturnCode =  0L;
   helpinit.pszTutorialName =  (PSZ)NULL;
   /* Help table in application resource */
   helpinit.phtHelpTable = (PHELPTABLE)MAKEULONG( IDHT_APPLICATION, 0xffff );
   helpinit.hmodHelpTableModule = NULLHANDLE;
   /* Default action bar and accelerators */
   helpinit.hmodAccelActionBarModule = NULLHANDLE;
   helpinit.idAccelTable = 0;
   helpinit.idActionBar = 0;
   helpinit.pszHelpWindowTitle = "APPNAME HELP";
   helpinit.fShowPanelId = CMIC_SHOW_PANEL_ID;
   helpinit.pszHelpLibraryName = "APPNAME.HLP";

   /* Register the class */
   if( WinRegisterClass( ... ) )
   {
      /* create the main window */
      flStyle = FCF_STANDARD;
      hwnd = WinCreateStdWindow( ... );

      if( hwnd )
      {
         /* Create and associate the help instance */
         hwndHelp = WinCreateHelpInstance( hab, &helpinit );
```

```
         if( hwndHelp && WinAssociateHelpInstance( hwndHelp, hwnd ) )
         {
            /* Process messages */
            while( WinGetMsg( hab, &qmsg, NULLHANDLE, 0, 0 ) )
            {
               WinDispatchMsg( hab, &qmsg );
            } /* endwhile */
         }

         /* Remove help instance - note: add                    */
         /*     WinAssociateHelpInstance( NULLHANDLE, hwnd );      */
         /* to WM_DESTROY processing to remove the association. */
         WinDestroyHelpInstance( hwndHelp );
      }
   }

   /* finish the cleanup and exit */
   WinDestroyMsgQueue( hmq );
   WinTerminate( hab );
}
```

WinCreateHelpTable

This function is used to identify or change the help table.

Syntax

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinCreateHelpTable  (HWND hwndHelpInstance,
                          PHELPTABLE phtHelpTable)
```

Parameters

hwndHelpInstance (HWND) – input
Handle of an instance of the Help Manager.

This is the handle returned by the WinCreateHelpInstance call.

phtHelpTable (PHELPTABLE) – input
Help table allocated by the application.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function corresponds to the HM_CREATE_HELP_TABLE message that identifies a help table that is in memory.

Related Functions

- WinAssociateHelpInstance
- WinCreateHelpInstance
- WinDestroyHelpInstance
- WinLoadHelpTable
- WinQueryHelpInstance

Related Messages

- HM_CREATE_HELP_TABLE

Example Code

This example creates a help table in memory and passes the table to the Help Manager via WinCreateHelpTable. The help instance must have been created by WinCreateHelpInstance.

```
#define INCL_WINHELP
#include <os2.h>

/* DEFINEs for window id's, menu items, controls, panels, etc. should */
/* be inserted here or in additional include files.                   */

/* Subtable for the main window's help */
HELPSUBTABLE phtMainTable[] = { 2,              /* Length of each entry */
                                /* Fill in one line for each menu item   */
                                IDM_FILE,       PANELID_FILEMENU,
                                IDM_FILENEW,    PANELID_FILENEW,
                                IDM_FILEOPEN,   PANELID_FILEOPEN,
                                IDM_FILESAVE,   PANELID_FILESAVE,
                                IDM_FILESAVEAS, PANELID_FILESAVEAS,
                                IDM_FILEEXIT,   PANELID_FILEEXIT };

/* Subtable for the dialog window's help */
HELPSUBTABLE phtDlgTable[] = { 2,             /* Length of each entry */
                               /* Fill in one line for each control */
                               IDC_EDITFLD,    PANELID_DLGEDITFLD,
                               IDC_OK,         PANELID_DLGOK,
                               IDC_CANCEL,     PANELID_DLGCANCEL,
                               IDC_HELP,       PANELID_HELP };

/* Help table for the applications context sensitive help */
HELPTABLE phtHelpTable[]= { WINDOWID_MAIN, phtMainTable,
PANELID_MAINEXT,
                            WINDOWID_DLG,  phtDlgTable,  PANELID_DLGEXT,
                            0,             NULL,           0 };

BOOL CreateHelpTable( HWND hWnd )
{
    BOOL bSuccess = FALSE;
    HWND hwndHelp;

    /* Get the associated help instance */
    hwndHelp = WinQueryHelpInstance( hWnd );

    if( hwndHelp )
    {
        /* Pass address of help table to the Help Manager */
        bSuccess = WinCreateHelpTable( hwndHelp, phtHelpTable );
    }

    /* return success indicator */
    return bSuccess;
}
```

WinCreateShadow

This function creates a shadow of an object.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
HOBJECT WinCreateShadow  (HOBJECT hObjectofObject,
                          HOBJECT hObjectofDest, ULONG ulReserved)
```

Parameters

hObjectofObject (HOBJECT) – input
Handle of the object being copied.

hObjectofDest (HOBJECT) – input
Handle of the folder into which *hObjectofObject* is to be copied.

ulReserved (ULONG) – input
Reserved value.

Returns

hwndDlg (HOBJECT) – returns
Handle of the newly created object.

A return value of NULLHANDLE indicates that either *hObjectofDest* is NULLHANDLE or an object with the same name as *hObjectofObject* exists in the destination folder.

Remarks

Using HOBJECT for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

Related Functions

- WinCreateObject.
- WinDestroyObject
- WinMoveObject
- WinQueryObjectWindow
- WinSaveObject

WinCreateMenu

This function creates a menu window from the menu template.

Syntax

```
#define INCL_WINMENUS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinCreateMenu  (HWND hwndParent, PVOID lpmt)
```

Parameters

hwndParent (HWND) – input
Owner- and parent-window handle of the created menu window.

If this is HWND_OBJECT or a window handle returned by the WinQueryObjectWindow call, the menu window is created as an object window.

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

lpmt (PVOID) – input
Menu template in binary format.

Returns

hwndMenu (HWND) – returns
Menu-window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The menu window is created with an identity of FID_MENU.

When a WC_MENU window is created with the WinCreateWindow call, *pCtlData* is assumed to be a menu template, which is used to create the menu. If *pCtlData* is NULL, an empty menu is created.

Related Functions

- WinLoadMenu
- WinPopupMenu

Example Code

This code will load a menu template from a dll and then use it to add a menu to the frame window hwndFrame which has been previously created without a menu.

```
HMOD     hmod;
HWND     hwndFrame, hwndMenu;
USHORT   idMenu = 999;
BYTE     lpmt;

DosLoadModule(NULL, 0, "MYDLL.DLL", &hmod);

                        /* Load menu template */
DosGetResource2(hmod, (USHORT)RT_MENU, idMenu, &lpmt);

hwndMenu = WinCreateMenu(hwndFrame, lpmt); /* Create a menu */

DosFreeResource(lpmt); /* free menu template resource */
```

WinCreateMsgQueue

This function creates a message queue.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
HMQ WinCreateMsgQueue  (HAB hab, LONG lQueuesize)
```

Parameters

hab (HAB) – input
Anchor-block handle.

lQueuesize (LONG) – input
Maximum queue size.

This is the maximum number of messages that can be queued.

0	Use the system default queue size; that is 10 messages.
Other	Maximum queue size.

Returns

hmq (HMQ) – returns
Message-queue handle.

NULLHANDLE	Queue cannot be created.
Other	Message-queue handle.

Possible returns from WinGetLastError

PMERR_HEAP_MAX_SIZE_REACHED (0x1012)	The heap has reached its maximum size (64KB), and cannot be increased.
PMERR_HEAP_OUT_OF_MEMORY (0x1011)	An attempt to increase the size of the heap failed.
PMERR_QUEUE_TOO_LARGE (0x1018)	An attempt to create a message queue has failed because the value specified for the size of the message queue is too large.
PMERR_RESOURCE_NOT_FOUND (0x100A)	The specified resource identity could not be found.

PMERR_NOT_IN_A_PM_SESSION (0x1051)	An attempt was made to access function that is only available from PM programs from a non-PM session.
PMERR_MSG_QUEUE_ALREADY_EXISTS (0x1052)	An attempt to create a message queue for a thread failed because a message queue already exists for the calling thread.

Remarks

Most PM calls require a message queue. WinCreateMsgQueue must be issued after the WinInitialize function, but before any other PM calls are invoked. It must be issued only once per thread.

Related Functions

- WinBroadcastMsg
- WinCancelShutdown
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInitialize
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinTerminate
- WinWaitMsg

Example Code

This example creates a message queue of default size.

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HAB     hab;            /* anchor-block handle                  */
HMQ     hmq;            /* message queue handle                 */
QMSG    qmsg;           /* message                              */

hab = WinInitialize(0);          /* initialize PM */

hmq = WinCreateMsgQueue(hab, 0); /* create default size queue */

/*
.
.  initialize windows
.
*/

/* get and dispatch messages from queue */
while (WinGetMsg(hab, &qmsg, 0, 0, 0))
   WinDispatchMsg(hab, &qmsg);
```

WinCreateObject

This function creates an instance of object class *pszClassName*, with title *pszTitle*, and places the icon and title in the location referred to by *pszLocation*.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
HOBJECT WinCreateObject  (PSZ pszClassName, PSZ pszTitle,
                          PSZ pszSetupString, PSZ pszLocation,
                          ULONG ulFlags)
```

Parameters

pszClassName (PSZ) – input
Pointer to class name.

A pointer to a zero-terminated string which contains the name of the class of which this object is a member.

pszTitle (PSZ) – input
Pointer to initial title of object.

A pointer to a zero-terminated string which contains the initial title of the object as it is to appear when displayed on the user interface underneath an icon or on the title bar of an open object.

pszSetupString (PSZ) – input
Pointer to setup string.

pszLocation (PSZ) – input
Folder location.

This value can be in any of the following formats:

- Predefined object ids of system folders.

"<WP_NOWHERE>"	The hidden folder.
"<LOCATION_DESKTOP>"	The currently active desktop.
"<WP_OS2SYS>"	The System folder.
"<WP_TEMPS>"	The Templates folder.
"<WP_CONFIG>"	The System Setup folder.
"<WP_START>"	The Startup folder.
"<WP_INFO>"	The Information folder.
"<WP_DRIVES>"	The Drives folder.

- Real name specified as a fully qualified path name.

ulFlags (ULONG) – input
Creation flags.

This parameter can have one of the following values:

CO_FAILIFEXISTS	No object will be created if an object with the given object ID already exists. This is the default.
CO_REPLACEIFEXISTS	If an object with the given ID already exists, the existing object should be replaced.
CO_UPDATEIFEXISTS	If an object with the given ID already exists, the existing object should be updated with the new information.

Returns

rc (HOBJECT) – returns
Handle to the created object.

NULLHANDLE	Error occurred.
Other	A handle to the object created. This handle is persistent and can be used for the WinSetObjectData and WinDestroyObject function calls.

Remarks

The *pszSetupString* contains a series of "keyname=value" pairs, that change the behavior of the object. "keynames" are separated by semicolons, and "values" are separated by commas.

```
"key=value;key2=value1,value2;"
```

If you want a literal comma or a literal semicolon inside one of your fields you must type the following:

^,	A literal comma.
^;	A literal semicolon.

Each object class documents the keynames and the parameters it expects to see immediately following. See the following keyname-value pairs tables:

- wpSetup
- wpColorPalette
- wpFolder
- wpFontPalette
- wpPalette
- wpProgram
- wpProgramFile
- wpShadow

Note that ALL parameters have safe defaults, so it is never necessary to pass unnecessary parameters to an object.

For more information about object classes, see the *Workplace Shell Programming Reference*.

Using HOBJECT for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

Related Functions

- WinDeregisterObjectClass
- WinDestroyObject
- WinRegisterObjectClass
- WinReplaceObjectClass
- WinSetObjectData

WinCreatePointer

This function creates a pointer from a bit map.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HPOINTER WinCreatePointer (HWND hwndDesktop, HBITMAP hbmPointer,
                           BOOL fPointer, LONG xHotspot, LONG yHotspot)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle or HWND_DESKTOP.

hbmPointer (HBITMAP) – input
Bit-map handle from which the pointer image is created.

The bit map must be logically divided into two sections vertically, each half representing one of the two images used as the successive drawing masks for the pointer.

For an icon, there are two bit map images. The first half is used for the AND mask and the second half is used for the XOR mask. For details of bit map formats, see "Standard Bit-Map Formats" in the *Presentation Manager Programming Reference Volume II*.

fPointer (BOOL) – input
Pointer-size indicator.

TRUE	The bit map should be stretched (if necessary) to the system pointer dimensions.
FALSE	The bit map should be stretched (if necessary) to the system icon dimensions.

xHotspot (LONG) – input
x-offset of hot spot within pointer from its lower left corner (in pels).

yHotspot (LONG) – input
y-offset of hot spot within pointer from its lower left corner (in pels).

Returns

hptr (HPOINTER) – returns
Pointer handle.

NULLHANDLE	Error
Other	Handle of the newly created pointer.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_HBITMAP_BUSY (0x2032)	An internal bit map busy error was detected. The bit map was locked by one thread during an attempt to access it from another thread.

Remarks

A pointer can be created either as a true pointer (at pointer size), or as an icon pointer (at icon size). The latter is useful when using icons as direct-manipulation objects that the user can "pick up" and move about the screen as a means of performing some operation.

See also WinCreatePointerIndirect.

This function makes copies of the supplied bit maps.

Related Functions

- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinQuerySysPointerData
- WinSetPointer
- WinSetPointerPos
- WinSetSysPointerData
- WinShowPointer

Example Code

This example creates a pointer from a bit map during the creation of the window (WM_CREATE). The bit map (id IDP_BITMAP in the EXE file) is loaded via GpiLoadBitmap.

```
#define INCL_WINPOINTERS        /* Window Pointer Functions     */
#define INCL_GPIBITMAPS         /* Graphics bit map Functions    */
#include <os2.h>

HPS   hps;              /* presentation-space handle            */
HWND  hwnd;             /* window handle                        */
HPOINTER  hptr;         /* bit-map pointer handle               */
HBITMAP  hbm;           /* bit-map handle                       */

case WM_CREATE:
     hps = WinBeginPaint(hwnd, NULLHANDLE, NULL);
     hbm = GpiLoadBitmap(hps, 0L, IDP_BITMAP, 64L, 64L);
     WinEndPaint(hps);

     hptr = WinCreatePointer(HWND_DESKTOP, hbm,
                             TRUE, /* use true (system) pointer */
                             0, 0); /* hot spot offset (0,0)    */
```

WinCreatePointerIndirect

This function creates a colored pointer or icon from a bit map.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HPOINTER WinCreatePointerIndirect (HWND hwndDesktop, PPOINTERINFO pptri)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle or HWND_DESKTOP.

pptri (PPOINTERINFO) – input
Pointer information structure.

The fields in this structure must be set before the call is made:

- *fPointer* is set to TRUE if a pointer is being created, or to FALSE if an icon is being created.
- *xHotSpot yHotSpot* are set to the relative position in the icon or pointer that is associated with the mouse position.
- *hbmPointer* is a bit map that specifies the AND and XOR masks, as used for black and white pointers and icons.
- *hbmColor* is a color bit map that describes the color content of the pointer or icon.

It is an error if *hbmPointer* is NULLHANDLE. Also, the width of *hbmPointer* must be the same as that of *hbmColor hbmColor*, and the height of *hbmPointer* must be double that of *hbmColor* (to allow for both the AND and the XOR mask).

Returns

hptr (HPOINTER) – returns
Pointer handle.

NULLHANDLE	Error
Other	Handle of the newly created pointer or icon.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_HBITMAP_BUSY (0x2032)	An internal bit map busy error was detected. The bit map was locked by one thread during an attempt to access it from another thread.

Remarks

A pointer can be created either as a true pointer (at pointer size), or as an icon pointer (at icon size). The latter is useful when using icons as direct-manipulation objects that the user can "pick up" and move about the screen as a means of performing some operation (see also WinCreatePointer).

This function makes copies of the supplied bit maps.

Related Functions

- WinCreatePointer
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinQuerySysPointerData
- WinSetPointer
- WinSetPointerPos
- WinSetSysPointerData
- WinShowPointer

Example Code

This example creates a colored pointer from a bit map during the creation of the window (WM_CREATE). The pointer bit map (id IDP_BITMAPPTR in the EXE) and color bit map (id IDP_BITMAPCLR in the EXE file) are loaded via GpiLoadBitmap.

```
#define INCL_WINPOINTERS        /* Window Pointer Functions     */
#define INCL_GPIBITMAPS         /* Graphics bit map Functions    */
#include <os2.h>

HPS   hps;              /* presentation-space handle            */
HWND  hwnd;             /* window handle                        */
HPOINTER  hptr;         /* bit-map pointer handle               */
HBITMAP  hbmPointer;    /* bit-map handle (AND/XOR)             */
HBITMAP  hbmColor;      /* bit-map handle (color)               */
POINTERINFO  pptriPointerInfo; /* pointer info structure        */

case WM_CREATE:
     hps = WinBeginPaint(hwnd, NULLHANDLE, NULL);
     /* load pointer bit map */
     hbmPointer = GpiLoadBitmap(hps, NULLHANDLE, IDP_BITMAPPTR, 64L, 128L);
     /* load color bit map */
     hbmColor = GpiLoadBitmap(hps, NULLHANDLE, IDP_BITMAPCLR, 64L, 64L);
     WinEndPaint(hps);

     /* initialize POINTERINFO structure */
     pptriPointerInfo.fPointer = TRUE;  /* creating pointer */
     pptriPointerInfo.xHotspot = 0; /* x coordinate of hotspot */
     pptriPointerInfo.yHotspot = 0; /* y coordinate of hotspot */
     pptriPointerInfo.hbmPointer = hbmPointer;
     pptriPointerInfo.hbmColor = hbmColor;

     hptr = WinCreatePointerIndirect(HWND_DESKTOP,
                                     &pptriPointerInfo);
```

WinCreateStdWindow

This function creates a standard window.

Syntax

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
HWND WinCreateStdWindow  (HWND hwndParent, ULONG flStyle,
                          PULONG pflCreateFlags, PSZ pszClassClient,
                          PSZ pszTitle, ULONG flStyleClient,
                          HMODULE Resource, ULONG ulId,
                          PHWND phwndClient)
```

Parameters

hwndParent (HWND) – input
Parent-window handle.

If this parameter is a window handle returned from the WinQueryDesktopWindow function, or is HWND_DESKTOP, a main window is created.

If *hwndParent* is a window handle returned from WinQueryObjectWindow, or is HWND_OBJECT, an object window is created.

flStyle (ULONG) – input
Frame-window style.

This is a combination of any of the WS_* styles (see "Window Styles" in the *Presentation Manager Programming Reference Volume II*) and the FS_* (see "Frame Control Styles" in the *Presentation Manager Programming Reference Volume II*) frame styles.

The interpretation of the parameters is affected by the use of all the styles, except for WS_MINIMIZED and WS_MAXIMIZED. These two styles are ignored if they are specified.

pflCreateFlags (PULONG) – input
Frame-creation flags.

This contains a combination of any of the FCF_* flags.

The interpretation of the parameters is affected by the use of these flags; see "Frame Creation Flags" in the *Presentation Manager Programming Reference Volume II*.

pszClassClient (PSZ) – input
Client-window class name.

If *pszClassClient* is not a zero-length string, a client window of style *flStyleClient* and class *pszClassClient* is created. *pszClassClient* is either an application specified name

as defined by WinRegisterClass or the name of a preregistered WC_* class; see "Control Window Message Processing" in the *Presentation Manager Programming Reference Volume II*. Preregistered class names are of the form "#nnnnn," where nnnnn is 1 through 5 digits corresponding to the value of the WC_* class name constant.

If *pszClassClient* is NULL, no client area is created.

This parameter can also be specified directly as a WC_* constant.

pszTitle (PSZ) – input
Title-bar text.

This is ignored if FCF_TITLEBAR (or FCF_STANDARD) is not specified in *pflCreateFlags*.

flStyleClient (ULONG) – input
Client-window style.

This is ignored if *pszClassClient* is a zero-length string.

Resource (HMODULE) – input
Resource identifier.

This is ignored unless FCF_MENU, FCF_STANDARD, FCF_ACCELTABLE, or FCF_ICON is specified.

NULLHANDLE	Resource definitions are contained in the application .EXE file.
Other	The module handle returned by the DosLoadModule or DosQueryModuleHandle call of the Dynamic Link Library (DLL) containing the resource definitions.

ulId (ULONG) – input
Frame-window identifier.

The identifier within the resource definition of the required resource.

It is the responsibility of the application to ensure that all of the resources related to one frame window have the same *ulId* value. It must be greater or equal to 0 and less or equal to 0xFFFF.

phwndClient (PHWND) – output
Client-window handle.

This is returned if a client window is created.

Returns

hwndFrame (HWND) – returns
Frame-window handle.

This is NULLHANDLE if no window is created.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.
PMERR_INVALID_INTEGER_ATOM (0x1016)	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME (0x1015)	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND (0x1017)	The specified atom name is not in the atom table.

Remarks

The window is created with zero width and depth and positioned at the bottom left of the *hwndParent*, unless FCF_SHELLPOSITION is specified, in which case the size and position are set by the shell. The window can be positioned and sized by use of WinSetWindowPos.

If WS_VISIBLE is set, the frame window is created visible. It is recommended that standard windows that are not main windows are created with WS_VISIBLE not set.

hwndFrame is the window handle of the frame window, that is, the window of class WC_FRAME, and has a parent of *hwndParent*.

The frame window is created with identity *ulId*, all the component windows, known as the frame controls, have the standard window identifiers FID_*; see "Frame Control Window Processing" in the *Presentation Manager Programming Reference Volume II*. The identifier FID_CLIENT is used for the client area of the window.

It may be necessary to change the *ulId* of the frame window after it has been created, so that another frame window can be created with the same resource tables, and still maintain distinct window identities. This can be achieved with the WinSetWindowUShort call.

Some combinations of frame control flags are valid, but leave visual holes in the frame window. Specifically, if the *pflCreateFlags* parameter specifies any of FCF_SYSMENU, FCF_MINBUTTON, FCF_MAXBUTTON or FCF_MINMAX, but not FCF_TITLEBAR, the area of the top title line between the optional system menu and the minimize/maximize icons is not drawn by the default frame window procedure.

None of the following can be used with WinCreateStdWindow:

- WS_CLIPCHILDREN for the frame style
- WS_CLIPSIBLINGS for the style of the client window or any of the frame control windows
- CS_CLIPSIBLINGS for the class style of the window.

If any of the above are specified, the window is not redrawn correctly. Any style can be used for the children of the client. If it really is required that a client or a frame control is CLIPSIBLINGS, the application must ensure that it is in front of the client and all the other frame controls, for it to be drawn.

Related Functions

- WinCalcFrameRect
- WinCreateFrameControls
- WinCreateWindow
- WinDefWindowProc
- WinDestroyWindow
- WinQueryClassInfo
- WinQueryClassName
- WinRegisterClass
- WinSubclassWindow

Example Code

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. If you use this constant without an accelerator table for example, the function will fail.

```
#define INCL_WINFRAMEMGR        /* Window Frame Functions       */
#include <os2.h>
#define IDM_RESOURCE 1
HAB   hab;              /* Anchor-block handle                  */
CHAR szClassName[] = "Generic"; /* window class name            */
HWND hwndClient;        /* handle to the client                 */
HWND hwndFrame;         /* handle to the frame                  */
PFNWP GenericWndProc;
BOOL GenericInit()
{
    ULONG flStyle;

    flStyle = FCF_STANDARD;
    if (!WinRegisterClass(hab, szClassName, GenericWndProc, 0L, 0))
        return (FALSE);

    hwndFrame = WinCreateStdWindow(HWND_DESKTOP,
        0L,                     /* frame-window style           */
        &flStyle,               /* window style                 */
        szClassName,            /* class name                   */
        "Generic Application",  /* window title                 */
        0L,                     /* default client style         */
        NULLHANDLE,             /* resource in executable file  */
        IDM_RESOURCE,           /* resource id                  */
        &hwndClient);           /* receives client window handle */

    if (!hwndFrame)
        return (FALSE);
    else {
        .
        . /* other initialization code */
        .
```

WinCreateSwitchEntry

This function adds an entry to the Window List.

Syntax

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HSWITCH WinCreateSwitchEntry (HAB hab, PSWCNTRL pswctlSwitchData)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pswctlSwitchData (PSWCNTRL) – input
Switch data.

Contains information about the newly created Window List entry.

If the *szSwtitle* field of the SWCNTRL structure is NULL, the system uses the name under which the application is started. This only applies for OS/2 programs, and only for the first call to this function since the program started. Otherwise, a NULL entry name is invalid.

Leading and trailing blanks are removed from the title, which, if necessary, is also truncated to 60 characters.

If the *hprog* field of the SWCNTRL structure is NULLHANDLE, the value used by the system when the program was loaded (if it has been loaded) is substituted.

If the *idProcess* field of the SWCNTRL structure is 0, the current process ID is used.

If the *idSession* field of the SWCNTRL structure is 0, the current session ID is used.

Returns

hswitchSwitch (HSWITCH) – returns
Handle to the newly created Window List entry.

There is a system limit to the number of Window List entries. However, this is a large number (several hundred) and is unlikely to be reached in practice since other system limits, such as memory size, are likely to be reached first.

NULLHANDLE	Error occurred
Other	Handle to the newly created Window List entry.

Possible returns from WinGetLastError

PMERR_NO_SPACE (0x1201)	The limit on the number of Window List entries has been reached with WinAddSwitchEntry.
PMERR_INVALID_WINDOW (0x1206)	The window specified with a Window List call is not a valid frame window.
PMERR_INVALID_SESSION_ID (0x120B)	The specified session identifier is invalid. Either zero (for the application's own session) or a valid identifier must be specified.

Remarks

Both this function and the WinRemoveSwitchEntry function are not required if the main window is created with the frame creation flags FCF_TASKLIST or FCF_STANDARD, as these styles automatically update the Window List when the main window is created or destroyed, or when its title changes (see also WinAddSwitchEntry).

Related Functions

- WinAddSwitchEntry
- WinChangeSwitchEntry
- WinQuerySessionTitle
- WinQuerySwitchEntry
- WinQuerySwitchHandle
- WinQuerySwitchList
- WinQueryTaskSizePos
- WinQueryTaskTitle
- WinRemoveSwitchEntry
- WinSwitchToProgram

Example Code

This example creates a task-list entry for program name "Generic: NEW.APP".

```
#define INCL_WINSWITCHLIST      /* Window Switch List Functions */
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */

#include <os2.h>

HSWITCH hswitch;        /* task-list entry handle               */
SWCNTRL swctl;          /* switch-control data                  */
PID pid;                /* process id                           */
HAB hab;                /* anchor-block handle                  */
HWND hwndFrame;         /* frame handle                         */

hab = WinQueryAnchorBlock(hwndFrame);      /* gets anchor block */
WinQueryWindowProcess(hwndFrame, &pid, NULL);/* gets process id */

/* initialize switch structure */
swctl.hwnd = hwndFrame;                    /* window handle      */
swctl.hwndIcon = NULLHANDLE;               /* icon handle        */
swctl.hprog = NULLHANDLE;                  /* program handle     */
swctl.idProcess = pid;                     /* process identifier */
swctl.idSession = 0;                       /* session identifier */
swctl.uchVisibility = SWL_VISIBLE;         /* visibility         */
swctl.fbJump = SWL_JUMPABLE;               /* jump indicator     */
strcpy(swctl.szSwtitle,"Generic: NEW.APP"); /* program name      */

hswitch = WinCreateSwitchEntry(hab, &swctl);
```

WinCreateWindow

This function creates a new window of class *pszClass* and returns *hwnd*.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinCreateWindow (HWND hwndParent, PSZ pszClass, PSZ pszName,
                      ULONG flStyle, LONG x, LONG y, LONG cx, LONG cy,
                      HWND hwndOwner, HWND hwndInsertBehind,
                      ULONG id, PVOID pCtlData, PVOID pPresParams)
```

Parameters

hwndParent (HWND) – input
Parent-window handle.

If *hwndParent* is a desktop window handle, or is HWND_DESKTOP, a main window is created.

If *hwndParent* is HWND_OBJECT, or is a window handle returned by WinQueryObjectWindow, an object window is created.

pszClass (PSZ) – input
Registered-class name.

pszClass is either an application-specified name as defined by WinRegisterClass or the name of a preregistered WC_* class; see "Control Window Message Processing" in the *Presentation Manager Programming Reference Volume II*.

This parameter can also be specified directly as a WC_* constant.

pszName (PSZ) – input
Window text.

The actual structure of the data is class-specific. It is usually a null-terminated string that is often displayed in the window.

flStyle (ULONG) – input
Window style.

x (LONG) – input
x-coordinate of window position.

The value is in window coordinates relative to the origin of the parent window.

y (LONG) – input
y-coordinate of window position.

The value is in window coordinates relative to the origin of the parent window.

cx (LONG) – input
Width of window, in window coordinates.

cy (LONG) – input
Height of window, in window coordinates.

hwndOwner (HWND) – input
Owner-window handle.

Windows that send messages send them to their owner, as defined by this parameter. When an owner window is destroyed, all windows owned by it are also destroyed. The owner window must belong to the current thread.

hwndInsertBehind (HWND) – input
Sibling-window handle.

This is the sibling window behind which *hwnd* is placed. If this parameter is HWND_TOP or HWND_BOTTOM, *hwnd* is placed on top of all, or behind all of its siblings. This parameter must be HWND_TOP, HWND_BOTTOM, or a child of *hwndParent*.

id (ULONG) – input
Window identifier.

A value given by the application, that enables specific children of a window to be identified. For example, the controls of a dialog have unique identifiers so that an owner can distinguish which control has notified it. Window identifiers are also used for frame windows. It must be greater or equal to 0 and less or equal to 0xFFFF.

pCtlData (PVOID) – input
Pointer to control data.

Pointer to a Control-Data data structure.

The data referenced by this pointer is class-specific data passed to the window procedure by the WM_CREATE message.

This parameter MUST be a pointer rather than a long.

The first 2 bytes in the data referenced by the pointer must be the total size of the data referenced by the pointer (for example see the ENTRYFDATA or the FRAMECDATA structures). PM requires this information to enable it to ensure that the referenced data is accessible to both 16-bit and 32-bit code.

pPresParams (PVOID) – input
Presentation parameters.

This is class-specific presentation data passed to the window procedure by the WM_CREATE message.

Returns

hwnd (HWND) – returns
Window handle.

NULLHANDLE	Error occurred
Other	Window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.
PMERR_NO_MSG_QUEUE (0x1036)	
PMERR_INVALID_PARAMETERS (0x1208)	An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range –32 768 to +32 767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

Remarks

The appearance and behavior of a window are determined by its style, which is the combination of the style established by *pszClass* and *flStyle* ORed together. Any of the standard styles WS_* (see "Window Styles" in the *Presentation Manager Programming Reference Volume II*) can be used in addition to any class-specific styles that may be defined.

A window is usually created enabled and invisible. For more information on the initial state of a created window, see the list of the standard window styles.

Messages may be received from other processes or threads when this function is called.

This function sends the WM_CREATE message to the window procedure of the window being created.

This function sends the WM_ADJUSTWINDOWPOS message after the WM_CREATE message, and before the window is displayed (if applicable). The values passed are those given to the WinCreateWindow function. If the window has style WS_VISIBLE, the window is created visible.

The WM_SIZE message is not sent by the WinCreateWindow function while the window is being created. Any required size processing can be performed during the processing of the WM_CREATE message.

Because windows are often created with zero height or width and sized later, it is good practice not to perform any size-related processing if the size of the window is zero.

If the WinCreateWindow function is called for a window of class WC_FRAME, the controls specified by *flCreateFlags* are created but not formatted. The frame is formatted when a WM_FORMATFRAME message is received but this is not sent during window creation. To cause the frame to format, either a WM_FORMATFRAME message must be sent, or the window position adjusted using the WinSetWindowPos function call which sends a WM_SIZE message if the position or size is changed.

The only limitation to the size and position specified for a window is the number range allowed for the size and position parameters; that is, an application can create windows that are larger than the screen or that are positioned partially or totally off the screen. However, the user interface for manipulation of window sizes and positions is affected if part or all of the window is off the screen.

It is recommended that part of the title bar be left on the screen, if the window has one, to enable the user to move the window around.

When a WC_MENU window is created with this call, *pCtlData* is assumed to be a menu template, which is used to create the menu. If *pCtlData* is NULL, an empty menu is created.

Related Functions

- WinCalcFrameRect
- WinCreateFrameControls
- WinCreateStdWindow
- WinDefWindowProc
- WinDestroyWindow
- WinQueryClassInfo
- WinQueryClassName
- WinRegisterClass
- WinSubclassWindow

Related Messages

- WM_ADJUSTWINDOWPOS
- WM_CREATE
- WM_FORMATFRAME
- WM_SIZE

Example Code

This example creates a list box window named "List Box" as a child of the desktop located at (0,0), of size 200x100.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINLISTBOXES       /* List Box definitions         */
#include <os2.h>

HWND  hwnd;             /* cursor display window                */
ULONG ListBoxId;        /* window id  (supplied by application) */
ULONG flStyle;          /* window style                         */

flStyle = WS_VISIBLE;                   /* create window visible  */

/* create button window */
hwnd = WinCreateWindow(HWND_DESKTOP,  /* parent window          */
                       WC_LISTBOX,    /* class name             */
                       "List Box",    /* window text            */
                       flStyle,       /* window style           */
                       0, 0,          /* position (x,y)         */
                       200, 100,      /* size (width,height)    */
                       NULLHANDLE,    /* owner window           */
                       HWND_TOP,      /* sibling window         */
                       ListBoxId,     /* window id              */
                       NULL,          /* control data           */
                       NULL);         /* presentation parms     */
```

WinDdeInitiate

This function is issued by a client application to one or more other applications, to request initiation of a dynamic data exchange conversation with a national language conversation context.

Syntax

```
#define INCL_WINDDE /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinDdeInitiate  (HWND hwndClient, PSZ pszAppName, PSZ pszTopicName,
                      PCONVCONTEXT pContext)
```

Parameters

hwndClient (HWND) – input
Client's window handle.

pszAppName (PSZ) – input
Application name.

This is the name of the desired server application. If it is a zero-length string, any application can respond.

Application names may not contain slashes or backslashes.

pszTopicName (PSZ) – input
Topic name.

This is the name of the desired topic. If it is a zero-length string, each responding application will respond once for each topic which it can support.

pContext (PCONVCONTEXT) – input
Conversation context.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion. The WM_DDE_INITIATE message is successfully sent to all appropriate windows.
FALSE	Error occurred.

Remarks

This function sends a WM_DDE_INITIATE message to all top level frame windows. These are windows registered with CS_FRAME, whose parent is the desktop window. No message is sent to object windows.

The WinDdeInitiate function does not return to the client application until all receiving applications have, in sequence, processed the WM_DDE_INITIATE message, and the client application has received all the corresponding WM_DDE_INITIATEACK messages (see WinDdeRespond).

To support DDE conversations between applications running in different memory models (16-bit and 32-bit) it is necessary to process all DDE messages in the application window procedure. The use of the WinDispatchMsg function ensures that conversion is performed on memory or segment addresses.

Related Functions

- WinDdePostMsg
- WinDdeRespond

Related Messages

- WM_DDE_INITIATE
- WM_DDE_INITIATEACK

Example Code

This example uses WinDdeInitiate to initiate—during the creation of a client window—a dynamic data exchange (DDE) conversation with any available server applications, asking that the server applications respond for each topic they can support. It also allocates the shared memory that will be used once the conversation is established.

```
#define INCL_WINDDE             /* Window DDE Functions         */
#define INCL_DOSMEMMGR          /* Memory Manager values        */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
HWND  hwndClient;       /* client window                        */
char  pszAppName[15]="";/* server application                   */
char  pszTopicName[15]="";/* topic                              */
CONVCONTEXT Context;
PDDESTRUCT  pddeData;   /* DDE structure                        */

case WM_CREATE:
     /* issue DDE initialize call */
     fSuccess = WinDdeInitiate(hwndClient, pszAppName,
                               pszTopicName, &Context);

     /* allocate shared memory for conversation data */
     DosAllocSharedMem((PVOID)pddeData,"DDESHAREMEM",
                       sizeof(DDESTRUCT) + 50,
                       PAG_READ | PAG_WRITE | PAG_COMMIT |
                       OBJ_GIVEABLE | OBJ_GETTABLE);
```

WinDdePostMsg

This function is issued by an application to post a message to another application with which it is carrying out a dynamic data exchange conversation with a national language conversation context.

Syntax

```
#define INCL_WINDDE /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinDdePostMsg  (HWND hwndTo, HWND hwndFrom, ULONG usMsgId,
                     PDDESTRUCT pData, ULONG ulOptions)
```

Parameters

hwndTo (HWND) – input
Window handle of target.

hwndFrom (HWND) – input
Window handle of originator.

usMsgId (ULONG) – input
Message identifier.

Identifies the message to be posted.

The following messages are valid:

WM_DDE_ACK
WM_DDE_ADVISE
WM_DDE_DATA
WM_DDE_EXECUTE
WM_DDE_POKE
WM_DDE_REQUEST
WM_DDE_TERMINATE
WM_DDE_UNADVISE.

pData (PDDESTRUCT) – input
Pointer to the DDE control structure being passed.

ulOptions (ULONG) – input
Options.

It must be one of the following values:

DDEPM_RETRY — This controls what happens if the message cannot be posted because the destination queue is full.

If this option is set, then message posting is retried at 1-second intervals, until the message is posted successfully. In this case,

this function dispatches any messages in the queue of the application issuing this function, by calling the WinPeekMsg and WinDispatchMsg functions in a loop, so that messages sent by other applications can be received. This means that the application can continue to receive DDE messages (or other kinds of messages), while attempting to post DDE messages, thereby preventing deadlock between two applications whose queues are full and who are both attempting to post a message to each other with this option set.

Applications which rely on inspecting messages prior to issuing the WinPeekMsg function can either, use the WinSetHook function and detect the above situation in the invoked hook procedure by testing the MSGF_DDEPOSTMSG value of the *msgf* parameter, or not use this option, in order to avoid the deadlock situation.

If this option is not set, then this function returns FALSE without retrying.

Note: If the message posting fails for any other reason (for example, an invalid window handle is specified), this function returns FALSE even if this option has been selected.

DDEPM_NOFREE This option prevents the WinDdePostMsg call from freeing the shared memory block passed in on the pData parameter. If this option is used, the caller is responsible for freeing the memory block at some subsequent time (for example, the same memory block could be used in multiple calls to WinDdePostMsg and then freed once at the end of those calls.

If this option is not specified, the DDE structure will be freed.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

To support DDE conversations between applications running in different memory models (16-bit and 32-bit) it is necessary to process all DDE messages in the application window procedure. The use of the WinDispatchMsg function ensures that conversion is performed on memory or segment addresses.

Related Functions

- WinDdeInitiate
- WinDdeRespond

Related Messages

- WM_DDE_ACK
- WM_DDE_ADVISE
- WM_DDE_DATA
- WM_DDE_EXECUTE
- WM_DDE_POKE
- WM_DDE_REQUEST
- WM_DDE_TERMINATE
- WM_DDE_UNADVISE

Example Code

This example uses WinDdePostMsg to request a security item from the server once it has received an acknowledgement (via WM_DDEINITIATEACK) to the WinDdeInitiate call. Note the use of the shared memory segment to pass and receive necessary information.

```
#define INCL_WINDDE             /* Window DDE Functions         */
#define INCL_DOSMEMMGR          /* Memory Manager values        */
#include <os2.h>

BOOL  fSuccess;          /* success indicator                   */
HWND  hwndClient;        /* client window                       */
HWND  hwndServer;        /* server window                       */
CHAR  pszAppName[15]="";/* server application                   */
CHAR  pszTopicName[15]="";/* topic                              */
HWND  hwndTo;            /* target window                       */
HWND  hwndFrom;          /* source window                       */
USHORT  usMsgId;         /* message id                          */
BOOL  fRetry;            /* retry indicator                     */
CONVCONTEXT Context;
PDDESTRUCT  pddeData;    /* DDE structure                       */
MRESULT  mresReply;      /* message return data                 */

case WM_CREATE:
     fSuccess = WinDdeInitiate(hwndClient, pszAppName,
                               pszTopicName, &Context);
     DosAllocSharedMem((PVOID)pddeData,"DDESHAREMEM",
                       sizeof(DDESTRUCT) + 50,
                       PAG_READ | PAG_WRITE | PAG_COMMIT |
                       OBJ_GIVEABLE | OBJ_GETTABLE);
```

```
case WM_DDE_INITIATEACK:
     /* issue a request message to DDE partner */
     usMsgId = WM_DDE_REQUEST;

     /* initialize DDE conversation structure */
     pddeData->cbData = sizeof(DDESTRUCT);      /* Total length  */
     pddeData->fsStatus = DDE_FACK;    /* Status - positive ack  */
     pddeData->usFormat = DDEFMT_TEXT;          /* Data format  */
     pddeData->offszItemName = sizeof(DDESTRUCT);/* Offset to item  */

     /* set name of item to 'Security', copying the information to
        the shared memory at the end of pddeData */
     strcpy((BYTE *)pddeData + pddeData->offszItem,
            SZDDESYS_ITEM_SECURITY);

     /* Offset to beginning of data (notice additional offset due
        to item information) */

     pddeData->offabData = sizeof(DDESTRUCT) +
                           strlen(SZDDESYS_ITEM_SECURITY);

     /* set name of item to 'Security', copying the information to
        the shared memory at the end of pddeData */
     strcpy((BYTE *)pddeData + pddeData->offszItem,
            SZDDESYS_ITEM_SECURITY);

     fSuccess = WinDdePostMsg(hwndTo, hwndFrom, usMsgId, pddeData,
                              fRetry);
```

WinDdeRespond

This function is issued by a server application to indicate that it can support a dynamic data exchange conversation on a particular topic with a national language conversation context.

Syntax

```
#define INCL_WINDDE /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
MRESULT WinDdeRespond  (HWND hwndClient, HWND hwndServer,
                        PSZ pszAppName, PSZ pszTopicName,
                        PCONVCONTEXT pContext)
```

Parameters

hwndClient (HWND) – input
Client's window handle.

hwndServer (HWND) – input
Server's window handle.

If a server application is responding for more than one topic, it must use a different window for each topic.

pszAppName (PSZ) – input
Application name.

This is the name of the responding server application. It must not be a zero-length string or null.

Application names may not contain slashes or backslashes.

pszTopicName (PSZ) – input
Topic name.

This is the name of the topic which the server is willing to support. It must not be a zero-length string or null.

pContext (PCONVCONTEXT) – input
Conversation context.

Returns

mresReply (MRESULT) – returns
Message return data.

Remarks

This function is issued by a server application after receiving a WM_DDE_INITIATE message that identifies this server application (or indicates that any application can respond), and also either identifies a particular topic which the server can support, or asks for all supported topics (see WinDdeInitiate). This function sends a WM_DDE_INITIATEACK message back to the client, that is the sender of the WM_DDE_INITIATE message.

If the server application can respond, it issues this function once if a specific topic was requested, or once for each topic which it can support, if all supported topics were requested.

A DDE conversation is initiated each time this function is successfully issued. The client is expected to terminate all unwanted conversations. Once a conversation is initiated, it is controlled by the client issuing WinDdePostMsg functions.

To support DDE conversations between applications running in different memory models (16-bit and 32-bit) it is necessary to process all DDE messages in the application window procedure. The use of the WinDispatchMsg function ensures that conversion is performed on memory or segment addresses.

Related Functions

- WinDdeInitiate
- WinDdePostMsg

Related Messages

- WM_DDE_INITIATE
- WM_DDE_INITIATEACK

Example Code

This example uses WinDdeRespond to respond to an initiate message (WM_DDEINITIATE) generated by the client window issuing WinDdeInitiate. Here, the server responds as a DDE Server that supports a System topic.

```
#define INCL_WINDDE             /* Window DDE Functions        */
#include <os2.h>

HWND  hwndClient;       /* client window                       */
HWND  hwndServer;       /* server window                       */
char  pszAppName[15]="DDE Server"; /* server application        */
char  pszTopicName[15]=SZDDESYS_TOPIC; /* topic ('System')     */
MRESULT  mresReply;     /* message return data                 */
CONVCONTEXT Context;
case WM_DDE_INITIATE:
     mresReply = WinDdeRespond(hwndClient, hwndServer, pszAppName,
                               pszTopicName, &Context);
```

WinDefDlgProc

This function invokes the default dialog procedure with *hwndDlg*, *msg*, *mp1*, and *mp2*.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
MRESULT WinDefDlgProc  (HWND hwndDlg, ULONG msg, MPARAM mp1,
                        MPARAM mp2)
```

Parameters

hwndDlg (HWND) – input
Dialog-window handle.

msg (ULONG) – input
Message identity.

mp1 (MPARAM) – input
Parameter 1.

mp2 (MPARAM) – input
Parameter 2.

Returns

mresReply (MRESULT) – returns
Message-return data.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The action taken by the default dialog procedure is such that the values passed in *mp1* and *mp2*, and the values returned in *mresReply* are defined for each *msg*.

The default dialog procedure provides default processing for any dialog window messages that an application chooses not to process. It can be used to ensure that every message is processed and is called with the same parameters that were received by the dialog procedure.

The action of the WinDefDlgProc function on receiving messages is precisely the same as for the frame window procedure except for WM_CLOSE messages where WinDismissDlg will be called. If an application processes a message instead of sending it to the WinDefDlgProc

function, it may be required to perform some or all of the frame window procedure actions for itself.

Related Functions

- WinCreateDlg
- WinDismissDlg
- WinDlgBox
- WinGetDlgMsg
- WinLoadDlg
- WinProcessDlg

Example Code

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.

```
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#include <os2.h>

MRESULT AboutDlg(HWND hwnd, ULONG  ulMessage, MPARAM mpParam1,
                            MPARAM mpParam2)
{
    switch (ulMessage) {

        /*
         * Process whatever messages you want here and send the rest
         * to WinDefDlgProc.
         */

        default:
            return (WinDefDlgProc(hwnd, ulMessage, mpParam1, mpParam2));
    }
}
```

WinDefFileDlgProc

This function is the default dialog procedure for the file dialog.

Syntax

```
#define INCL_winstdfile
#include <os2.h>
MRESULT WinDefFileDlgProc  (HWND hwnd, ULONG msg, MPARAM mp1,
                            MPARAM mp2)
```

Parameters

hwnd (HWND) – input
Dialog-window handle.

msg (ULONG) – input
Message identity.

mp1 (MPARAM) – input
Parameter 1.

mp2 (MPARAM) – input
Parameter 2.

Returns

mresReply (MRESULT) – returns
Message-return data.

Remarks

All unprocessed messages in a custom dialog procedure should be passed to the default file dialog procedure so that the dialog can implement its default behavior.

Example Code

This example uses the default dialog procedure for the file dialog to cause default processing of unprocessed dialog messages.

```
#define INCL_WINSTDFILE  /* Window Standard File Functions       */
#include <os2.h>

MRESULT MyFileDlgProc(HWND  hwnd, ULONG  msg, MPARAM mpParam1,
                      MPARAM mpParam2)
{
switch(msg)
   {

   /*************************************************************/
   /* Process user-supported messages                           */
   /*************************************************************/
   .
   .
   .
   default:
      return (WinDefFileDlgProc(hwnd, msg, mpParam1, mpParam2));
   }
}
```

WinDefFontDlgProc

This function is the default dialog procedure for the font dialog.

Syntax

```
#define INCL_winstdfont
#include <os2.h>
MRESULT WinDefFontDlgProc  (HWND hwnd, ULONG msg, MPARAM mp1,
                            MPARAM mp2)
```

Parameters

hwnd (HWND) – input
Dialog-window handle.

msg (ULONG) – input
Message identity.

mp1 (MPARAM) – input
Parameter 1.

mp2 (MPARAM) – input
Parameter 2.

Returns

mresReply (MRESULT) – returns
Message-return data.

Remarks

All unprocessed messages in a custom dialog procedure should be passed to the default font dialog procedure so that the dialog can implement its default behavior.

Example Code

This example uses the default dialog procedure for the font dialog to cause default processing of unprocessed dialog messages.

```
#define INCL_WINSTDFONT  /* Window Standard Font Functions       */
#include <os2.h>

MRESULT MyFontDlgProc(HWND hwnd, ULONG msg, MPARAM mpParam1,
                      MPARAM mpParam2)
{
switch(msg)
   {

   /************************************************************/
   /* Process user-supported messages                          */
   /************************************************************/
   .
   .
   .
   default:
      return (WinDefFontDlgProc(hwnd, msg, mpParam1, mpParam2));
   }
}
```

WinDefWindowProc

This function invokes the default window procedure.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
MRESULT WinDefWindowProc  (HWND hwnd, ULONG ulMsgid,
                           MPARAM mpParam1, MPARAM mpParam2)
```

Parameters

hwnd (HWND) – input
Window handle.

ulMsgid (ULONG) – input
Message identity.

mpParam1 (MPARAM) – input
Parameter 1.

mpParam2 (MPARAM) – input
Parameter 2.

Returns

mresReply (MRESULT) – returns
Message-return data.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The default window provides default processing for any window messages that an application chooses not to process. It can be used to ensure that every message is processed. This function should be made with the same parameters as those received by the window procedure.

The action taken by the default window procedure, the values passed in *mpParam1*, *mpParam2* and the values returned in *mresReply* are defined for each *ulMsgid* (see "Introduction to Message Processing" in the *Presentation Manager Programming Reference Volume II*).

Related Functions

- WinCalcFrameRect
- WinCreateFrameControls
- WinCreateStdWindow
- WinCreateWindow
- WinDestroyWindow
- WinQueryClassInfo
- WinQueryClassName
- WinRegisterClass
- WinSubclassWindow

Example Code

This example uses the default window procedure, called by WinDefWindowProc, for default processing of non supported window messages.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

MRESULT GenericWndProc(HWND  hwnd, ULONG  ulMsgid, MPARAM mpParam1,
                       MPARAM mpParam2)
{
switch(ulMsgid)
   {
   /*
   .
   . process user supported messages
   .
   */

   default:
     return (WinDefWindowProc(hwnd, ulMsgid, mpParam1, mpParam2));
   }
}
```

WinDeleteAtom

This function deletes an atom from an atom table.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ATOM WinDeleteAtom  (HATOMTBL hatomtblAtomTbl, ATOM atom)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

This is the handle returned from a previous WinCreateAtomTable or WinQuerySystemAtomTable function.

atom (ATOM) – input
Atom identifying the atom to be deleted.

Returns

rc (ATOM) – returns
Return code.

0	Call successful
Other	The call fails and the atom has not been deleted, in which case this is equal to the *atom* parameter.

Possible returns from WinGetLastError

PMERR_INVALID_HATOMTBL (0x1013)	An invalid atom-table handle was specified.
PMERR_INVALID_ATOM (0x1014)	The specified atom does not exist in the atom table.

Remarks

If the passed atom is an integer atom, 0 is returned. If it is not an integer atom and it is a valid atom for the given atom table, that is, it has an atom name and use count, its use count is decremented by one and 0 is returned. If the use count has been decremented to zero, the atom name and use count are removed from the atom table.

Related Functions

- WinAddAtom
- WinCreateAtomTable
- WinDestroyAtomTable
- WinFindAtom
- WinQueryAtomLength
- WinQueryAtomName
- WinQueryAtomUsage
- WinQuerySystemAtomTable

Example Code

This example deletes a newly created atom in an Atom Table based on the atom value returned by WinAddAtom.

```
#define INCL_WINATOM            /* Window Atom Functions        */
#include <os2.h>

ATOM  atom;             /* new atom value                       */
ATOM  atomDelete;       /* result of atom delete                */
HATOMTBL  hatomtblAtomTbl; /* atom-table handle                 */
char   pszAtomName[10]; /* atom name                            */
ULONG   ulInitial = 0;  /* initial atom table size (use default)*/
ULONG   ulBuckets = 0;  /* size of hash table (use default)     */

/* create atom table of default size */
hatomtblAtomTbl = WinCreateAtomTable(ulInitial, ulBuckets);

/* define name for new atom and add to table */
strcpy(pszAtomName,"newatom");
atom = WinAddAtom(hatomtblAtomTbl, pszAtomName);

atomDelete = WinDeleteAtom(hatomtblAtomTbl, atom);
```

WinDeleteLboxItem

This macro deletes the indexed item from the List Box. It returns the number of items left.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinDeleteLboxItem  (HWND hwndLbox, LONG index)
```

Parameters

hwndLbox (HWND) – input
Listbox handle.

index (LONG) – input
Index of the listbox item.

Returns

lltems (LONG) – returns
Number of items left.

Remarks

This macro is defined as:

```
#define WinDeleteLboxItem(hwndLbox, index)    \
        ((LONG)WinSendMsg(hwndLbox,           \
                          LM_DELETEITEM,      \
                          MPFROMLONG(index),  \
                          (MPARAM)NULL))
```

This macro requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- LM_DELETEITEM

Example Code

This example responds to an item in the list box being selected (LN_SELECT, WM_CONTROL message) by deleting the selected item using WinDeleteLboxItem.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINLISTBOXES       /* Window List Box definitions  */
#include <os2.h>

LONG    lIndex;         /* selected item index                  */
LONG    lLeft;          /* items left after delete              */
HWND    hwndLbox;       /* list box window handle               */
MPARAM  mpParam1;       /* Parameter 1 (rectl structure)        */
MPARAM  mpParam2;       /* Parameter 2 (frame boolean)          */

case WM_CONTROL:
     /* switch on control code */
     switch(SHORT2FROMMP(mpParam1))
        {
        case LN_SELECT:
             hwndLbox = HWNDFROMMP(mpParam2);

             /* query index of selected item */
             lIndex = WinQueryLboxSelectedItem(hwndLbox);

             /* delete selected listbox item */
             lLeft = WinDeleteLboxItem(hwndLbox, lIndex);
             break;
        }
```

WinDeleteLibrary

This function deletes the library *hlibLibhandle*, which is previously loaded by the WinLoadLibrary function.

Syntax

```
#define INCL_WINLOAD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinDeleteLibrary (HAB hab, HLIB hlibLibhandle)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hlibLibhandle (HLIB) – input
Library handle to be deleted.

This handle was previously loaded by the the WinLoadLibrary function.

Returns

rc (BOOL) – returns
Library-deleted indicator.

TRUE	Library successfully deleted
FALSE	Library not successfully deleted.

Related Functions

- WinDeleteProcedure
- WinLoadLibrary
- WinLoadProcedure

Example Code

This example deletes the library identified by the library handle returned from WinLoadLibrary.

```
#define INCL_WINLOAD            /* Window Load Functions        */
#include <os2.h>

BOOL    fSuccess;       /* success indicator                    */
HAB     hab;            /* anchor-block handle                  */
HLIB    hlib;           /* library handle                       */

fSuccess = WinDeleteLibrary(hab, hlib);
```

WinDeleteProcedure

This function deletes the window or dialog procedure that was previously loaded using the WinLoadProcedure function.

Syntax

```
#define INCL_WINLOAD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinDeleteProcedure  (HAB hab, PFNWP pwndproc)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pwndproc (PFNWP) – input
Window procedure identifier to be deleted.

Returns

rc (BOOL) – returns
Procedure-deleted indicator.

TRUE	Procedure successfully deleted
FALSE	Procedure not successfully deleted.

Related Functions

- WinDeleteLibrary
- WinLoadLibrary
- WinLoadProcedure

Example Code

This example deletes the procedure identified by the procedure pointer returned from WinLoadProcedure.

```
#define INCL_WINLOAD            /* Window Load Functions        */
#include <os2.h>

BOOL    fSuccess;       /* success indicator                    */
PFNWP   pWndproc;       /* procedure pointer                    */
HAB     hab;            /* anchor-block handle                  */

fSuccess = WinDeleteProcedure(hab, pWndproc);
```

WinDeregisterObjectClass

This function deregisters (removes) a workplace object class.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinDeregisterObjectClass  (PSZ pszClassName)
```

Parameters

pszClassName (PSZ) – input
Pointer to class name.

A pointer to a zero-terminated string which contains the name of the object class being removed from the workplace.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

Workplace object classes are not deleted unless the application issues a WinDeregisterObjectClass. Object classes will be automatically registered when a dynamic-link library containing an object definition is added to the system. The only advantage of deregistering an object class is to optimize the system performance. All registered classes are maintained in the OS2.INI and are cached upon system initialization. If the class is no longer needed, it should be removed.

Related Functions

- WinCreateObject
- WinRegisterObjectClass
- WinReplaceObjectClass

WinDestroyAccelTable

This function destroys an accelerator table.

Syntax

```
#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinDestroyAccelTable  (HACCEL  haccelAccel)
```

Parameters

haccelAccel (HACCEL) – input
Accelerator-table handle.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HACCEL (0x101A)	An invalid accelerator-table handle was specified.

Remarks

Before an application is terminated, it should call the WinDestroyAccelTable function for every accelerator table that is created with the WinCreateAccelTable function.

Related Functions

- WinCopyAccelTable
- WinCreateAccelTable
- WinLoadAccelTable
- WinQueryAccelTable
- WinSetAccelTable
- WinTranslateAccel

Example Code

This example destroys an accelerator-table based on the handle returned from either WinCreateAccelTable or WinLoadAccelTable.

```
#define INCL_WINACCELERATORS    /* Window Accelerator Functions */
#include <os2.h>

HACCEL  hAccel;         /* Accelerator-table handle             */
BOOL  fSuccess;         /* success indicator                    */

fSuccess = WinDestroyAccelTable(hAccel);
```

WinDestroyAtomTable

This function destroys a private atom table, which is created by WinCreateAtomTable.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HATOMTBL WinDestroyAtomTable  (HATOMTBL hatomtblAtomTbl)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

This is the handle returned from a previous call to the WinCreateAtomTable function. If NULL then this function does nothing.

Returns

rc (HATOMTBL) – returns
Return code.

0	Function successful.
Other	The call fails and the atom table has not been destroyed, in which case this is equal to the *hatomtblAtomTbl* parameter.

Possible returns from WinGetLastError

PMERR_INVALID_HATOMTBL (0x1013)	An invalid atom-table handle was specified.

Remarks

This function makes no attempt to ensure that the handle to the atom table is not reused by a later call to the WinCreateAtomTable function.

The system atom table (see the WinQuerySystemAtomTable function) cannot be destroyed.

Related Functions

- WinAddAtom
- WinCreateAtomTable
- WinDeleteAtom
- WinFindAtom
- WinQueryAtomLength
- WinQueryAtomName
- WinQueryAtomUsage
- WinQuerySystemAtomTable

Example Code

This example destroys an Atom Table of one atom, based on its handle, which is returned by WinCreateAtomTable.

```
#define INCL_WINATOM            /* Window Atom Functions        */
#include <os2.h>

ATOM  atom;             /* new atom value                       */
HATOMTBL  hatomtblAtomTbl; /* atom-table handle                 */
HATOMTBL  hatomtblDestroy; /* result of destroy table           */
char   pszAtomName[10]; /* atom name                            */
USHORT  usInitial = 0;  /* initial atom table size (use default)*/
USHORT  usBuckets = 0;  /* size of hash table (use default)     */

/* create atom table of default size */
hatomtblAtomTbl = WinCreateAtomTable(usInitial, usBuckets);

/* define name for new atom and add to table */
strcpy(pszAtomName,"newatom");
atom = WinAddAtom(hatomtblAtomTbl, pszAtomName);

hatomtblDestroy = WinDestroyAtomTable(hatomtblAtomTbl);
```

WinDestroyCursor

This function destroys the current cursor, if it belongs to the specified window.

Syntax

```
#define INCL_WINCURSORS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinDestroyCursor  (HWND  hwnd)
```

Parameters

hwnd (HWND) – input
Window handle to which the cursor belongs.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function has no effect if the current cursor does not belong to the specified window.

It is not necessary to call this function before calling the WinCreateCursor function.

If the cursor was created with the CURSOR_FLASH option, then the TID_CURSOR timer is also destroyed.

Related Functions

- WinCreateCursor
- WinQueryCursorInfo
- WinShowCursor

Example Code

This example destroys the cursor defined for the specified input window.

```
#define INCL_WINCURSORS         /* Window Cursor Functions      */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
HWND    hwnd;           /* cursor display window                */

fSuccess = WinDestroyCursor(hwnd);
```

WinDestroyHelpInstance

This function destroys the specified instance of the Help Manager.

Syntax

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinDestroyHelpInstance  (HWND  hwndHelpInstance)
```

Parameters

hwndHelpInstance (HWND) – input
Handle of the instance of the Help Manager to be destroyed.

This is the handle returned by the WinCreateHelpInstance call.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Related Functions

- WinAssociateHelpInstance
- WinCreateHelpInstance
- WinCreateHelpTable
- WinLoadHelpTable
- WinQueryHelpInstance

Example Code

This example shows a typical main function for an application which uses help. Following creation of the main application window the Help Manager is initialized and associated with the window. The help table is defined in the application's resources. When the window is destroyed, terminating the application, the help instance is also destroyed.

```
#define INCL=_WIN
#include <os2.h>

#define IDHT_APPLICATION        100     /* id of HELP TABLE in resource file
*/

main( int argc, char *argv[], char *envp[] )
{
   HAB  hab = WinInitialize( 0 );
   HMQ  hmq = WinCreateMsgQueue( hab, 0 );
   HWND hwnd;
   HWND hwndClient;
   HWND hwndHelp;
   QMSG qmsg;
   ULONG flStyle;
   HELPINIT helpinit;

   /* Setup the help initialization structure */
   helpinit.cb = sizeof( HELPINIT );
   helpinit.ulReturnCode =  0L;
   helpinit.pszTutorialName =  (PSZ)NULL;
   /* Help table in application resource */
   helpinit.phtHelpTable = (PHELPTABLE)MAKEULONG( IDHT_APPLICATION, 0xffff );
   helpinit.hmodHelpTableModule = NULLHANDLE;
   /* Default action bar and accelerators */
   helpinit.hmodAccelActionBarModule = NULLHANDLE;
   helpinit.idAccelTable = 0;
   helpinit.idActionBar = 0;
   helpinit.pszHelpWindowTitle = "APPNAME HELP";
   helpinit.fShowPanelId = CMIC_SHOW_PANEL_ID;
   helpinit.pszHelpLibraryName = "APPNAME.HLP";

   /* Register the class */
   if( WinRegisterClass( ... ) )
   {
      /* create the main window */
      flStyle = FCF_STANDARD;
      hwnd = WinCreateStdWindow( ... );

      if( hwnd )
      {
         /* Create and associate the help instance */
         hwndHelp = WinCreateHelpInstance( hab, &helpinit );
```

```
        if( hwndHelp && WinAssociateHelpInstance( hwndHelp, hwnd ) )
        {
           /* Process messages */
           while( WinGetMsg( hab, &qmsg, NULLHANDLE, 0, 0 ) )
           {
              WinDispatchMsg( hab, &qmsg );
           } /* endwhile */
        }

        /* Remove help instance - note: add                    */
        /*     WinAssociateHelpInstance( NULLHANDLE, hwnd );      */
        /* to WM_DESTROY processing to remove the association. */
        WinDestroyHelpInstance( hwndHelp );
     }
  }

  /* finish the cleanup and exit */
  WinDestroyMsgQueue( hmq );
  WinTerminate( hab );
}
```

WinDestroyMsgQueue

This function destroys the message queue.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinDestroyMsgQueue (HMQ hmq)
```

Parameters

hmq (HMQ) – input
Message-queue handle.

Returns

rc (BOOL) – returns
Queue-destroyed indicator.

TRUE	Queue destroyed
FALSE	Queue not destroyed.

Possible returns from WinGetLastError

PMERR_INVALID_HMQ (0x1002)	An invalid message-queue handle was specified.

Remarks

This function must be called before terminating a thread or an application. Only the thread that called WinCreateMsgQueue may call this function with that handle.

Related Functions

- WinBroadcastMsg
- WinCancelShutdown
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos

- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example destroys, using WinDestroyMsgQueue, a message queue previously created by WinCreateMsgQueue.

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

BOOL    fDestroyed;     /* success of destroy call              */
HAB     hab;            /* anchor-block handle                  */
HMQ     hmq;            /* message queue handle                 */

hab = WinInitialize(0);          /* initialize PM */

hmq = WinCreateMsgQueue(hab, 0); /* create default size queue */

/*
.
.  initialize windows, message loop
.
*/

fDestroyed = WinDestroyMsgQueue(hmq);
```

WinDestroyObject

This function is called to delete a workplace object.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinDestroyObject  (HOBJECT  object)
```

Parameters

object (HOBJECT) – input
Handle to a workplace object.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

The WinDestroyObject function will permanently remove an object that was created with the WinCreateObject function.

Using HOBJECT for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

Related Functions

- WinCreateObject
- WinSetObjectData

WinDestroyPointer

This function destroys a pointer or icon.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinDestroyPointer  (HPOINTER hptrPointer)
```

Parameters

hptrPointer (HPOINTER) – input
Handle of pointer to be destroyed.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HPTR (0x101B)	An invalid pointer handle was specified.

Remarks

A pointer can only be destroyed by the thread that created it.

The system pointers and icons must not be destroyed.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinQuerySysPointerData
- WinSetPointer
- WinSetPointerPos
- WinSetSysPointerData
- WinShowPointer

Example Code

This example destroys a bit-map pointer, created by either WinCreatePointer or WinCreatePointerIndirect, once the window has received a close message (WM_CLOSE).

```
#define INCL_WINPOINTERS        /* Window Pointer Functions     */
#define INCL_GPIBITMAPS         /* Graphics Bit-map Functions    */
#include <os2.h>
#define IDP_BITMAP 1

HPS   hps;              /* presentation-space handle            */
HWND  hwnd;             /* window handle                        */
HPOINTER  hptr;         /* bit-map pointer handle               */
HBITMAP  hbm;           /* bit-map handle                       */
BOOL  fSuccess;         /* success indicator                    */

case WM_CREATE:
     hps = WinBeginPaint(hwnd, NULLHANDLE, NULL);
     hbm = GpiLoadBitmap(hps, 0L, IDP_BITMAP, 64L, 64L);
     WinEndPaint(hps);

     hptr = WinCreatePointer(HWND_DESKTOP, hbm,
                             TRUE, /* use true (system) pointer */
                             0, 0); /* hot spot offset (0,0)    */


case WM_CLOSE:
     fSuccess = WinDestroyPointer(hptr);
```

WinDestroyWindow

This call destroys a window and its child windows.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinDestroyWindow  (HWND hwnd)
```

Parameters

hwnd (HWND) – input
Window handle.

Returns

rc (BOOL) – returns
Window-destroyed indicator.

TRUE	Window destroyed
FALSE	Window not destroyed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The window to be destroyed must have been created by the thread that is issuing this call. Before *hwnd* is itself destroyed, all windows owned by *hwnd* are also destroyed.

If *hwnd* cannot be destroyed, for example because *hwnd* is an invalid window handle or is not associated with the current thread, *rc* returns FALSE.

Note: If *hwnd* is locked, this call does not return until the window is unlocked (and destroyed).

Messages may be received from other processes or threads during the processing of this call.

If a Presentation Space is associated with the window, it is disassociated from it by this function. If the presentation space was obtained by use of GpiCreatePS then it is disassociated from the window, but not destroyed. That is, this function calls GpiAssociate to disassociate the presentation space but does not call GpiDestroyPS. If the presentation space was obtained by use of the WinGetPS function, it is released by this function; that is, this function performs the WinReleasePS function.

Messages sent by this call are:

WM_DESTROY	Always sent to the window being destroyed after the window has been hidden on the device, but before its child windows have been destroyed. The message is sent first to the window being destroyed, then to the child windows as they are destroyed. Therefore, during processing the WM_DESTROY it can be assumed that all the children still exist.
WM_ACTIVATE	Sent with *usactive* set to FALSE if the window being destroyed is the active window.
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, both the active window and the input focus window are transferred to another window when the window is destroyed. The window that becomes the active window is the next window, as defined for the "Alt+Esc" function. This usually corresponds to the next application in the sequence. The input focus transfers to whichever window the new active window decides should have it.

If a menu window is being destroyed, any bit maps associated with the menu are not deleted. They will be deleted automatically when the application terminates.

Related Functions

- WinCalcFrameRect
- WinCreateFrameControls
- WinCreateStdWindow
- WinCreateWindow
- WinDefWindowProc
- WinQueryClassInfo
- WinQueryClassName
- WinRegisterClass
- WinSubclassWindow

Related Messages

- WM_ACTIVATE
- WM_DESTROY
- WM_RENDERALLFMTS

Example Code

This example destroys the specified window and all other windows owned by that window in response to a WM_CLOSE message.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>
ULONG   fSuccess;
HWND    hwnd;           /* cursor display window                */

case WM_CLOSE:
     fSuccess = WinDestroyWindow(hwnd);
```

WinDismissDlg

This function hides the modeless dialog window, or destroys the modal dialog window, and causes the WinProcessDlg or WinDlgBox functions to return.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinDismissDlg  (HWND hwndDlg, ULONG usResult)
```

Parameters

hwndDlg (HWND) – input
Dialog-window handle.

usResult (ULONG) – input
Reply value.

Returned to the caller of the WinProcessDlg or WinDlgBox functions.

Returns

rc (BOOL) – returns
Dialog-dismissed indicator.

TRUE	Dialog successfully dismissed
FALSE	Dialog not successfully dismissed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function is required to complete the processing of a modal dialog window and is called from its dialog procedure. It is made implicitly if the dialog procedure passes a WM_COMMAND message to WinDefDlgProc or if a WM_QUIT message is encountered during a WinProcessDlg or WinGetDlgMsg function.

This function hides the dialog window, and re-enables any windows that were disabled by a WinProcessDlg or WinGetDlgMsg function.

It does not destroy the dialog window; a WinDestroyWindow function must be issued to destroy the dialog window when it is no longer needed. However, the WinDlgBox function destroys the dialog window it creates, when the dialog window is dismissed by the use of this function.

This function can be issued during the processing of the the WM_INITDLG (Default Dialogs) message.

Note: This function can be made from a modeless dialog window, although this is not necessary as there is no internal message processing loop. If it is called, the dialog window is hidden and it is the responsibility of the application to destroy the dialog window, if required.

Related Functions

- WinCreateDlg
- WinDefDlgProc
- WinDlgBox
- WinGetDlgMsg
- WinLoadDlg
- WinProcessDlg

Related Messages

- WM_COMMAND
- WM_QUIT
- WM_INITDLG (Default Dialogs)

Example Code

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 TRUE.

```
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#include <os2.h>
#define  ID_ENTER  101;
#define  ID_CANCEL 102;
MPARAM   mpParam1;
HWND     hwnd;

case WM_COMMAND:
    switch (SHORT1FROMMP(mpParam1)) {
        case ID_ENTER:     /* OK button selected  */
            WinDismissDlg(hwnd, TRUE);
            return (0L);

        case ID_CANCEL:   /* Cancel button selected */
            WinDismissDlg(hwnd, FALSE);
            return (0L);
     }
```

WinDispatchMsg

This function invokes a window procedure.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
MRESULT WinDispatchMsg  (HAB hab, PQMSG pqmsgMsg)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pqmsgMsg (PQMSG) – input
Message structure.

Returns

mresReply (MRESULT) – returns
Message-return data.

Remarks

This function is equivalent to using the WinSendMsg function with the parameters corresponding to those in *pqmsgMsg*.

The time and pointer position information within *pqmsgMsg* can be obtained by the window procedure with the WinQueryMsgTime and WinQueryMsgPos functions.

mresReply is the value returned by the invoked window procedure. For standard window classes, the values of *mresReply* are documented with the message definitions; see "Introduction to Message Processing" in the *Presentation Manager Programming Reference Volume II*.

Related Functions

- WinBroadcastMsg
- WinCancelShutdown
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg

- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example, after uses WinDispatchMsg within a WinGetMsg loop to dispatch window messages to a window procedure.

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HAB     hab;            /* anchor-block handle                  */
HMQ     hmq;            /* message queue handle                 */
QMSG    qmsg;           /* message                              */

hab = WinInitialize(0);          /* initialize PM */

hmq = WinCreateMsgQueue(hab, 0); /* create default size queue */

/*
.
.  initialize windows
.
*/

/* get and dispatch messages from queue */
while (WinGetMsg(hab, &qmsg, 0, 0, 0))
   WinDispatchMsg(hab, &qmsg);
```

WinDlgBox

This function loads and processes a modal dialog window and returns the result value established by the WinDismissDlg call.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
ULONG WinDlgBox (HWND hwndParent, HWND hwndOwner, PFNWP pfnDlgProc,
                 HMODULE hmod, ULONG idDlg, PVOID pCreateParams)
```

Parameters

hwndParent (HWND) – input
Parent-window handle of the created dialog window.

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

hwndOwner (HWND) – input
Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the WinLoadDlg function.

pfnDlgProc (PFNWP) – input
Dialog procedure for the created dialog window.

hmod (HMODULE) – input
Resource identity containing the dialog template.

NULLHANDLE	Use the application's .EXE file.
Other	Module handle returned from the DosLoadModule or DosQueryModuleHandle call.

idDlg (ULONG) – input
Dialog-template identity within the resource file.

It is also used as the identity of the created dialog window. Is must be greater or equal to 0 and less or equal to 0xFFFF.

pCreateParams (PVOID) – input
Pointer to application-defined data area.

This is passed to the dialog procedure in the WM_INITDLG message.

This parameter MUST be a pointer rather than a long.

Returns

ulResult (ULONG) – returns
Reply value.

Value established by the WinDismissDlg call or DID_ERROR if an error occurs.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_INTEGER_ATOM (0x1016)	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME (0x1015)	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND (0x1017)	The specified atom name is not in the atom table.
PMERR_RESOURCE_NOT_FOUND (0x100A)	The specified resource identity could not be found.

Remarks

The use of parameters to this function are the same as those of the WinLoadDlg function.

This function should not be used while pointing device capture is set (see WinSetCapture).

This function does not return until WinDismissDlg is called.

This function is equivalent to:

```
WinLoadDlg (., ., ., ., ., ., dlg);
WinProcessDlg (dlg, result);
WinDestroyWindow (dlg, success);
return  (result);
```

and the remarks documented under these calls also apply.

If a dialog template (typically compiled using the resource compiler) references another resource (for example an icon resource for an icon static control), this function always searches for that resource in the .EXE file. If an application wishes to keep resources referenced by a dialog template in a .DLL library, these resources must be loaded by an explicit function call during the processing of the WM_INITDLG message.

This can be considered to be a customizable "read from screen" call. The caller supplies a data buffer (the *pCreateParams* parameter), filled with initial values. It receives a return code which indicates whether the data in the buffer has been updated and validated, or whether the end user cancelled the dialog. The end user interface is encapsulated within the dialog window. The dialog template provides a view of the current state of the data buffer, the dialog procedure defines how the user can change the data. The caller need know nothing about the details of the end user interface. It makes a single "read from screen" call and continues with its work.

Note: If a dialog box or a message box is up for a window, and the parent or owner of that window is destroyed, the code following the WinDlgBox or WinMessageBox call is executed even though the parent/owner window no longer exists. This can result in accessing data that no longer exists; especially data referenced in the window words. Therefore, it is extremely important to determine the state of your child-window procedure after this function returns. The most straightforward method for doing this is to call WinQueryWindowPtr to get a pointer to the window words. If the returned pointer is NULL, then you should exit immediately. Should this be the case, the bottom-up rule (that is, the child window gets WM_DESTROY messages first, then the parent window) still applies, and it becomes the child window procedure's responsibility to exit gracefully.

Related Functions

- WinCreateDlg
- WinDefDlgProc
- WinDismissDlg
- WinGetDlgMsg
- WinLoadDlg
- WinProcessDlg

Related Messages

- WM_INITDLG

Example Code

This example processes an application-defined message (IDM_OPEN) and calls WinDlgBox to load a dialog box.

```
#define IDD_OPEN 1
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#include <os2.h>

HWND   hwndFrame;       /* frame window handle                  */
PFNWP  OpenDlg;

case IDM_OPEN:
    if (WinDlgBox(HWND_DESKTOP,
            hwndFrame,    /* handle of the owner          */
            OpenDlg,      /* dialog procedure address     */
            NULLHANDLE,   /* location of dialog resource */
            IDD_OPEN,     /* resource identifier          */
            NULL)) {      /* application-specific data    */
        .
        . /* code executed if dialog box returns TRUE */
        .
    }
```

WinDrawBitmap

This function draws a bit map using the current image colors and mixes.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinDrawBitmap (HPS hpsDst, HBITMAP hbm, PRECTL pwrcSrc,
                    PPOINTL pptlDst, LONG clrFore, LONG clrBack,
                    ULONG fl)
```

Parameters

hpsDst (HPS) – input
Handle of presentation space in which the bit map is drawn.

hbm (HBITMAP) – input
Bit-map handle.

pwrcSrc (PRECTL) – input
Subrectangle of bit map to be drawn.

Note: The value of each field in this structure must be in the range –32 768 through 32 767. The data type WRECT can also be used, if supported by the language.

NULL The whole of the bit map is drawn
Other The whole of the bit map is not drawn.

pptlDst (PPOINTL) – input
Bit-map destination.

The bottom left corner of the bit-map destination is specified in device coordinates.

If DBM_STRETCH is set in *fl*, this parameter is used as a pointer to a RECTL data structure that contains the coordinates of the rectangle into which the source bit map is to be drawn. In this situation, *pptlDst* should be treated as a PRECTL datatype.

clrFore (LONG) – input
Foreground color.

This is used if *hbm* refers to a monochrome bit map. In this instance, bit-map bits that are set to 1 are drawn using *clrFore*. Ignored if DBM_IMAGEATTRS is specified.

clrBack (LONG) – input
Background color.

This is used if *hbm* refers to a monochrome bit map. In this instance, bit-map bits that are set to zero are drawn using *clrBack*. Ignored if DBM_IMAGEATTRS is specified.

fl (ULONG) – input
Flags that determine how the bit map is drawn.

DBM_NORMAL	Draw the bit map normally using ROP_SRCCOPY, as defined in GpiBitBlt.
DBM_INVERT	Draw the bit map inverted using ROP_NOTSRCCOPY, as defined in GpiBitBlt.
DBM_STRETCH	*pptlDst* is used to point to a RECTL data structure representing a rectangle in the destination presentation space, into which the bit map will be stretched or compressed. If compression is required, some rows and columns of the bit map are eliminated.
DBM_HALFTONE	Use the OR operator to combine the bit map with an alternating pattern of ones or zeros before drawing it. It can be used with either DBM_NORMAL or DBM_INVERT.
DBM_IMAGEATTRS	If this is specified, color conversion of monochrome bit maps is done by using the image attributes.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.
PMERR_HBITMAP_BUSY (0x2032)	An internal bit map busy error was detected. The bit map was locked by one thread during an attempt to access it from another thread.

Remarks

This function should only be used in draw mode (DM_DRAW) to a screen device context (see "GpiSetDrawingMode" in the *Graphics Programming Interface Programming Reference*). The presentation space handle can be to either a micro-presentation space or a normal presentation space (see "GpiCreatePS" in the *Graphics Programming Interface Programming Reference*).

If *hbm* refers to a color bit map, no color conversion is performed.

The current position in the presentation space is not changed by this function.

Related Functions

- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinFillRect
- WinGetSysBitmap
- WinInvertRect
- WinQueryPresParam
- WinRemovePresParam
- WinScrollWindow
- WinSetPresParam

Example Code

This example uses WinDrawBitmap to draw the system-defined menu check mark bit map in response to the user selecting a menu item (WM_MENUSELECT), using the bit-map handle returned by WinGetSysBitmap.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINPOINTERS        /* Window Pointer Functions     */
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINMENUS           /* Window Menu Functions        */

#include <os2.h>

HPS   hps;              /* presentation-space handle            */
HBITMAP hbmCheck;       /* check mark bit-map handle            */
HWND    hwndMenu;       /* menu handle                          */
USHORT  usItemId;       /* menu item id                         */
RECTL   rclItem;        /* item border rectangle                */
MPARAM  mpParam1;       /* Parameter 1 (menu item id)           */
MPARAM  mpParam2;       /* Parameter 2 (menu handle)            */

case WM_CREATE:
     /* obtain check mark bit-map handle */
     hbmCheck = WinGetSysBitmap(HWND_DESKTOP, SBMP_MENUCHECK);

case WM_MENUSELECT:
     usItemId = SHORT1FROMMP(mpParam1);
     hwndMenu = HWNDFROMMP(mpParam2);

     /* get rectangle of selected item */
     WinSendMsg(hwndMenu,
                MM_QUERYITEMRECT,
                MPFROM2SHORT(usItemId, TRUE),
                MPFROMP(&rclItem));
```

```
/* draw the check mark in the lower left corner of item's
   rectangle */
if (hbmCheck != NULL)
   {
   WinDrawBitmap(hps,
                 hbmCheck,          /* check mark           */
                 NULL,              /* draw whole bit map   */
                 (PPOINTL)&rclItem,/* bit-map destination  */
                 0L,                /* ignored since color  */
                 0L,                /* bit map */
                 DBM_NORMAL);       /* draw normal size     */
   }
```

WinDrawBorder

This function draws the borders and interior of a rectangle.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinDrawBorder  (HPS hps, PRECTL prcl, LONG cx, LONG cy,
                     LONG clrFore, LONG clrBack, ULONG flCmd)
```

Parameters

hps (HPS) – input
Presentation-space handle.

prcl (PRECTL) – input
Bounding rectangle for the border.

The rectangle is in device coordinates.

The border is drawn within the rectangle. Along the bottom and left edges of the rectangle, the edges of the border coincide with the rectangle edges. Along the top and right edges of the rectangle, the border is drawn one device unit inside the rectangle edges.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type WRECT can also be used, if supported by the language.

cx (LONG) – input
Width of border rectangle vertical sides.

This is the width of the left and right sides in device coordinates.

cy (LONG) – input
Width of border rectangle horizontal sides.

This is the width of the top and bottom sides in device coordinates.

clrFore (LONG) – input
Color of edge of border.

Not used if DB_AREAATTRS is specified.

clrBack (LONG) – input
Color of interior of border.

Not used if DB_AREAATTRS is specified.

flCmd (ULONG) – input
Flags controlling the way in which the border is drawn.

Some of the DB_* flags are mutually exclusive. Only one of these four can be significant:

- DB_PATCOPY (default)
- DB_PATINVERT
- DB_DESTINVERT
- DB_AREAMIXMODE.

Possible values are described in the following list:

DB_ROP
A group of flags that specify the mix to be used, for both the border and the interior.

DB_PATCOPY
Use the ROP_PATCOPY raster operation (see "GpiBitBlt" in the *Graphics Programming Interface Programming Reference*). This is a copy of the pattern to the destination.

DB_PATINVERT
Use the ROP_PATINVERT raster operation (see "GpiBitBlt" in the *Graphics Programming Interface Programming Reference*). This is an exclusive-OR of the pattern with the destination.

DB_DESTINVERT
Use the ROP_DESTINVERT raster operation (see "GpiBitBlt" in the *Graphics Programming Interface Programming Reference*). This inverts the destination.

DB_AREAMIXMODE
Map the current area foreground mix attribute into a Bitblt raster operation (see "GpiBitBlt" in the *Graphics Programming Interface Programming Reference*). The area background mix mode is ignored.

DB_INTERIOR
The area contained within the given rectangle, and not included within the borders (as given by *cx* and *cy*), is drawn.

DB_AREAATTRS

- If this is specified:

  For any border, the pattern used is the pattern as currently defined in the area attribute.

  For any interior, the pattern used is the same as if GpiSetAttrs for the area attributes is made with the background color of the area attribute being passed for the foreground color, and the foreground color of the area attribute being passed as the background color.

- If this is not specified (default):

  For any border, the pattern used is the same as if GpiSetAttrs for the area attributes is made with a foreground color of *clrFore*, and a background color of *clrBack*.

For any interior, the pattern used is the same as if GpiSetAttrs for the area attributes is made with a foreground color of *clrBack*, and a background color of *clrFore*.

DB_STANDARD — *cx* and *cy* are multiplied by the system SV_CXBORDER and SV_CYBORDER constants to produce the widths of the vertical and horizontal sides of the border.

DB_DLGBORDER — A standard dialog border is drawn, in the active titlebar color if DB_PATCOPY is specified, or the inactive titlebar color if DB_PATINVERT is specified. Other DB_ROP options, and DB_AREAATTRS, are ignored.

DB_ROP and DB_AREAATTRS are also ignored for the interior. The interior is drawn in the color specified by *clrBack*.

Returns

rc (BOOL) – returns
Success indicator.

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_FLAG (0x1019) — An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

PMERR_INV_DRAW_BORDER_OPTION (0x2068) — An invalid option parameter was specified with WinDrawBorder.

Remarks

A border is a rectangular frame, normally used around the edge of a window.

This function should only be used in draw mode (DM_DRAW), to a screen device context; *hps* can be either a micro-presentation space or a normal presentation space (see "GpiCreatePS" in the *Graphics Programming Interface Programming Reference*). DB_DESTINVERT inverts the destination.

If DB_AREAMIXMODE is given, the foreground mix mode from the area attribute is mapped into an equivalent ROP_ value (see "GpiBitBlt" in the *Graphics Programming Interface Programming Reference*). The area background mix mode is ignored.

Either or both *cx* or *cy* can be zero. If both are zero, the interior is still drawn. If either the x borders overlap or the y borders overlap, the border is drawn as a single rectangle with no interior.

Related Functions

- WinDrawBitmap
- WinDrawPointer
- WinDrawText
- WinFillRect
- WinGetSysBitmap
- WinInvertRect
- WinQueryPresParam
- WinRemovePresParam
- WinScrollWindow
- WinSetPresParam

Example Code

This example uses WinDrawBorder to draw the border (width of 5) and interior of a 300x200 rectangle anchored at (0,0), and using the area's current attributes for both the border and interior colors.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HPS   hps;              /* presentation-space handle            */
BOOL    fSuccess;       /* success indicator                    */
RECTL   prclRectangle={0,0,300,200}; /* border rectangle        */
LONG    lVertSideWidth=5; /* Width of border rectangle vertical
                             sides                              */
LONG    lHorizSideWidth=5;/* Width of border rectangle horizontal
                             sides                              */
ULONG   flCmd;          /* draw flags                           */

/* use current area attributes */
flCmd = DB_AREAATTRS;

fSuccess = WinDrawBorder(hps, &prclRectangle, lVertSideWidth,
                         lHorizSideWidth, 0L, 0L, flCmd);
```

WinDrawPointer

This function draws a pointer in the passed *hps* at the passed coordinates [*lx*, *ly*].

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinDrawPointer (HPS hps, LONG lx, LONG ly, HPOINTER hptrPointer,
                     ULONG ulHalftone)
```

Parameters

hps (HPS) – input
Presentation-space handle into which the pointer is drawn.

This can be either a micro presentation space or a normal presentation space (see "GpiCreatePS" in the *Graphics Programming Interface Programming Reference*).

lx (LONG) – input
x-coordinate at which to draw the pointer, in device coordinates.

ly (LONG) – input
y-coordinate at which to draw the pointer, in device coordinates.

hptrPointer (HPOINTER) – input
Pointer handle.

The pointer should be loaded using WinLoadPointer, WinCreatePointer, or WinCreatePointerIndirect.

ulHalftone (ULONG) – input
Shading control with which to draw the pointer.

DP_NORMAL	As it normally appears.
DP_HALFTONED	With a halftone pattern where black normally appears.
DP_INVERTED	Inverted, black for white and white for black.
DP_MINIICON	Bit map of a mini icon.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Function failed.

Possible returns from WinGetLastError

PMERR_INVALID_HPTR (0x101B)	An invalid pointer handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

This function should only be used in draw mode (DM_DRAW) to a screen device context.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawBitmap
- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinFillRect
- WinGetSysBitmap
- WinInvertRect
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQueryPresParam
- WinQuerySysPointer
- WinQuerySysPointerData
- WinRemovePresParam
- WinScrollWindow
- WinSetPointer
- WinSetPointerPos
- WinSetPresParam
- WinShowPointer
- WinDrawPointer
- WinSetSysPointerData

Example Code

This example draws a bit map pointer, created by either WinCreatePointer or WinCreatePointerIndirect, in response to a paint message (WM_PAINT).

```
#define INCL_WINPOINTERS        /* Window Pointer Functions     */
#define INCL_GPIBITMAPS         /* Graphics bit-map functions    */
#include <os2.h>

HPS   hps;              /* presentation-space handle            */
HWND  hwnd;             /* window handle                        */
HPOINTER  hptr;         /* bit-map pointer handle               */
HBITMAP  hbm;           /* bit-map handle                       */
BOOL  fSuccess;         /* success indicator                    */
ULONG   ulHalftone=DP_NORMAL; /* draw with normal shading       */

case WM_CREATE:
     hps = WinBeginPaint(hwnd, NULLHANDLE, NULL);
     hbm = GpiLoadBitmap(hps, 0L, IDP_BITMAP, 64L, 64L);
     WinEndPaint(hps);

     hptr = WinCreatePointer(HWND_DESKTOP, hbm,
                             TRUE, /* use true (system) pointer */
                             0, 0); /* hot spot offset (0,0)    */

case WM_PAINT:
     hps = WinBeginPaint(hwnd, NULLHANDLE, NULL);
     fSuccess = WinDrawPointer(hps, 50, 50, hptr, ulHalftone);
     WinEndPaint(hps);
```

WinDrawText

This function draws a single line of formatted text into a specified rectangle.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinDrawText (HPS hps, LONG cchText, PCH lpchText, PRECTL prcl,
                  LONG clrFore, LONG clrBack, ULONG flCmd)
```

Parameters

hps (HPS) – input
Presentation-space handle.

cchText (LONG) – input
Count of the number of characters in the string.

This parameter must be greater or equal to -1L.

-1L The string is null-terminated and its length is to be calculated by this function.
Other Count of the number of characters in the string.

lpchText (PCH) – input
Character string to be drawn.

A carriage-return or line-feed character allways terminates a line. If the length of a line is less than *cchText*(the string is divided into more than one line) each line is terminated by either of the mentioned characters.

prcl (PRECTL) – in/out
Text rectangle.

Rectangle within which the text is to be formatted, in world coordinates. Points on the boundary of this rectangle are deemed to be inside the rectangle.

On input, this rectangle is the desired rectangle in which the text is to be drawn.

On output, the height of the rectangle is modified to the actual size needed to draw the given text string. The return value is only of interest in the instance where DT_QUERYEXTENT is set in *flCmd*.

Note: The value of each field in this structure must be in the range -32768 through 32767. The data type WRECT can also be used, if supported by the language.

clrFore (LONG) – input
Foreground color.

Ignored if DT_TEXTATTRS is specified.

clrBack (LONG) – input
Background color.

The background is drawn with the current background mix. The default is BM_LEAVEALONE, that is, *clrBack* is ignored unless GpiSetBackMix is called.

The background rectangle is the rectangle that bounds the text; it is not the input parameter rectangle.

This parameter is ignored if DT_TEXTATTRS is specified.

flCmd (ULONG) – input
An array of flags that determines how the text is drawn.

Some of the DT_ flags are mutually exclusive. Only **one** from each of these groups is significant:

- DT_LEFT (default), DT_CENTER, DT_RIGHT
- DT_TOP (default), DT_VCENTER, DT_BOTTOM.

When mutually-exclusive flags are used together, the function gives indeterminate results.

If DT_HALFTONE, DT_ERASERECT, or DT_MNEMONIC is used, the presentation space must be in PU_PELS units.

DT_LEFT	Left-justify the text.
DT_CENTER	Center the text.
DT_RIGHT	Right-justify the text.
DT_VCENTER	Vertically center the text.
DT_TOP	Top-justify the text.
DT_BOTTOM	Bottom-justify the text.
DT_HALFTONE	Halftone the text display.
DT_MNEMONIC	If a mnemonic prefix character is encountered, the next character is drawn with mnemonic emphasis.
DT_QUERYEXTENT	The height *prcl* is changed to a rectangle that bounds the string if it were drawn with WinDrawText.
DT_WORDBREAK	Only words that fit completely within the supplied rectangle are drawn. A *word* is defined as: Any number of leading spaces followed by one or more visible characters and terminated by a space, carriage return, or line-feed character. When calculating whether a particular word fits within the given rectangle, this function does not consider the trailing blanks. Only the length of the visible part of the word is tested against the right edge of the rectangle.

Also, note that this function always tries to draw at least one word, even if that word does not fit in the passed rectangle. This is so that progress is always made when drawing multiline text.

DT_EXTERNALLEADING — This flag causes the "external leading" value for the current font to be added to the bottom of the bounding rectangle before returning. It has an effect only when both DT_TOP and DT_QUERYEXTENT are also specified.

DT_TEXTATTRS — If this is specified, text is drawn using the character foreground and background colors of the presentation space, and *clrFore* and *clrBack* are ignored.

DT_ERASERECT — If this is specified, the rectangle defined by *prcl* is erased before drawing the text. Otherwise, the background of the characters themselves can be erased if the character background mix (see "GpiSetAttrs" and "GpiSetBackMix" in the *Graphics Programming Interface Programming Reference*) is set to BM_OVERPAINT.

DT_UNDERSCORE — Underscore the characters. See FATTR_SEL_UNDERSCORE in the FATTRS datatype.

DT_STRIKEOUT — Overstrike the characters. See FATTR_SEL_STRIKEOUT in the FATTRS datatype.

Returns

lChars (LONG) – returns

Count of characters drawn within the rectangle.

If DT_WORDBREAK is specified, this parameter returns the number of characters displayed. However, if the first word of the string does not fit in the rectangle, this parameter reflects the fact that the entire word is drawn.

If DT_WORDBREAK is **not** specified, the count returned is the full length of the string regardless of how much fits into the bounding rectangle.

0 Error occurred
Other Count of characters drawn within the rectangle.

Possible returns from WinGetLastError

PMERR_INVALID_FLAG (0x1019) — An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

Text is always drawn in the current font with the current foreground and background mix modes.

This function must only be used in draw mode (DM_DRAW), to a screen device context; *hps* can be either a micro presentation space or a normal presentation space (see "GpiCreatePS" in the *Graphics Programming Interface Programming Reference*).

Related Functions

- WinDrawBitmap
- WinDrawBorder
- WinDrawPointer
- WinFillRect
- WinGetSysBitmap
- WinInvertRect
- WinQueryPresParam
- WinRemovePresParam
- WinScrollWindow
- WinSetPresParam

Example Code

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 is 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 *hTotalDrawn* variable to provide an offset into the string for the next call to WinDrawText.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwnd;           /* parent window                        */
RECTL   rcl;            /* update region                        */
HPS     hps;            /* presentation-space handle            */
char    *pszText;       /* string                               */
LONG    hText;         /* length of string                     */
LONG   cyCharHeight;    /* set character height                 */
LONG    hTotalDrawn;   /* total characters drawn               */
LONG    hDrawn;        /* characters drawn by WinDrawText      */
LONG    cchText;
LONG    cchTotalDrawn;
LONG    cchDrawn;
```

```
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 =  (LONG)strlen(pszText);        /* get length of string  */
cyCharHeight = 15L;                      /* set character height  */

/* until all chars drawn */
for (cchTotalDrawn = 0;  hTotalDrawn !=  hText;
                        rcl.yTop -= cyCharHeight)
   {
   /* draw the text */

    hDrawn = WinDrawText(hps,      /* presentation-space handle */
        hText -  hTotalDrawn,     /* length of text to draw    */
       pszText +  hTotalDrawn,    /* address of the text       */
       &rcl,                       /* rectangle to draw in     */
       0L,                         /* foreground color         */
       0L,                         /* background color         */
       DT_WORDBREAK | DT_TOP | DT_LEFT | DT_TEXTATTRS);
   if (cchDrawn)
        hTotalDrawn +=  hDrawn;
   else
       break;                      /* text could not be drawn   */
   }

WinReleasePS(hps);                  /* release the ps            */
```

WinEmptyClipbrd

This function empties the clipboard, removing and freeing all handles to data that is in the clipboard.

Syntax

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinEmptyClipbrd  (HAB hab)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The clipboard must be opened using WinOpenClipbrd before using this function.

This function will send a WM_DESTROYCLIPBOARD message to the clipboard owner.

Related Functions

- WinCloseClipbrd
- WinEnumClipbrdFmts
- WinOpenClipbrd
- WinQueryClipbrdData
- WinQueryClipbrdFmtInfo
- WinQueryClipbrdOwner
- WinQueryClipbrdViewer
- WinSetClipbrdData
- WinSetClipbrdOwner
- WinSetClipbrdViewer

Example Code

This example empties the clipboard (opened by WinOpenClipbrd), removing and freeing all handles to data in the clipboard.

```
#define INCL_WINCLIPBOARD       /* Window Clipboard Functions   */
#include <os2.h>

BOOL  fSuccess;          /* success indicator                   */
HAB   hab;               /* anchor-block handle                 */


fSuccess = WinOpenClipbrd(hab);

if (fSuccess)
   fSuccess = WinEmptyClipbrd(hab);
```

WinEnableControl

This macro sets the enable state of the item in the dialog template to the enable flag.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinEnableControl  (HWND hwndDlg, USHORT usId, BOOL fEnable)
```

Parameters

hwndDlg (HWND) – input
Dialog window handle.

usId (USHORT) – input
Identity of the item in the dialog template (button id).

fEnable (BOOL) – input
Enable flag.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This macro expands to:

```
#define WinEnableControl(hwndDlg, usId, fEnable)
WinEnableWindow(WinWindowFromId(hwndDlg, usId), fEnable)
```

This function requires the existence of a message queue.

Related Functions

- WinEnableWindow
- WinWindowFromID

Example Code

This example uses WinEnableControl to enable a dialog control if it is currently disabled.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwndDlg;        /* dialog window                        */
MPARAM  mp1;            /* Parameter 1                          */
USHORT  usId;           /* dialog control id                    */


if (!WinIsControlEnabled(hwndDlg, usId))
   WinEnableControl(hwndDlg, usId, TRUE);
```

WinEnableMenuItem

This macro sets the state of the specified menu item to the enable flag.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinEnableMenuItem (HWND hwndMenu, USHORT usId, BOOL fEnable)
```

Parameters

hwndMenu (HWND) – input
Menu window handle.

usId (USHORT) – input
Item identifier.

fEnable (BOOL) – input
Enable flag.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This macro expands to:

```
#define WinEnableMenuItem(hwndMenu, usId, fEnable)
((BOOL)WinSendMsg(hwndMenu,
                  MM_SETITEMATTR,
                  MPFROM2SHORT(usId, TRUE),
                  MPFROM2SHORT(MIA_DISABLED, (BOOL)(fEnable) ? 0 : MIA_DISABLED)))
```

This function requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- MM_SETITEMATTR

Example Code

This example uses WinEnableMenuItem to make a menu item selection available when the menu is initialized (WM_INITMENU).

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINMENUS           /* Window Menu Functions        */
#include <os2.h>

BOOL    fResult;        /* message-posted indicator             */
MPARAM  mpParam1;       /* Parameter 1 (rectl structure)        */
MPARAM  mpParam2:       /* Parameter 2 (frame boolean)          */
USHORT  usItemId;       /* menu item id                         */
HWND    hwndMenu;       /* menu handle                          */

case WM_INITMENU:
     usItemId = SHORT1FROMMP(mpParam1);
     hwndMenu = HWNDFROMMP(mpParam2);

     /* enable menu item */
     fResult = WinEnableMenuItem(hwndMenu, usItemId, TRUE);
```

WinEnablePhysInput

This function enables or disables queuing of physical input (keyboard or mouse).

Syntax

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinEnablePhysInput  (HWND hwndDesktop, BOOL fEnable)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

fEnable (BOOL) – input
New state for the queuing of physical input.

TRUE	Pointing device and keyboard input are queued
FALSE	Pointing device and keyboard input are disabled.

Returns

rc (BOOL) – returns
Previous state for the queuing of physical input.

TRUE	Pointing device and keyboard input were queued
FALSE	Pointing device and keyboard input were disabled.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinFocusChange
- WinGetKeyState
- WinGetPhysKeyState
- WinQueryFocus
- WinSetFocus
- WinSetKeyboardStateTable

Example Code

This example uses WinEnablePhysInput to enable queuing of physical input (pointing device and keyboard).

```
#define INCL_WININPUT           /* Window Input Functions       */
#include <os2.h>

BOOL  fOldInputState;   /* previous queuing state               */
BOOL  fNewInputState=TRUE;/* new queuing state                  */

/* enable queuing of physical input */
fOldInputState = WinEnablePhysInput(HWND_DESKTOP, fNewInputState);
```

WinEnableWindow

This function sets the window enabled state.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinEnableWindow  (HWND hwnd, BOOL fNewEnabled)
```

Parameters

hwnd (HWND) – input
Window handle.

fNewEnabled (BOOL) – input
New enabled state.

TRUE	Set window state to enabled
FALSE	Set window state to disabled.

Returns

rc (BOOL) – returns
Window enabled indicator.

TRUE	Window enabled state successfully updated
FALSE	Window enabled state not successfully updated.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

If the enable state of *hwnd* is changing, a WM_ENABLE message is sent before this function returns.

If a window is disabled, its child windows are also disabled, although they are not sent the WM_ENABLE message. Typically, a window changes appearance when disabled. For example, a disabled push button is displayed with half-tone text.

If *hwnd* is disabled, and it, or one of its descendants, is the focus window, that window loses the focus, so that no window has the focus. However, a disabled window may subsequently be assigned the focus, in which case it should respond to keyboard input.

Related Functions

- WinIsThreadActive
- WinIsWindow
- WinIsWindowEnabled
- WinQueryDesktopWindow
- WinQueryObjectWindow
- WinQueryWindowDC
- WinQueryWindowProcess
- WinQueryWindowRect
- WinWindowFromDC
- WinWindowFromID
- WinWindowFromPoint

Related Messages

- WM_ENABLE

Example Code

This example uses WinEnableWindow to enable the system menu window for the given parent window, after verifying that the parent window handle is valid (WinIsWindow), belongs to the calling thread (WinIsThreadActive), and is not presently enabled (WinIsWindowEnabled).

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINFRAMEMGR        /* Window Frame Functions       */
#include <os2.h>

HAB   hab;              /* anchor-block handle                  */
HWND    hwndSysmenu;    /* system menu window                   */
HWND    hwnd;           /* parent window                        */
BOOL  fSuccess;         /* success indicator                    */

/* if handle specifies a valid window and the window belongs to the
   current thread, query the enabled status of the system menu */
if (WinIsWindow(hab, hwnd) && WinIsThreadActive(hab))
   {
   /* obtain handle for system menu */
   hwndSysmenu = WinWindowFromID(hwnd,FID_SYSMENU);

   /* if system menu is not enabled, enable it */
   if (!WinIsWindowEnabled(hwndSysmenu))
      fSuccess = WinEnableWindow(hwndSysmenu, TRUE);
   }
```

WinEnableWindowUpdate

This function sets the window visibility state for subsequent drawing.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinEnableWindowUpdate  (HWND hwnd, BOOL fEnable)
```

Parameters

hwnd (HWND) – input
Window handle.

fEnable (BOOL) – input
New visibility state.

TRUE	Set window state visible
FALSE	Set window state invisible.

Returns

rc (BOOL) – returns
Visibility-changed indicator.

TRUE	Window visibility successfully changed
FALSE	Window visibility not successfully changed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

This function can be used to defer drawing when making a series of changes to a window. A window can be redrawn by using the WinShowWindow function.

WS_VISIBLE (style bit of a window) is set to *fEnable* without causing redrawing. That is, if the window was previously visible, it remains visible on the device when WS_VISIBLE is set to 0, and if the window was previously invisible, it is not shown when WS_VISIBLE is set to 1. If *fEnable* is set to TRUE, any subsequent drawing into the window is visible. If *fEnable* is set to FALSE, any subsequent drawing into the window is not visible.

If the value of the WS_VISIBLE style bit has been changed, the WM_SHOW message is sent to the window of *hwnd* before the call returns.

Any alteration to the appearance of a window disabled for window update is not presented. Therefore, the application must ensure that the window is redrawn. To show a window and ensure that it is redrawn after calling the WinEnableWindowUpdate function with *fEnable* set to FALSE, use the WinShowWindow function with *fNewVisibility* set to TRUE. In particular, if a window is destroyed while in this state its image is not removed from the display. After window updating is reenabled, the application should ensure that the window gets totally invalidated so that it repaints.

Related Functions

- WinBeginPaint
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Related Messages

- WM_PAINT

Example Code

This example uses WinEnableWindowUpdate to set a window's WS_VISIBLE style to visible and cause the window to be updated by a WM_PAINT message.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwnd;           /* parent window                        */
BOOL  fSuccess;         /* success indicator                    */

case WM_CREATE:
     /* if window has WS_VISIBLE off, set state to visible */
     if (!WinIsWindowVisible(hwnd))
        {
        /* set state to visible and cause WM_PAINT message */
        fSuccess = WinEnableWindowUpdate(hwnd, EWUF_ENABLE);
        }
```

WinEndEnumWindows

This function ends the enumeration process for a specified enumeration.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinEndEnumWindows  (HENUM henum)
```

Parameters

henum (HENUM) – input
Enumeration handle.

Returned by previous call to the WinBeginEnumWindows call.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HENUM (0x101C)	An invalid enumeration handle was specified.

Remarks

This function destroys the window hierarchy remembered by the WinBeginEnumWindows function. After this function, the *henum* parameter is no longer valid.

Related Functions

- WinBeginEnumWindows
- WinEnumDlgItem
- WinGetNextWindow
- WinIsChild
- WinMultWindowFromIDs
- WinQueryWindow
- WinSetOwner
- WinSetParent

Example Code

This example ends the child window enumeration and releases the enumeration handle supplied by WinBeginEnumWindows after WinGetNextWindow has enumerated all immediate children of the Desktop.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND  hwndParent;       /* Handle of the window whose child windows
                           are to be enumerated                 */
HWND  hwndNext;         /* current enumeration handle           */
HENUM  henum;           /* enumeration handle                   */
BOOL  fSuccess;         /* success indicator                    */
SHORT sRetLen;          /* returned string length               */
SHORT sLength = 10;     /* string buffer length                 */
char  pchBuffer[10];    /* string buffer                        */

hwndParent = HWND_DESKTOP;

henum = WinBeginEnumWindows(hwndParent);

while ((hwndNext = WinGetNextWindow(henum)) != NULLHANDLE) {
   .
   .
   .
}
fSuccess = WinEndEnumWindows (henum);
```

WinEndPaint

This function indicates that the redrawing of a window is complete, generally as part of the processing of a WM_PAINT message.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinEndPaint  (HPS hps)
```

Parameters

hps (HPS) – input
Presentation-space handle.

Handle of the presentation space that is used for drawing and that is returned by a previous call to the WinBeginPaint function.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The presentation space is restored to its state before the WinBeginPaint function:

- Cache presentation space is returned to the cache.
- Other presentation spaces have their original drawing state restored, including reassociating the original device context (if there was one).

If the pointer is hidden by the WinBeginPaint function, it is reshown by this function.

Any child windows having a synchronous painting style of the window associated with the presentation space are updated during the processing of this function, if they have non-NULL update regions.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS

- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Related Messages

- WM_PAINT

Example Code

This example uses WinEndPaint to end the update of a region and release the presentation space obtained by WinBeginPaint.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwnd;           /* parent window                        */
RECTL   rcl;            /* update region                        */
HPS     hps;            /* presentation-space handle            */

 case WM_PAINT:
    hps = WinBeginPaint(hwnd,   /* handle of the window      */
        NULLHANDLE,             /* get a cache presentation space */
        &rcl);                  /* receives update rectangle  */
    WinFillRect(hps, &rcl, CLR_WHITE);
    WinEndPaint(hps);
```

WinEnumClipbrdFmts

This function enumerates the list of clipboard data formats available in the clipboard.

Syntax

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinEnumClipbrdFmts  (HAB hab, ULONG fmt)
```

Parameters

hab (HAB) – input
Anchor-block handle.

fmt (ULONG) – input
Previous clipboard-data format index.

Specifies the index of the last clipboard data format enumerated using this function.

This should start at zero, in which instance the first available format is obtained. Subsequently, it should be set to the last format index value returned by this function.

Returns

ulNext (ULONG) – returns
Next clipboard-data format index.

Possible values include:

0	Enumeration is complete; that is, there are no more clipboard formats available.
Other	Index of the next available clipboard-data format in the clipboard.

Remarks

The clipboard should be open before this function is used.

Related Functions

- WinCloseClipbrd
- WinEmptyClipbrd
- WinOpenClipbrd
- WinQueryClipbrdData
- WinQueryClipbrdFmtInfo
- WinQueryClipbrdOwner
- WinQueryClipbrdViewer
- WinSetClipbrdData
- WinSetClipbrdOwner
- WinSetClipbrdViewer

Example Code

This example enumerates and counts the available clipboard data formats for the clipboard opened by WinOpenClipbrd.

```
#define INCL_WINCLIPBOARD       /* Window Clipboard Functions   */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
HAB   hab;              /* anchor-block handle                  */
ULONG   ulNext;         /* next format index                    */
ULONG   ulPrev;         /* previous format index                */
ULONG   ulNumFormats=0; /* number of available formats          */


fSuccess = WinOpenClipbrd(hab);

if (fSuccess)
   {
   ulPrev = 0;
   /* enumerate formats and maintain count */
   while ((ulNext = WinEnumClipbrdFmts(hab, ulPrev)) != 0)
      {
      ulNumFormats++;
      ulPrev = ulNext;
      }
   }
```

WinEnumDlgItem

This function returns the window handle of a dialog item within a dialog window.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinEnumDlgItem (HWND hwndDlg, HWND hwnd, ULONG code)
```

Parameters

hwndDlg (HWND) – input
Dialog-window handle.

hwnd (HWND) – input
Child-window handle.

This may be an immediate child of the dialog window or a window lower in the window hierarchy, such as a child of a child window.

NULLHANDLE can be specified if *code* is EDI_FIRSTTABITEM or EDI_LASTTABITEM.

code (ULONG) – input
Item-type code.

Determines the type of dialog item to return. This parameter can have one of the following values:

EDI_PREVTABITEM	Previous item with style WS_TABSTOP. Wraps around to end of dialog item list when beginning is reached.
EDI_NEXTTABITEM	Next item with style WS_TABSTOP. Wraps around to beginning of dialog item list when end is reached.
EDI_FIRSTTABITEM	First item in dialog with style WS_TABSTOP. *hwnd* is ignored.
EDI_LASTTABITEM	Last item in dialog with style WS_TABSTOP. *hwnd* is ignored.
EDI_PREVGROUPITEM	Previous item in the same group. Wraps around to end of group when the start of the group is reached. For information on the WS_GROUP style, see "Window Styles" in the *Presentation Manager Programming Reference Volume II*.
EDI_NEXTGROUPITEM	Next item in the same group. Wraps around to beginning of group when the end of the group is reached.
EDI_FIRSTGROUPITEM	First item in the same group.

EDI_LASTGROUPITEM Last item in the same group.

Returns

hwndItem (HWND) – returns
Item-window handle.

As dictated by *code*.

The window is always an immediate child of *hwndDlg*, even if *hwnd* is not an immediate child window.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinBeginEnumWindows
- WinEndEnumWindows
- WinGetNextWindow
- WinIsChild
- WinMultWindowFromIDs
- WinQueryWindow
- WinSetOwner
- WinSetParent

Example Code

This example uses WinEnumDlgItem to query the first dialog item for each immediate child of the specified dialog window. The immediate children are enumerated using a WinBeginEnumWindows - WinGetNextWindow - WinEndEnumWindows loop.

```
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND  hwndDlg;          /* Handle of the parent dialog window   */
HWND  hwndChild;        /* current dialog child                 */
HWND  hwndItem;         /* first dialog item                    */
HENUM  henum;           /* enumeration handle                   */
BOOL  fSuccess;         /* success indicator                    */

henum = WinBeginEnumWindows(hwndDlg);

while ((hwndChild = WinGetNextWindow(henum)) != NULL)
   hwndItem = WinEnumDlgItem(hwndDlg, hwndChild, EDI_FIRSTTABITEM);

fSuccess = WinEndEnumWindows (henum);
```

WinEnumObjectClasses

The WinEnumObjectClasses function will return a list of all workplace object classes that have been registered.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinEnumObjectClasses (POBJCLASS pObjClass, PULONG pSize)
```

Parameters

pObjClass (POBJCLASS) – input
Pointer to object class.

A pointer to a buffer to be filled with information about the registered workplace object classes.

pSize (PULONG) – in/out
Length of the *pObjClass* buffer in bytes.

If *pObjClass* is NULL, the actual size of *pObjClass* is returned in *pSize*

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

WinEnumObjectClasses will return a buffer containing all workplace object classes that are currently registered with the system. Workplace object classes are registered with the system through the function call WinRegisterObjectClass.

Related Functions

- WinRegisterObjectClass
- WinReplaceObjectClass

WinEqualRect

This function compares two rectangles for equality.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinEqualRect (HAB hab, PRECTL prcl1, PRECTL prcl2)
```

Parameters

hab (HAB) – input
Anchor-block handle.

prcl1 (PRECTL) – input
First rectangle.

Note: The value of each field in this structure must be in the range -32768 through 32767. The data type WRECT can also be used, if supported by the language.

prcl2 (PRECTL) – input
Second rectangle.

Note: The value of each field in this structure must be in the range -32768 through 32767. The data type WRECT can also be used, if supported by the language.

Returns

rc (BOOL) – returns
Equality indicator.

TRUE	Rectangles are identical
FALSE	Rectangles are not identical, or an error occurred.

Remarks

If both rectangles are empty (for example, *yTop* is equal to *yBottom* or *xRight* is equal to *xLeft*), they are considered equal even if the actual coordinate values are different.

Related Functions

- WinCopyRect
- WinFillRect
- WinInflateRect
- WinIntersectRect
- WinIsRectEmpty
- WinOffsetRect
- WinPtInRect

- WinSetRect
- WinSetRectEmpty
- WinSubtractRect
- WinUnionRect

Example Code

This example compares two rectangles for equality.

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions   */
#include <os2.h>

BOOL  fEqual;           /* equal indicator                      */
HAB     hab;            /* anchor-block handle                  */
RECTL prclRect1 = {0,0,100,100}; /* first rectangle             */
RECTL prclRect2 = {0,0,200,200}; /* second rectangle            */

fEqual = WinEqualRect(hab, &prclRect1, &prclRect2);
```

WinExcludeUpdateRegion

This function subtracts the update region (invalid region) of a window from the clipping region of a presentation space.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinExcludeUpdateRegion  (HPS hps, HWND hwnd)
```

Parameters

hps (HPS) – input
Presentation-space handle whose clipping region is to be updated.

hwnd (HWND) – input
Window handle.

Handle of window whose update region is subtracted from the clipping region of the presentation space.

Returns

lComplexity (LONG) – returns
Complexity value.

This indicates the resulting form of the clipping area. The values and meanings of this parameter are defined in GpiCombineRegion.

Complexity of resulting region/error indicator:

RGN_NULL	Null Region
RGN_RECT	Rectangle region
RGN_COMPLEX	Complex region
RGN_ERROR	Error.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function is typically used to prevent drawing into parts of a window that are known to be invalid, as during an incremental update optimization process.

It is the application's responsibility to reset the clipping region when necessary.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Example Code

This example uses WinExcludeUpdateRegion to prevent drawing into the window's known invalid regions (to optimize updates) by excluding the window's update region from the clipping region of the presentation space. The clipping region will need to be reset later by the application, which can accomplished using GpiIntersectClipRectangle with the rectangle comprising the window as input.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_GPIREGIONS         /* Region functions             */
#include <os2.h>

LONG  lComplexity;      /* clipping complexity/error return     */
HWND    hwnd;           /* parent window                        */
RECTL  rcl;             /* update region                        */
HPS    hps;             /* presentation-space handle            */

 case WM_PAINT:
    lComplexity = WinExcludeUpdateRegion(hps, hwnd);

    hps = WinBeginPaint(hwnd,   /* handle of the window          */
        NULLHANDLE,             /* get a cache presentation space */
        &rcl);                  /* receives update rectangle  */
    WinFillRect(hps, &rcl, CLR_WHITE);
    WinEndPaint(hps);
```

WinFileDlg

This function creates and displays the file dialog and returns the user's selection or selections.

Syntax

```
#define INCL_winstdfile
#include <os2.h>
HWND WinFileDlg (HWND hwndP, HWND hwndO, PFILEDLG pfild)
```

Parameters

hwndP (HWND) – input
Parent-window handle.

Parent-window handle of the created dialog window.

HWND_DESKTOP	The desktop window.
Other	Specified window.

hwndO (HWND) – input
Requested owner-window handle.

Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the WinLoadDlg function.

pfild (PFILEDLG) – input
Pointer to a FILEDLG structure.

Returns

hwndDlg (HWND) – returns
File dialog window handle.

If the FDS_MODELESS flag is set by the application, the return value is the window handle of the file dialog, or NULLHANDLE if the dialog cannot be created. If the FDS_MODELESS flag is not set, the return value is TRUE if dialog creation is successful, or NULLHANDLE if it is unsuccessful.

Remarks

The *pfild* parameter is required and the FILEDLG structure must be properly initialized.

On return, the FILEDLG structure is updated with any user alterations, and the *lReturn* field is set to the value returned by the file dialog's WinDismissDlg function. By default, this is the

ID of the push button pressed to dismiss the dialog, DID_OK or DID_CANCEL, unless the application supplied additional push buttons in its template.

For convenience, the pointer to the FILEDLG structure is placed in the QWL_USER field of the dialog's frame window. If in a custom file dialog procedure the pointer to the FILEDLG structure is desired, it should be queried from the frame window with the WinQueryWindowULong function.

To subclass the default file dialog with a new template, the application must give the module and ID of the new file dialog template and the address of a dialog procedure for message handling. Window IDs in the range 0x0000 through 0x0FFF are reserved for the standard file dialog controls. IDs from outside this range must be chosen for any controls or windows added to a custom file dialog.

When a modeless dialog is dismissed, the owner of the file dialog will receive a WM_COMMAND message with the *ussource* parameter equal to CMDSRC_FILEDLG and the *uscmd* parameter equal to the ID of the file dialog.

Example Code

This example uses WinFileDlg to create and display a single file selection dialog using the system default open file dialog template and procedure.

```
#define INCL_WINSTDFILE  /* Window Standard File Functions       */
#include <os2.h>

FILEDLG fild;            /* File dialog info structure           */
char pszTitle[10] = "Open File"; /* Title of dialog              */
char pszFullFile[CCHMAXPATH] = "*.C";/* File filter string       */
HWND hwndMain;           /* Window that owns the file dialog     */
HWND hwndDlg;            /* File dialog window                   */

/********************************************************************/
/* Initially set all fields to 0                                    */
/********************************************************************/

memset(&pfdFiledlg, 0, sizeof(FILEDLG));

/********************************************************************/
/* Initialize those fields in the FILEDLG structure that are        */
/* used by the application                                          */
/********************************************************************/
fild.cbSize = sizeof(FILEDLG);       /* Size of structure           */
fild.fl = FDS_HELPBUTTON | FDS_CENTER | FDS_OPEN_DIALOG      ;
                                /* FDS_* flags                 */
fild.pszTitle = pszTitle;         /* Dialog title string        */
strcpy(fild.szFullFile, pszFullFile);       /* Initial path,    */
                                            /* file name, or    */
                                            /* file filter      */

/********************************************************************/
/* Display the dialog and get the file                              */
/********************************************************************/

hwndDlg = WinFileDlg(HWND_DESKTOP, hwndMain, &fild);

if (hwndDlg && (pfild.lReturn == DID_OK))
{
   /*****************************************************************/
   /* Upon successful return of a file, open it for reading and     */
   /* further processing                                            */
   /*****************************************************************/
}
```

WinFillRect

This function draws a filled rectangular area.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinFillRect  (HPS hps, PRECTL prcl, LONG lColor)
```

Parameters

hps (HPS) – input
Presentation-space handle.

This can be either a micro-presentation space or a normal presentation space.

prcl (PRECTL) – input
Rectangle to be filled, in window coordinates.

Points on the left and bottom boundaries of the rectangle are included in the fill, but points on the right and top boundaries are not, except where they are also on the left and bottom boundaries; that is, the top-left and bottom-right corners.

Note: The value of each field in this structure must be in the range -32768 through 32767. The data type WRECT can also be used, if supported by the language.

lColor (LONG) – input
Color with which to fill the rectangle.

This is either a color index, or an RGB color value, depending upon whether and how a logical color table has been loaded. (See "GpiCreateLogColorTable" in the *Graphics Programming Interface Programming Reference*.)

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function does not change any presentation space state.

This function must only be used in draw mode (DM_DRAW) to a screen device context.

If an empty rectangle is specified, this function draws nothing and completes successfully (that is, TRUE is returned).

Related Functions

- WinCopyRect
- WinDrawBitmap
- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinEqualRect
- WinGetSysBitmap
- WinInflateRect
- WinIntersectRect
- WinIsRectEmpty
- WinInvertRect
- WinOffsetRect
- WinPtInRect
- WinQueryPresParam
- WinRemovePresParam
- WinScrollWindow
- WinSetPresParam
- WinSetRect
- WinSetRectEmpty
- WinSubtractRect
- WinUnionRect

Example Code

This example fills an update rectangle with a white background in response to the WM_PAINT message, after obtaining a presentation space handle via WinBeginPaint.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
HAB     hab;            /* anchor-block handle                  */
PRECTL prclRect1 = {0,0,100,100}; /* fill rectangle              */
LONG  lColor=CLR_WHITE; /* fill color                           */
HWND   hwnd;            /* client window handle                 */
HPS    hps;             /* presentation-space handle            */

case WM_PAINT:
     hps = WinBeginPaint(hwnd, NULLHANDLE, &prclRect1);
     fSuccess = WinFillRect(hps, &prclRect1, lColor);
     WinEndPaint(hps);
```

WinFindAtom

This function finds an atom in the atom table.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ATOM WinFindAtom  (HATOMTBL hatomtblAtomTbl, PSZ pszAtomName)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

This is the handle returned from a previous WinCreateAtomTable or WinQuerySystemAtomTable function.

pszAtomName (PSZ) – input
Atom name.

This is a null terminated character string to be found in the table.

If the string begins with a "#" character, the five ASCII digits that follow are converted into an integer atom.

If the string begins with a "!" character, the next two bytes are interpreted as an atom.

If the high order word of the string is -1, the low order word is an atom.

Returns

atom (ATOM) – returns
Atom value.

Atom	The atom associated with the passed string
0	Invalid atom table handle or invalid atom name specified.

Possible returns from WinGetLastError

PMERR_INVALID_HATOMTBL (0x1013)	An invalid atom-table handle was specified.
PMERR_INVALID_INTEGER_ATOM (0x1016)	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME (0x1015)	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND (0x1017)	The specified atom name is not in the atom table.

Remarks

This function is identical to the WinAddAtom function, except that:

- If the atom name is not found in the table, it is not added to the table and 0 is returned.
- If the atom name is found in the table, the use count is not incremented.

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.

Related Functions

- WinAddAtom
- WinCreateAtomTable
- WinDeleteAtom
- WinDestroyAtomTable
- WinQueryAtomLength
- WinQueryAtomName
- WinQueryAtomUsage
- WinQuerySystemAtomTable

Example Code

This example queries an Atom Table for the atom name of a newly created atom "newatom" and then verifies that the atom value returned by the query matches the atom value returned by WinAddAtom.

```
#define INCL_WINATOM           /* Window Atom Functions        */
#include <os2.h>

ATOM  atom;             /* new atom value                      */
ATOM  atomFound;        /* atom value from WinFindAtom         */
HATOMTBL  hatomtblAtomTbl; /* atom-table handle                */
char   pszAtomName[10]; /* atom name                           */
ULONG   ulInitial = 0;  /* initial atom table size (use default)*/
ULONG   ulBuckets = 0;  /* size of hash table (use default)    */
BOOL atomMatch = FALSE; /* indicates atom values match         */

/* create atom table of default size */
hatomtblAtomTbl = WinCreateAtomTable(ulInitial, ulBuckets);

/* define name for new atom and add to table */
strcpy(pszAtomName,"newatom");
atom = WinAddAtom(hatomtblAtomTbl, pszAtomName);

atomFound = WinFindAtom(hatomtblAtomTbl, pszAtomName);

/* verify that the atom values match */
if (atom == atomFound)
   atomMatch = TRUE;
```

WinFlashWindow

This function starts or stops a window flashing.

Syntax

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinFlashWindow  (HWND hwndFrame, BOOL fFlash)
```

Parameters

hwndFrame (HWND) – input
Handle of window to be flashed.

fFlash (BOOL) – input
Start-flashing indicator.

TRUE	Start window flashing
FALSE	Stop window flashing.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

Flashing a window brings the user's attention to a window that is not the active window, where some important message or dialog must be seen by the user.

Flashing is typically done by inverting the title bar continuously. The alarm is sounded for the first five flashes.

Note: It should be used only for important messages, for example, where some component of the system is failing and requires immediate attention to avoid damage.

Related Functions

- WinAlarm
- WinMessageBox

Example Code

This example uses WinFlashWindow to flash an inactive window to draw the user's attention to an important message in the window.

```
#define INCL_WINFRAMEMGR        /* Window Frame Functions       */
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#include <os2.h>

BOOL  fSuccess;         /* Success indicator                    */
HWND  hwnd;             /* window handle                        */

/* flash window to get user's attention */
fSuccess = WinFlashWindow(hwnd, TRUE);

/* vital message is displayed */
WinMessageBox(HWND_DESKTOP,
    hwnd,                       /* client-window handle   */
    "Important message: must be seen by user",/* message */
    "Vital message",            /* title of the message   */
    0,                          /* message box id         */
    MB_NOICON | MB_OK);         /* icon and button flags  */
```

WinFocusChange

This function changes the focus window.

Syntax

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinFocusChange (HWND hwndDeskTop, HWND hwndNewFocus,
                     ULONG flFocusChange)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

hwndNewFocus (HWND) – input
Window handle to receive the focus.

flFocusChange (ULONG) – input
Focus changing indicators.

These indicators are passed on in the WM_FOCUSCHANGE message:

FC_NOSETFOCUS	Do not send the WM_SETFOCUS message to the window receiving the focus.
FC_NOLOSEFOCUS	Do not send the WM_SETFOCUS message to the window losing the focus.
FC_NOSETACTIVE	Do not send the WM_ACTIVATE message to the window being activated.
FC_NOLOSEACTIVE	Do not send the WM_ACTIVATE message to the window being deactivated.
FC_NOSETSELECTION	Do not send the WM_SETSELECTION message to the window being selected.
FC_NOLOSESELECTION	Do not send the WM_SETSELECTION message to the window being deselected.
FC_NOBRINGTOTOP	Do not bring any window to the top.
FC_NOBRINGTOTOPFIRSTWINDOW	Do not bring the first frame window to the top.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function sends a WM_FOCUSCHANGE message to the window that is losing the focus and a WM_FOCUSCHANGE message to the window that is receiving the focus.

This function fails if another process or thread is currently using this function.

Other messages may be sent as a consequence of the frame control processing of the WM_FOCUSCHANGE (in Frame Controls) message, depending on the value of the *flFocusChange* parameter. These messages, if sent, are sent in this order:

1. WM_SETFOCUS to the window losing the focus.
2. WM_SETSELECTION to the windows losing their selection.
3. WM_ACTIVATE to the windows being deactivated.
4. WM_ACTIVATE to the windows being activated.
5. WM_SETSELECTION to the windows being selected.
6. WM_SETFOCUS to the window receiving the focus.

Note: If the WinQueryFocus function is used during processing of this function:

- The window handle of the window losing the focus is returned while the WM_FOCUSCHANGE message with the *usSetFocus* parameter set to FALSE is being processed.
- The window handle of the window receiving the focus is returned while the WM_FOCUSCHANGE (in Frame Controls) message with the *usSetFocus* parameter set to TRUE is being processed.

If the WinQueryActiveWindow function is used during processing of this function:

- The window handle of the window being deactivated is returned while the WM_ACTIVATE message with the *usactive* parameter set to FALSE is being processed.
- The window handle of the window being activated is returned while the WM_ACTIVATE message with the *usactive* parameter set to TRUE is being processed.

Also, there is a short period during the time after the old active window has acted on the deactivation message and before the new active window has acted on the

activation message when the WinQueryActiveWindow function returns NULLHANDLE.

This function should not be made unless it is directly or indirectly the result of operator input.

Even if FC_NOSETSELECTION is not specified, the WM_SETSELECTION is not sent to a frame window that is already selected. This can occur if the focus is being transferred from a parent to a child window and FC_NOLOSESELECTION was specified.

Related Functions

- WinEnablePhysInput
- WinGetKeyState
- WinGetPhysKeyState
- WinQueryFocus
- WinSetFocus
- WinSetKeyboardStateTable

Related Messages

- WM_ACTIVATE
- WM_FOCUSCHANGE
- WM_SETFOCUS
- WM_SETSELECTION

Example Code

This example uses WinFocusChange to change the focus to the selected window, using the handle returned by WinQueryFocus.

```
#define INCL_WININPUT           /* Window Input Functions       */
#include <os2.h>

HWND  hwndNewFocus;     /* Handle of new focus window           */
BOOL  fSuccess;         /* success indicator                    */
MPARAM  mpParam1;       /* Parameter 1 (select boolean)         */

case WM_SETSELECTION:
     /* if window is being selected, change focus to the window */
     if (SHORTFROMMP(mpParam1))
        {
        if ((hwndNewFocus =
             WinQueryFocus(HWND_DESKTOP)) != 0L)
           fSuccess = WinFocusChange(HWND_DESKTOP, hwndNewFocus,
                                     0L);
        }
```

WinFontDlg

This dialog allows the user to select a font.

Syntax

```
#define INCL_winstdfont
#include <os2.h>
HWND WinFontDlg (HWND hwndP, HWND hwndO, PFONTDLG pfntd)
```

Parameters

hwndP (HWND) – input
Parent-window handle.

Parent-window handle of the created dialog window.

HWND_DESKTOP	The desktop window.
Other	Specified window.

hwndO (HWND) – input
Requested owner-window handle.

Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified in the description of the WinLoadDlg function.

pfntd (PFONTDLG) – input
Pointer to an initialized FONTDLG structure.

Returns

hwnd (HWND) – returns
Font dialog window handle.

If the FNTS_MODELESS flag is set by the application, the return value is the window handle of the font dialog, or NULLHANDLE if the dialog cannot be created. If the FNTS_MODELESS flag is not set, the return value is TRUE if dialog creation is successful, or NULLHANDLE if it is unsuccessful.

Remarks

The *pfntd* parameter is required and the FONTDLG structure must be properly initialized.

Note: If the FNTS_MODELESS flag is specified in the *fl* field in the FONTDLG structure, *pfntd* must be a pointer to a FONTDLG structure that is either static or allocated from the heap. This FONTDLG structure must not be allocated on the stack.

Upon return, the FONTDLG structure is updated with any user alterations and the *lReturn* field contains the value returned by the font dialog's WinDismissDlg function. By default this is the ID of the push button pressed to dismiss the dialog, DID_OK or DID_CANCEL, unless the application supplied additional push buttons in its template.

The pointer to the FONTDLG structure is placed in the QWL_USER field of the dialog's frame window. If in a custom font dialog procedure the pointer to the FONTDLG structure is desired, it should be queried from the frame window with WinQueryWindowULong.

To subclass the default font dialog with a new template, the application must give the module and ID of the new font dialog template and the address of a dialog procedure for message handling. Window IDs in the range 0x0000 through 0x0FFF are reserved for the font dialog controls. IDs from outside this range must be chosen for any controls added to a custom font dialog.

When a modeless dialog is dismissed, the owner of the font dialog will receive a WM_COMMAND message with the *ussource* parameter equal to CMDSRC_FONTDLG and the *uscmd* parameter equal to the ID of the font dialog.

Example Code

This example displays a font selection dialog by using WinFontDlg, which allows the user to select a font.

```
#define INCL_WINSTDFONT  /* Window Standard Font Functions     */
#include <os2.h>
#include <string.h>

HPS     hpsScreen;       /* Screen presentation space          */
FONTDLG pfdFontdlg;      /* Font dialog info structure         */
HWND    hwndMain;        /* Window that owns the font dialog   */
HWND    hwndFontDlg;     /* Font dialog window                 */

char szFamilyname[FACESIZE];

/*************************************************************/
/* Initially set all fields to 0                             */
/*************************************************************/
memset(&pfdFontdlg, 0, sizeof(FONTDLG));

/*************************************************************/
/* Initialize those fields in the FONTDLG structure that are */
/* used by the application                                   */
/*************************************************************/
pfdFontdlg.cbSize = sizeof(FONTDLG);  /* Size of structure     */
pfdFontdlg.hpsScreen = hpsScreen;     /* Screen presentation   */
                                      /* space                 */
szFamilyname[0] = 0;        /* Use default font      */
pfdFontdlg.pszFamilyname  = szFamilyname; /* Provide buffer    */
pfdFontdlg.usFamilyBufLen = sizeof(szFamilyname); /* for font  */
                                      /* family name           */
pfdFontdlg.fxPointSize = MAKEFIXED(10,0); /* Font point size   */
pfdFontdlg.fl = FNTS_HELPBUTTON | FNTS_CENTER; /* FNTS_* flags */
pfdFontdlg.clrFore = CLR_BLACK;       /* Foreground color      */
pfdFontdlg.clrBack = CLR_WHITE;       /* Background color      */
pfdFontdlg.fAttrs.usCodePage = 437;   /* Code page to select   */
                                      /* from                  */

/*************************************************************/
/* Display the font dialog and get the font                  */
/*************************************************************/
hwndFontDlg = WinFontDlg(HWND_DESKTOP, hwndMain, &pfdFontdlg);

if (hwndFontDlg && (pfdFontdlg.lReturn == DID_OK))
{

   /*********************************************************/
   /* Upon successful return of a font, the application can */
   /* use font information selected by the user to create a */
   /* font, load a font, and so forth                       */
   /*********************************************************/

}
```

WinFreeErrorInfo

This function releases memory allocated for an error-information block.

Syntax

```
#define INCL_WINERRORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinFreeErrorInfo  (PERRINFO perriErrorInfo)
```

Parameters

perriErrorInfo (PERRINFO) – input
Error-information block whose memory is to be released.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	*perriErrorInfo* is not an error-information block for the current thread.

Related Functions

- WinGetErrorInfo
- WinGetLastError

Example Code

This example frees memory allocated (by WinGetErrorInfo) for an error-information block using WinFreeErrorInfo.

```
#define INCL_WINERRORS          /* Window Error Functions       */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
ERRORID  erridErrorCode;/* last error id code                   */
PERRINFO  perriErrorInfo;/* error info structure                */
HAB   hab;              /* anchor-block handle                  */

/* obtain error block and assign error code */
perriErrorInfo = WinGetErrorInfo(hab);
erridErrorCode = perriErrorInfo->idError;

/* free error block */
fSuccess = WinFreeErrorInfo(perriErrorInfo);
```

WinFreeFileDlgList

This function frees the storage allocated by the file dialog when the FDS_MULTIPLESEL dialog flag is set.

Syntax

```
#define INCL_winstdfile
#include <os2.h>
BOOL WinFreeFileDlgList (PAPSZ papszFQFilename)
```

Parameters

papszFQFilename (PAPSZ) – input
Pointer to a table of pointers of fully-qualified file names returned by the dialog.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

When the FDS_MULTIPLESEL style flag is set and the user selects one or more files from the file name list box, the fully-qualified file names of the selected files are returned in the *papszFQFilename* field of the FILEDLG structure. After the application retrieves all of the information it needs from the *papszFQFilename* array, it should call WinFreeFileDlgList to free the storage.

Example Code

This example uses the WinFreeFileDlgList function to deallocate the table of file name pointers returned by the WinFileDlg function when the FDS_MULTIPLESEL flag is set in the *fl* field of the FILEDLG structure.

```
#define INCL_WINSTDFILE  /* Window Standard File Functions       */
#include <os2.h>

BOOL     fSuccess;       /* Success indicator                    */
FILEDLG pfdFiledlg;      /* File dialog info structure           */
HWND     hwndMain;       /* Window that owns the file dialog     */
HWND     hwndDlg;        /* File dialog window                   */

/********************************************************************/
/* initialize FILEDLG structure                                     */
/********************************************************************/
pfdFiledlg.cbSize = sizeof(FILEDLG); /* Size of structure           */
pfdFiledlg.fl = FDS_MULTIPLESEL | FDS_HELPBUTTON | FDS_CENTER |
                FDS_OPEN_DIALOG;     /* FDS_* flags                 */

/********************************************************************/
/* Set remaining fields here                                        */
/********************************************************************/
.
.
.

/********************************************************************/
/* Display the dialog and get the files                             */
/********************************************************************/

hwndDlg = WinFileDlg(HWND_DESKTOP, hwndMain, &pfdFiledlg);

if (hwndDlg && (pfdFiledlg.lReturn == DID_OK))
{

   /*****************************************************************/
   /* Upon successful return of the files, open them for further */
   /* processing using the table of file name pointers              */
   /*****************************************************************/

   /*****************************************************************/
   /* Find out whether the pointer array was allocated              */
   /*****************************************************************/

   if (pfdFiledlg.papszFQFilename)

      /**************************************************************/
      /* If so, free the table of file name pointers                */
      /**************************************************************/

      fSuccess = WinFreeFileDlgList(pfdFiledlg.papszFQFilename);
}
```

WinFreeFileIcon

This function frees an icon pointer that was originally allocated by WinLoadFileIcon.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinFreeFileIcon  (HPOINTER hptr)
```

Parameters

hptr (HPOINTER) – input
A pointer to an icon loaded by WinLoadFileIcon.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Related Functions

- WinSetFileIcon
- WinLoadFileIcon

WinGetClipPS

This function obtains a clipped cache presentation space.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
HPS WinGetClipPS  (HWND hwnd, HWND hwndClipWindow, ULONG ulClipflags)
```

Parameters

hwnd (HWND) – input
Handle of window for which the presentation space is required.

hwndClipWindow (HWND) – input
Handle of window for clipping.

Values to be specified can be one of the following:

HWND_BOTTOM	Clip the last window in the sibling chain and continue clipping until the next window is *hwnd* or NULLHANDLE.
HWND_TOP	Clip the first window in the sibling chain and continue clipping until the next window is *hwnd* or NULLHANDLE.
NULLHANDLE	Clip all siblings to the window *hwnd*.

ulClipflags (ULONG) – input
Clipping control flags.

PSF_CLIPSIBLINGS	Clip out all siblings of *hwnd*.
PSF_CLIPCHILDREN	Clip out all children of *hwnd*.
PSF_CLIPUPWARDS	Taking *hwndClipWindow* as a reference window, clip out all sibling windows before *hwndClipWindow*. This value may not be used with PSF_CLIPDOWNWARDS.
PSF_CLIPDOWNWARDS	Taking *hwndClipWindow* as a reference window, clip out all sibling windows after *hwndClipWindow*. This value may not be used with PSF_CLIPUPWARDS.
PSF_LOCKWINDOWUPDATE	Calculate a presentation space that keeps a visible region even though output may be locked by the WinLockWindowUpdate function.
PSF_PARENTCLIP	Calculate a presentation space that uses the visible region of the parent of *hwnd* but with an origin calculated for *hwnd*.

Returns

hps (HPS) – returns
Presentation-space handle that can be used for drawing.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The presentation space obtained by this function is a cache "micro-presentation space" present in the system. This can be used for simple drawing operations that do not depend on long-term data being stored in the presentation space.

When the application finishes using the clipped cache presentation space, it should destroyed it using the WinReleasePS function.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Example Code

This example responds to an application defined message (IDM_FILL) and uses WinGetClipPS to obtain and associate a cached presentation space with a window, where the PS is clipped to the children of the window.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwnd;           /* PS window                            */
HWND    hwndClip;       /* clipping window                      */
RECTL   rcl;            /* update region                        */
HPS     hps;            /* presentation-space handle            */

case IDM_FILL:
     hps = WinGetClipPS(hwnd,   /* handle of the PS window */
         hwndClip,               /* handle of clipping window */
         PSF_CLIPCHILDREN);      /* clipping flags */
     WinFillRect(hps, &rcl, CLR_WHITE);
     WinReleasePS(hps);
```

WinGetCurrentTime

This function returns the current time.

Syntax

```
#define INCL_WINTIMER /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinGetCurrentTime  (HAB  hab)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

ulTime (ULONG) – returns
System-timer count.

The time is in milliseconds, from the system Initial Program Load (IPL). This is the same value as stored in the information segment.

Related Functions

- WinQueryMsgTime
- WinStartTimer
- WinStopTimer

Example Code

This example uses WinGetCurrentTime to return the current time.

```
#define INCL_WINTIMER          /* Window Timer Functions        */
#include <os2.h>

HAB  hab;              /* anchor-block handle                   */
ULONG  ulTime;         /* current time                          */

ulTime = WinGetCurrentTime(hab);
```

WinGetDlgMsg

This function obtains a message from the application's queue associated with the specified dialog.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinGetDlgMsg  (HWND hwndDlg, PQMSG pqmsg)
```

Parameters

hwndDlg (HWND) – input
Dialog-window handle.

pqmsg (PQMSG) – output
Message structure.

Returns

rc (BOOL) – returns
Continue message indicator.

TRUE Message returned is not a WM_QUIT message and the dialog has not been dismissed.

FALSE Message returned is a WM_QUIT message or the dialog has been dismissed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001) An invalid window handle was specified.

Remarks

This function enables a language that cannot support window procedures to provide the function of a modal dialog. The application creates a modeless dialog by the use of the WinCreateDlg or the WinLoadDlg functions and then issues this call to process messages only associated with the dialog.

The first time that this function is issued, the owner of the window specified by *hwndDlg* is disabled, thereby preventing input into windows other than the dialog. The owner of the window specified by *hwndDlg* is enabled when the WinDismissDlg function is issued either by the application or by the default dialog procedure.

If a WM_QUIT is encountered, WinGetDlgMsg itself issues a WinDismissDlg function, and posts the WM_QUIT message back to the queue so that the application main loop terminates in the normal way.

Related Functions

- WinGetDlgMsg
- WinBroadcastMsg
- WinCreateDlg
- WinCreateMsgQueue
- WinDefDlgProc
- WinDestroyMsgQueue
- WinDismissDlg
- WinDispatchMsg
- WinDlgBox
- WinGetMsg
- WinInSendMsg
- WinGetDlgMsg
- WinLoadDlg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinProcessDlg
- WinGetDlgMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Related Messages

- WM_QUIT

Example Code

This example uses WinGetDlgMsg to provide a modal dialog. When the user causes an open message (application defined IDM_OPEN), the dialog is loaded and displayed; WinGetDlgMsg then loops, grabbing messages from the queue and calling MyDlgRoutine—the dialog procedure which processes the messages—with the appropriate parameters. When the dialog issues a WM_QUIT, WinGetDlgMsg returns FALSE and the loop ends, returning control to owner window.

```
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#include <os2.h>

HWND   hwnd;             /* owner window                      */
HWND   hwndDlg;          /* dialog window                     */
PQMSG  pqmsgmsg;         /* message                           */

case IDM_OPEN:
     hwndDlg = WinLoadDlg(HWND_DESKTOP,  /* parent is desk top */
                          hwnd,         /* owner window handle */
                          NULL,          /* modeless dialog */
                          0L,            /* load from .EXE */
                          DLG_ID,        /* dialog resource id */
                          NULL);       /* no dialog parameters */

     /* loop and process dialog messages until WM_QUIT, calling
        dialog procedure for each message */
     while (WinGetDlgMsg(hwndDlg, &qmsg))
        MyDlgRoutine(hwndDlg, qmsg.msg, qmsg.mp1, qmsg.mp2);
     break;



MRESULT MyDlgRoutine(HWND  hwndDlg, ULONG usMsgid, MPARAM mp1,
                     MPARAM mp2)
{
switch(usMsgid)
   {
   /*
   .
   . process messages
   .
   */

   default:
      return (WinDefDlgProc(hwndDlg, usMsgid, mp1, mp2));
   }
}
```

WinGetErrorInfo

This function returns detailed error information.

Syntax

```
#define INCL_WINERRORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
PERRINFO WinGetErrorInfo  (HAB hab)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

perriErrorInfo (PERRINFO) – returns
Error information.

This structure contains information about the previous error code for the current thread:

NULL No error information available
Other Error information.

Remarks

This function allocates a single private segment to contain the ERRINFO structure. All the pointers to string fields within the ERRINFO structure are offsets to memory within that segment.

The memory allocated by this function is not released until the returned pointer is passed to the WinFreeErrorInfo function.

Related Functions

- WinFreeErrorInfo
- WinGetLastError

Example Code

This example uses WinGetErrorInfo to obtain detailed error information, assigns the error code, and frees the error block with WinFreeErrorInfo.

```
#define INCL_WINERRORS          /* Window Error Functions      */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                   */
ERRORID  erridErrorCode;/* last error id code                  */
PERRINFO  perriErrorInfo;/* error info structure               */
HAB   hab;              /* anchor-block handle                 */

/* obtain error block */
perriErrorInfo = WinGetErrorInfo(hab);
erridErrorCode = perriErrorInfo->idError;

/* free error block */
fSuccess = WinFreeErrorInfo(perriErrorInfo);
```

WinGetKeyState

This function returns the state of the key at the time that the last message obtained from the queue was posted.

Syntax

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinGetKeyState  (HWND hwndDeskTop, LONG vkey)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

vkey (LONG) – input
Virtual key value.

Contains the virtual key value in the low-order byte, and zero in the high-order byte. See remarks for possible values.

Returns

lKeyState (LONG) – returns
Key state.

This value is the OR combination of the following bits:

0x0001	The key has been pressed an odd number of times since the system has been started.
0x8000	The key is down.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

See also the WinGetPhysKeyState function. 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 pointing device buttons with the VK_BUTTON1, VK_BUTTON2, and VK_BUTTON3 virtual key codes. The following are the possible values for the *vkey* parameter:

VK_BUTTON1	VK_PRINTSCRN	VK_F18
VK_BUTTON2	VK_INSERT	VK_F19
VK_BUTTON3	VK_DELETE	VK_F20
VK_BREAK	VK_SCRLLOCK	VK_F21
VK_BACKSPACE	VK_NUMLOCK	VK_F22
VK_TAB	VK_ENTER	VK_F23
VK_BACKTAB	VK_SYSRO	VK_F24
VK_NEWLINE	VK_F1	VK_ENDDRAG
VK_SHIFT	VK_F2	VK_CLEAR
VK_CTRL	VK_F3	VK_EREOF
VK_ALT	VK_F4	VK_PA1
VK_ALTGRAF	VK_F5	VK_ATTN
VK_PAUSE	VK_F6	VK_CRSEL
VK_CAPSLOCK	VK_F7	VK_EXCEL
VK_ESC	VK_F8	VK_COPY
VK_SPACE	VK_F9	VK_BLK1
VK_PAGEUP	VK_F10	VK_BLK2
VK_PAGEDOWN	VK_F11	VK_MENU
VK_END	VK_F12	VK_DBCSFIRST
VK_HOME	VK_F13	VK_DBCSLAST
VK_LEFT	VK_F14	VK_BIDI_FIRST
VK_UP	VK_F15	VK_BIDI_LAST
VK_RIGH	VK_F16	VK_USERFIRST
VK_DOWN	VK_F17	VK_USERLAST

Related Functions

- WinEnablePhysInput
- WinFocusChange
- WinGetPhysKeyState
- WinQueryFocus
- WinSetFocus
- WinSetKeyboardStateTable

Example Code

This example uses WinGetKeyState to check if mouse button 1 was depressed when a WM_TIMER message was received. A high pitched beep is emitted if it was depressed, and a low pitched beep if it was not.

```
#define INCL_WININPUT           /* Window Input Functions       */
#define INCL_DOSPROCESS         /* OS/2 Process Functions       */
#include <os2.h>

LONG   lKeyState;       /* key state                            */
LONG   vkey;            /* virtual key value                    */

case WM_TIMER:
     /* get key state of mouse button 1 */
     vkey = VK_BUTTON1;
     lKeyState = WinGetKeyState(HWND_DESKTOP, vkey);

     /* emit high pitched beep if mouse button 1 is depressed
        when timer message occurred; otherwise, emit low pitched
        beep */
     if (lKeyState & 0x8000)
        DosBeep(1000,100L);
     else
        DosBeep(200,100L);
```

WinGetLastError

This function returns the error code set by the failure of a Presentation Manager function.

Syntax

```
#define INCL_WINERRORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ERRORID WinGetLastError  (HAB hab)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

erridErrorCode (ERRORID) – returns
Last error code.

The returned error code is a 32-bit quantity. The high order 16 bits is a severity code. The low order 16 bits is the error code.

See the Notes for more information on using the returned ERRORID value.

Remarks

Returns the last nonzero error code, and sets the error code to zero.

The current error code is reset to zero.

In multiple thread applications where there are multiple anchor blocks, errors are stored in the anchor block created by the WinInitialize function of the thread invoking a call. The last error for the process and thread on which this function call is made will be returned.

The returned error code is a 32-bit quantity. The high order 16 bits is a severity code. The low order 16 bits is the error code.

The severity codes are defined as follows:

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

Error codes are described in "Error Codes" and "Error Explanations" in the *Presentation Manager Programming Reference Volume II*.

Two macros have been defined to simplify extracting severity codes and error codes from the returned ERRORID value.

```
/* Extract severity from an errorid */
#define ERRORIDSEV(errid)              (HIUSHORT(errid))

/* Extract error number from an errorid */
#define ERRORIDERROR(errid)            (LOUSHORT(errid))
```

Related Functions

- WinFreeErrorInfo
- WinGetErrorInfo

Example Code

This example uses WinGetLastError to obtain the error code corresponding to the last nonzero error for the specified anchor block. If only the error code is required, this function is preferable to the WinGetErrorInfo/WinFreeErrorInfo call sequence.

```
#define INCL_WINERRORS          /* Window Error Functions       */
#include <os2.h>

ERRORID  erridErrorCode;/* last error id code                   */
HAB   hab;              /* anchor-block handle                  */

/* get last nonzero error for this anchor block */
erridErrorCode = WinGetLastError(hab);
```

WinGetMaxPosition

The WinGetMaxPosition function fills an SWP structure with the maximized-window size and position.

Syntax

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinGetMaxPosition  (HWND  hwnd, PSWP  pswp)
```

Parameters

hwnd (HWND) – input
Frame-window handle.

Identifies the window whose maximum size will be retrieved.

pswp (PSWP) – output
Set window position structure.

Points to the SWP structure that retrieves the size and position of a maximized window.

The SWP_SIZE and SWP_MOVE indicators are set in this parameter on return from this call, implying that the *x*, *y*, *cx*, and *cy* parameters have been initialized.

Returns

fSuccess (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Example Code

This example uses WinGetMaxPosition to determine the maximized position for the window in response to a maximize message (WM_MINMAXFRAME), and then calls WinSetWindowPos to maximize the window to that position.

```
#define INCL_WINFRAMEMGR        /* Window Frame Functions       */
#include <os2.h>

BOOL  fSuccess;         /* Success indicator                    */
HWND  hwnd;             /* window handle                        */
MPARAM  mpParam1;       /* Parameter 1 (window position)        */
PSWP   pSwp;            /* Set window position structure        */

case WM_MINMAXFRAME:
     pSwp = (PSWP)PVOIDFROMMP(mpParam1);

     switch(pSwp->fl)
        {
        case SWP_MAXIMIZE:
             fSuccess = WinGetMaxPosition(hwnd, pSwp);

             WinSetWindowPos(hwnd, 0L,
                 pSwp->x,            /* x pos  */
                 pSwp->y,            /* y pos  */
                 pSwp->cx,           /* x size */
                 pSwp->cy,           /* y size */
                 SWP_MAXIMIZE);      /* flags */
             break;
        }
```

WinGetMinPosition

This function returns the position to which a window is minimized.

Syntax

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinGetMinPosition  (HWND hwnd, PSWP pswp, PPOINTL pptl)
```

Parameters

hwnd (HWND) – input
Frame-window handle.

pswp (PSWP) – output
Set window position structure.

The SWP_SIZE and SWP_MOVE indicators are set in this parameter on return from this function, implying that the *x*, *y*, *cx*, and *cy* parameters have been initialized.

pptl (PPOINTL) – input
Preferred position.

NULL System is to choose the position
Other System is to choose the position nearest to the specified point.

Returns

rc (BOOL) – returns
Success indicator.

TRUE Successful completion.

The WS_MINIMIZE style is set for *hwnd*. This enables the system to determine which other frame windows are minimized, during the enumeration process performed by this function.

Also, the window words QWS_XMINIMIZE and QWS_YMINIMIZE for *hwnd* are initialized. This enables the system to ensure that no windows that have been, or are being, minimized use the same position.

FALSE Error occurred.

Remarks

This function chooses the position for a minimized window. It enumerates all the siblings of the specified window to determine the first available position.

Related Functions

- WinQueryActiveWindow
- WinQueryWindowPos
- WinSaveWindowPos
- WinSetActiveWindow
- WinSetMultWindowPos
- WinSetWindowPos

Example Code

This example uses WinGetMinPosition to determine the minimized position for the window in response to a minimize message (WM_MINMAXFRAME), and then calls WinSetWindowPos to minimize the window to that position.

```
#define INCL_WINFRAMEMGR        /* Window Frame Functions       */
#include <os2.h>

BOOL  fSuccess;         /* Success indicator                    */
HWND  hwnd;             /* window handle                        */
MPARAM  mpParam1;       /* Parameter 1 (window position)        */
PSWP   pSwp;            /* Set window position structure        */

case WM_MINMAXFRAME:
     pSwp = (PSWP)PVOIDFROMMP(mpParam1);

     switch(pSwp->fl)
        {
        case SWP_MINIMIZE:
             fSuccess = WinGetMinPosition(hwnd, pSwp, NULL);

             WinSetWindowPos(hwnd, 0L,
                 pSwp->x,            /* x pos  */
                 pSwp->y,            /* y pos  */
                 pSwp->cx,           /* x size */
                 pSwp->cy,           /* y size */
                 SWP_MINIMIZE);      /* flags */
             break;
        }
```

WinGetMsg

This function gets, waiting if necessary, a message from the thread's message queue and returns when a message conforming to the filtering criteria is available.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinGetMsg (HAB hab, PQMSG pqmsgmsg, HWND hwndFilter,
                ULONG ulFirst, ULONG ulLast)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pqmsgmsg (PQMSG) – output
Message structure.

hwndFilter (HWND) – input
Window filter.

ulFirst (ULONG) – input
First message identity.

ulLast (ULONG) – input
Last message identity.

Returns

rc (BOOL) – returns
Continue message indicator.

TRUE	Message returned is not a WM_QUIT message
FALSE	Message returned is a WM_QUIT message.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

If system or queue hooks are installed, they are called before this function returns.

rc is generally used to determine when to terminate the application's main loop and exit the program.

hwndFilter constrains the returned message to be for a specific window or its children. When *hwndFilter* is null, the returned message can be for any window. The message identity is restricted to the range of message identities specified by *ulFirst* and *ulLast* inclusive. When *ulFirst* and *ulLast* are both zero, any message satisfies the range constraint. When *ulFirst* is greater than *ulLast*, messages except those whose identities lie between *ulFirst* and *ulLast* are eligible to be returned. Messages that do not conform to the filtering criteria remain in the queue.

When *hwndFilter* is null, and *ulFirst* and *ulLast* are both zero, all messages are returned in the order that they were posted to the queue.

By using filtering, messages can be processed in an order that is different from the one in the queue. Filtering is used in situations where applications receive messages of a particular type, rather than having to deal with other types of message at an inconvenient point in the logic of the application. For example, when a "mouse down" message is received, filtering can be used to wait for the "mouse up" message without having to be concerned with receiving other messages.

These constants can also be used when filtering messages:

WM_MOUSEFIRST	Lowest value pointing device message
WM_MOUSELAST	Highest value pointing device message
WM_BUTTONCLICKFIRST	Lowest value pointing device button click message
WM_BUTTONCLICKLAST	Highest value pointing device button click message
WM_DDE_FIRST	Lowest value DDE message
WM_DDE_LAST	Highest value DDE message.

Great care must be taken if filtering is used, to ensure that a message that satisfies the specification of the filtering parameters can occur, otherwise this function cannot complete. For example, calling this function with *ulFirst* and *ulLast* equal to WM_CHAR and with *hwndFilter* set to a window handle that does not have the input focus, prevents this function from returning.

Keystrokes are passed to the WinTranslateAccel call, which implies that accelerator keys are translated into WM_COMMAND or WM_SYSCOMMAND messages, and so are not seen as WM_CHAR messages by the application.

Note: An application must be prepared to receive messages other than those documented in this publication. All messages that an application does not want to handle should be dispatched to the appropriate window procedure using the WinDispatchMsg function.

Related Functions

- WinBroadcastMsg
- WinCancelShutdown
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg

- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Related Messages

- WM_CHAR
- WM_QUIT
- WM_SYSCOMMAND

Example Code

This example uses WinGetMsg to continually loop and retrieve messages from the message queue until a WM_QUIT message occurs.

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HAB     hab;            /* anchor-block handle                  */
HMQ     hmq;            /* message queue handle                 */
PQMSG   pqmsgmsg;       /* message                              */

hab = WinInitialize(0);          /* initialize PM */

hmq = WinCreateMsgQueue(hab, 0); /* create default size queue */

/*
.
.  initialize windows
.
*/

/* get and dispatch messages from queue */
while (WinGetMsg(hab, &qmsgmsg, 0, 0, 0))
   WinDispatchMsg(hab, &qmsgmsg);
```

WinGetNextWindow

This function gets the window handle of the next window in a specified enumeration list.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinGetNextWindow  (HENUM henum)
```

Parameters

henum (HENUM) – input
Enumeration handle.

Returned by previous call to the WinBeginEnumWindows call.

Returns

hwndNext (HWND) – returns
Next window handle in enumeration list.

NULLHANDLE	Error occurred, *henum* was invalid, or all the windows have been enumerated.
Other	Next window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HENUM (0x101C)	An invalid enumeration handle was specified.

Remarks

Enumeration starts with the topmost child window and then proceeds downward through the enumeration list, in z-order at the time the WinBeginEnumWindows was issued, until all the windows have been enumerated. At this point, the call returns NULLHANDLE. The enumeration then wraps and the handle of the topmost child window is returned on the next call. This function does not lock windows.

Related Functions

- WinBeginEnumWindows
- WinEndEnumWindows
- WinEnumDlgItem
- WinIsChild
- WinMultWindowFromIDs
- WinQueryWindow
- WinSetOwner
- WinSetParent

Example Code

This example moves through all the child windows in a enumeration list, using an enumeration handle provided by WinBeginEnumWindows; for each child window, the class name is queried and placed in a buffer.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND  hwndParent;       /* Handle of the window whose child windows
                           are to be enumerated                 */
HWND  hwndNext;         /* current enumeration handle           */
HENUM  henum;           /* enumeration handle                   */
BOOL  fSuccess;         /* success indicator                    */
SHORT sRetLen;          /* returned string length               */
SHORT sLength = 10;     /* string buffer length                 */
char  pchBuffer[10];    /* string buffer                        */

hwndParent = HWND_DESKTOP;

henum = WinBeginEnumWindows(hwndParent);

while ((hwndNext = WinGetNextWindow(henum)) != NULLHANDLE)
   sRetLen = WinQueryClassName(hwndNext, sLength, pchBuffer);

fSuccess = WinEndEnumWindows (henum);
```

WinGetPhysKeyState

This function returns the physical key state.

Syntax

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinGetPhysKeyState  (HWND hwndDeskTop, LONG sc)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

sc (LONG) – input
Hardware scan code.

Contains the scan code value in the low-order byte, and zero in the high-order byte.

Returns

lKeyState (LONG) – returns
Key state.

This value is the OR combination of the following bits:

0x0001	The key has been pressed an odd number of times since the system has been started.
0x0002	The key has been pressed since the last time this function was issued, or since the system has been started if this is the first time the call has been issued.
0x8000	The key is down.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function returns information about the asynchronous (interrupt level) state of the virtual key indicated by the *sc* parameter.

This function returns the physical state of the key; it is not synchronized to the processing of input (see the WinGetKeyState function).

Related Functions

- WinEnablePhysInput
- WinFocusChange
- WinGetKeyState
- WinQueryFocus
- WinSetFocus
- WinSetKeyboardStateTable

Example Code

This example uses WinGetPhysKeyState to check the current state of the caps lock key; if it is depressed, a high pitch beep is emitted, while a low pitch beep is emitted if it is not depressed.

```
#define INCL_WININPUT           /* Window Input Functions       */
#define INCL_DOSPROCESS         /* OS/2 Process Functions       */
#include <os2.h>

LONG   lKeyState;       /* key state                            */
LONG   lScancode;       /* scan code value                      */

/* get physical key state for caps lock key */
lScancode = 0x3a;
lKeyState = WinGetPhysKeyState(HWND_DESKTOP, lScancode);

/* emit high pitched beep if caps lock is currently depressed;
   otherwise, emit low pitched beep */
if (lKeyState & 0x8000)
   DosBeep(1000,100L);
else
   DosBeep(200,100L);
```

WinGetPS

This function gets a cache presentation space.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
HPS WinGetPS  (HWND hwnd)
```

Parameters

hwnd (HWND) – input
Handle of window for which the presentation space is required.

HWND_DESKTOP	The desktop-window handle; a presentation space for the whole of the desktop window is returned
Other	Handle of window for which the presentation space is required.

Returns

hps (HPS) – returns
Presentation-space handle that can be used for drawing in the window.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The presentation space created by this function is a cache "micro presentation space" present in the system. This can be used for simple drawing operations that do not depend on long-term data being stored in the presentation space.

The initial state of the presentation space is the same as that of a presentation space created using GpiCreatePS. The color table is in default color index mode. The visible region associated with *hps* depends upon the window and class styles of *hwnd*:

Style	**Visible region of presentation space**
WS_CLIPCHILDREN	All child windows of the window are excluded.
WS_CLIPSIBLINGS	All the sibling windows of *hwnd* are excluded.
CS_PARENTCLIP	Is the same as that of the parent window of the window. The presentation space origin is established normally, that is, relative to the lower left of the window itself, not its parent.

This style optimizes the use of the presentation space cache by minimizing the calculation of the visible region for child windows.

Any presentation space created by WinGetPS must be released by calling WinReleasePS. This should be done before the application terminates.

Note: This call requires the presence of a message queue and should not be made until after the application's message queue has been created.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Example Code

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.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwnd;           /* parent window                        */
RECTL  rcl;             /* update region                        */
HPS    hps;             /* presentation-space handle            */

case IDM_FILL:
    hps = WinGetPS(hwnd);       /* get presentation space for   */
                                /* the entire window */

    WinQueryWindowRect(hwnd, &rcl); /* get window dimensions    */

    WinFillRect(hps, &rcl, CLR_WHITE); /* clear entire window   */

    WinReleasePS(hps);                  /* release the presentation */
                                        /* space                    */
    return 0L;
```

WinGetScreenPS

This function returns a presentation space that can be used for drawing anywhere on the screen.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HPS WinGetScreenPS  (HWND hwndDeskTop)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Returns

hpsScreenPS (HPS) – returns
Presentation-space handle.

A micro presentation space that can be used for drawing over the entire desktop window (the whole screen).

NULLHANDLE	*hwndDeskTop* is not HWND_DESKTOP or a desktop window handle obtained from the WinQueryDesktopWindow function.
Other	Presentation space handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

Take great care when using this function. The returned presentation space is not clipped to any of the other windows present on the screen. Thus it is possible to draw in regions belonging to windows of other threads and processes.

The WinLockWindowUpdate function should be used to avoid simultaneous updates to the same part of the screen. This does not cause the presentation space returned by this function to become clipped in any way. Care of the appearance of windows of other threads is still the responsibility of the user of the screen presentation space.

When the application finishes using the screen presentation space, it should be destroyed using the WinReleasePS call.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Example Code

This example processes an application-defined message (IDM_FILL). It calls WinGetScreenPS to get a presentation space for the entire desktop window, gets the dimensions of the current window, fills the window, and calls WinReleasePS to release the presentation space.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwnd;           /* parent window                        */
RECTL   rcl;            /* update region                        */
HPS     hps;            /* presentation-space handle            */

case IDM_FILL:
    /* get presentation space for the entire desktop */
    hps = WinGetScreenPS(HWND_DESKTOP);

    WinQueryWindowRect(hwnd, &rcl);    /* get window dimensions */

    WinFillRect(hps, &rcl, CLR_WHITE); /* clear entire window   */

    WinReleasePS(hps);              /* release the presentation space   *
    return 0L;
```

WinGetSysBitmap

This function returns a handle to one of the standard bit maps provided by the system.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HBITMAP WinGetSysBitmap  (HWND hwndDesktop, ULONG ibm)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

ibm (ULONG) – input
System bit-map index value.

SBMP_SYSMENU	System menu
SBMP_SYSMENUDEP	System menu in depressed state
SBMP_SBUPARROW	Scroll bar up arrow
SBMP_SBUPARROWDEP	Scroll bar up arrow in depressed state
SBMP_SBUPARROWDIS	Scroll bar up arrow in disabled state
SBMP_SBDNARROW	Scroll bar down arrow
SBMP_SBDNARROWDEP	Scroll bar down arrow in depressed state
SBMP_SBDNARROWDIS	Scroll bar down arrow in disabled state
SBMP_SBRGARROW	Scroll bar right arrow
SBMP_SBRGARROWDEP	Scroll bar right arrow in depressed state
SBMP_SBRGARROWDIS	Scroll bar right arrow in disabled state
SBMP_SBLFARROW	Scroll bar left arrow
SBMP_SBLFARROWDEP	Scroll bar left arrow in depressed state
SBMP_SBLFARROWDIS	Scroll bar left arrow in disabled state
SBMP_MENUCHECK	Menu check mark
SBMP_MENUATTACHED	Cascading menu mark
SBMP_CHECKBOXES	Check box or radio button check marks
SBMP_COMBODOWN	Combobox down arrow
SBMP_BTNCORNERS	Push-button corners
SBMP_MINBUTTON	Minimize button
SBMP_MINBUTTONDEP	Minimize button in depressed state
SBMP_MAXBUTTON	Maximize button
SBMP_MAXBUTTONDEP	Maximize button in depressed state
SBMP_RESTOREBUTTON	Restore button
SBMP_RESTOREBUTTONDEP	Restore button in depressed state

SBMP_CHILDSYSMENU	System menu for child windows
SBMP_CHILDSYSMENUDEP	System menu for child windows in depressed state
SBMP_DRIVE	Drive
SBMP_FILE	File
SBMP_FOLDER	Folder
SBMP_TREEPLUS	Used by the file system to indicate that an entry in the directory can be expanded.
SBMP_TREEMINUS	Used by the file system to indicate that an entry in the directory can be collapsed.
SBMP_PROGRAM	Used by the file system to mark .EXE and .COM files.
SBMP_SIZEBOX	Used by some applications to display a sizebox in the bottom-right corner of a frame window.

Returns

hbm (HBITMAP) – returns
System bit-map handle.

NULLHANDLE	Error occurred
Other	System bit-map handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.
PMERR_RESOURCE_NOT_FOUND (0x100A)	The specified resource identity could not be found.

Remarks

The bit map returned can be used for any of the normal bit-map operations. This function provides a new copy of the system bit map each time it is called. The application should release any bit maps it gets with this function by using GpiDeleteBitmap.

Related Functions

- WinDrawBitmap
- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinFillRect
- WinInvertRect
- WinQueryPresParam
- WinRemovePresParam
- WinScrollWindow
- WinSetPresParam

Example Code

This example uses WinGetSysBitmap to retrieve the system defined handle for the menu check mark bit map during the window creation phase. The bit-map handle is then later used to draw the check mark in response to user selection of a menu item.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINPOINTERS        /* Window Pointer Functions     */
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINMENUS           /* Window Menu Functions        */
#include <os2.h>

HPS   hps;              /* presentation-space handle            */
HBITMAP hbmCheck;       /* check mark bit-map handle            */
HWND    hwndMenu;       /* menu handle                          */
USHORT  usItemId;       /* menu item id                         */
MPARAM  mp1;            /* Parameter 1 (menu item id)           */
MPARAM  mp2;            /* Parameter 2 (menu handle)            */
RECTL   rclItem;        /* item border rectangle                */

case WM_CREATE:
     /* obtain check mark bit-map handle */
     hbmCheck = WinGetSysBitmap(HWND_DESKTOP, SBMP_MENUCHECK);

case WM_MENUSELECT:
     usItemId = SHORT1FROMMP(mp1);
     hwndMenu = HWNDFROMMP(mp2);

     /* get rectangle of selected item */
     WinSendMsg(hwndMenu,
                MM_QUERYITEMRECT,
                MPFROM2SHORT(usItemId, TRUE),
                MPFROMP(&rclItem));

     /* draw the check mark in the lower left corner of item's
        rectangle */
     if (hbmCheck != NULL)
        {
        WinDrawBitmap(hps,
                      hbmCheck,         /* check mark           */
                      NULL,             /* draw whole bit map   */
                      (PPOINTL)&rclItem,/* bit-map destination  */
                      0L,               /* ignored since color  */
                      0L,               /* bit map */
                      DBM_NORMAL);      /* draw normal size     */
        }
```

WinInflateRect

This function expands a rectangle.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinInflateRect (HAB hab, PRECTL prcl, LONG cx, LONG cy)
```

Parameters

hab (HAB) – input
Anchor-block handle.

prcl (PRECTL) – in/out
Rectangle to be expanded.

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type WRECT can also be used, if supported by the language.

cx (LONG) – input
Horizontal expansion.

cy (LONG) – input
Vertical expansion.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function adjusts the size of the rectangle by applying the *cx* parameter horizontally at both vertical edges and the *cy* parameter vertically at both horizontal edges.

The *cx* parameter is subtracted from the left and added to the right of the rectangle, and the *cy* parameter is subtracted from the bottom and added to the top of the rectangle.

If the values of the *cx* and *cy* parameters are both positive, the rectangle is enlarged and surrounds the original rectangle. Conversely, if both these values are negative, the rectangle is reduced in size and is inset with respect to the original rectangle.

Related Functions

- WinCopyRect
- WinEqualRect
- WinFillRect
- WinIntersectRect
- WinIsRectEmpty
- WinOffsetRect
- WinPtInRect
- WinSetRect
- WinSetRectEmpty
- WinSubtractRect
- WinUnionRect

Example Code

This example doubles the size of a rectangle if the mouse is double clicked (WM_BUTTON1DBLCLK) within the rectangle (WinPtInRect).

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions   */
#include <os2.h>

BOOL  fSuccess;         /* success indicator                    */
HAB     hab;            /* anchor-block handle                  */
RECTL prclRect1 = {0,0,100,100}; /* rectangle                   */
LONG   lcx = 100;       /* Horizontal expansion                 */
LONG   lcy = 100;       /* Vertical expansion                   */
POINTL ptl;             /* current mouse position               */
MPARAM mpParam1;        /* Parameter 1  (x,y) point value       */

case WM_BUTTON1DBLCLK:
     ptl.x = (LONG) SHORT1FROMMP(mpParam1);
     ptl.y = (LONG) SHORT2FROMMP(mpParam1);

     if (WinPtInRect(hab, &prclRect1, &ptl))
        fSuccess = WinInflateRect(hab, &prclRect1, lcx, lcy);
```

WinInitialize

This function initializes the PM facilities for use by an application.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
HAB WinInitialize (ULONG flOptions)
```

Parameters

flOptions (ULONG) – input
Initialization options.

0 The initial state for newly created windows is that all messages for the window are available for processing by the application.

This is the only option available in PM.

Returns

hab (HAB) – returns
Anchor-block handle.

NULLHANDLE	An error occurred.
Other	Anchor-block handle.

Remarks

This must be the first PM call issued by any application thread using Presentation Manager facilities.

It returns *hab*, which is NULL if the initialization is not successful.

The operating system does not generally use the information supplied by the *hab* parameter to its calls; instead, it deduces it from the identity of the thread that is making the call. Thus an OS/2 application is not required to supply any particular value as the *hab* parameter. However, in order to be portable to other environments, an application must provide the *hab*, that is returned by the WinInitialize function of the thread, to any OS/2 function that requires it.

flOptions determines the initial state of message processing with respect to a created window.

Related Functions

- WinCancelShutdown
- WinCreateMsgQueue
- WinTerminate

Example Code

This example uses WinInitialize to obtain an anchor block and initialize Presentation Manager.

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HAB     hab;            /* anchor-block handle                  */
HMQ     hmq;            /* message queue handle                 */
QMSG    qmsg;           /* message                              */

hab = WinInitialize(0);          /* initialize PM */

hmq = WinCreateMsgQueue(hab, 0); /* create default size queue */

/*
.
.  initialize windows
.
*/

/* get and dispatch messages from queue */
while (WinGetMsg(hab, &qmsg, 0, 0, 0))
   WinDispatchMsg(hab, &qmsg);
```

WinInSendMsg

This function determines whether the current thread is processing a message sent by another thread.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinInSendMsg  (HAB hab)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

rc (BOOL) – returns
Message-processing indicator.

TRUE	Current thread is processing a message sent by another thread
FALSE	Current thread is not processing a message, or an error occurred.

Remarks

If the message is from another thread this function determines whether or not the message was initiated by the active thread. The "active thread" is the thread associated with the current active window. (See also the WinIsThreadActive function.)

Typically this function is used by applications to determine how to proceed with errors when the window processing the message is not the active window. For example, if the active window uses the WinSendMsg function to send a request for information to another window, the other window cannot become active until it returns control from the WinSendMsg function. The only methods an inactive window has to inform the user of an error are to create a message box (see WinMessageBox), or to flash a window (see WinFlashWindow).

This function can be used to tell if a function is being called recursively.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg

- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example determines, during a WM_ERROR message, if the current thread is processing a message sent by another thread using WinInSendMsg; if so, a message box is generated with the error information to alert the active window that originally sent the message.

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#include <os2.h>

HAB     hab;            /* anchor-block handle                  */
BOOL  fSuccess;         /* Success indicator                    */
MPARAM  mpParam1;       /* Parameter 1                          */
USHORT  errorcode;      /* error code                           */
CHAR szMsg[100];        /* message text                         */
HWND   hwnd;            /* handle of window with error msg      */

case WM_ERROR:
     /* get error code */
     errorcode = SHORT1FROMMP(mpParam1);

     if (WinInSendMsg(hab))
        {
        /* parse and display error message */
        sprintf(szMsg, "Error code %d occurred", errorcode);
        WinMessageBox(HWND_DESKTOP,
            hwnd,                       /* client-window handle  */
            szMsg,                      /* body of the message   */
            "Error notification",       /* title of the message  */
            0,                          /* message box id        */
            MB_NOICON | MB_OK);         /* icon and button flags */
        }
```

WinInsertLboxItem

This macro inserts text into a list box at index, index may be a LIT_ constant. The macro returns the actual index where it was inserted.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinInsertLboxItem (HWND hwndLbox, LONG index, PSZ psz)
```

Parameters

hwndLbox (HWND) – input
List box handle.

index (LONG) – input
Index of the list box item.

psz (PSZ) – input
Text to be inserted.

Returns

lRetIndex (LONG) – returns
Actual index where it was inserted.

Remarks

This macro is defined as:

```
#define WinInsertLboxItem(hwndLbox, index, psz)  \
        ((LONG)WinSendMsg(hwndLbox,              \
                          LM_INSERTITEM,         \
                          MPFROMLONG(index),     \
                          MPFROMP(psz)))
```

This macro requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- LM_INSERTITEM

Example Code

This example calls WinInsertLboxItem to insert items in a list box as part of initializing a dialog (WM_INITDLG message).

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINLISTBOXES       /* Window List Box definitions  */
#include <os2.h>

LONG    lIndex;         /* inserted item index                  */
HWND    hwndLbox;       /* list box window handle               */
MPARAM  mpParam1;       /* Parameter 1 (window handle)          */
/* Array of list box item names */
PSZ     pszItems[3] = {"Item1", "Item2", "Item3"};

case WM_INITDLG:
       .
       .
     /*******************************/
     /* Initialize List Box Control */
     /*******************************/

     /* get handle of list box */
     hwndLbox = HWNDFROMMP(mpParam1);

     /* insert 3 items into list box */
     lIndex = WinInsertLboxItem(hwndLbox, LIT_END, pszItems[0]);
     lIndex = WinInsertLboxItem(hwndLbox, LIT_END, pszItems[1]);
     lIndex = WinInsertLboxItem(hwndLbox, LIT_END, pszItems[2]);
       .
       .
```

WinIntersectRect

This function calculates the intersection of the two source rectangles and returns the result in the destination rectangle.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinIntersectRect (HAB hab, PRECTL pcrlDst, PRECTL pcrlSrc1,
                       PRECTL pcrlSrc2)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pcrlDst (PRECTL) – output
Intersection rectangle.

Is the intersection of *pcrlSrc1* and *pcrlSrc2*.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT can also be used, if supported by the language.

pcrlSrc1 (PRECTL) – input
First rectangle.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT can also be used, if supported by the language.

pcrlSrc2 (PRECTL) – input
Second rectangle.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT can also be used, if supported by the language.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Source rectangles intersect
FALSE	Source rectangles do not intersect, or an error occurred.

Remarks

If there is no intersection, an empty rectangle is returned in *pcrlDst*.

Related Functions

- WinCopyRect
- WinEqualRect
- WinFillRect
- WinInflateRect
- WinIsRectEmpty
- WinOffsetRect
- WinPtInRect
- WinSetRect
- WinSetRectEmpty
- WinSubtractRect
- WinUnionRect

Example Code

This example determines the intersection of two rectangles and places the result in a third rectangle structure.

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions   */
#include <os2.h>

BOOL   fSuccess;         /* success indicator                    */
HAB      hab;            /* anchor-block handle                  */
PRECTL prclRect1 = {0,0,100,100}; /* rectangle  1                */
PRECTL prclRect2 = {0,0,200,200}; /* rectangle  2                */
PRECTL prclDest;         /* destination rectangle                */

fSuccess = WinIntersectRect(hab, &prclDest, &prclRect1,
                            &prclRect2);
```

WinInvalidateRect

This function adds a rectangle to a window's update region.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinInvalidateRect (HWND hwnd, PRECTL pwrc, BOOL fIncludeChildren)
```

Parameters

hwnd (HWND) – input
Handle of window whose update region is to be changed.

HWND_DESKTOP	This function applies to the whole screen (or desktop).
Other	Handle of window whose update region is to be changed.

pwrc (PRECTL) – input
Update rectangle.

Note: The value of each field in this structure must be in the range -32768 through 32767. The data type WRECT can also be used, if supported by the language.

NULL	The whole window is to be added into the window's update region.
Other	Rectangle to be added to the window's update region.

fIncludeChildren (BOOL) – input
Invalidation-scope indicator.

TRUE	Include the descendants of *hwnd* in the invalid rectangle.
FALSE	Include the descendants of *hwnd* in the invalid rectangle, but only if the parent does not have a WS_CLIPCHILDREN style.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

The update region is a subregion of a window that is deemed "invalid" or incorrect in visual terms and in need of redrawing.

If the window has a CS_SYNCPAINT style, it is redrawn during the processing of this function and the update region should be NULL on return from this function.

If the window has a WS_CLIPCHILDREN style with part of its update region overlapping child windows with a CS_SYNCPAINT style, those children are updated before this function returns.

This function should not be called in response to a WM_PAINT request for windows of style CS_SYNCPAINT. CS_SYNCPAINT means that windows are updated synchronously when invalidated, which generates a WM_PAINT message. Thus, invalidating the window in response to a WM_PAINT message would cause another invalidate, and another WM_PAINT, and so on.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Related Messages

- WM_PAINT

Example Code

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.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwnd;           /* parent window                        */
RECTL  rcl;             /* update region                        */

WinQueryWindowRect(hwnd, &rcl);

WinInvalidateRect(hwnd,   /* window to invalidate    */
    &rcl,                 /* invalid rectangle       */
    FALSE);               /* do not include children */
```

WinInvalidateRegion

This function adds a region to a window's update region.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinInvalidateRegion (HWND hwnd, HRGN hrgn, BOOL fIncludeChildren)
```

Parameters

hwnd (HWND) – input
Handle of window whose update region is to be changed.

HWND_DESKTOP	This function applies to the whole screen (or desktop).
Other	Handle of window whose update region is to be changed.

hrgn (HRGN) – input
Handle of the region to be added to the update region of the window.

NULLHANDLE	The whole window is to be added into the window's update region.
Other	Handle of the region to be added to the window's update region.

fIncludeChildren (BOOL) – input
Invalidation-scope indicator.

TRUE	Include the descendants of *hwnd* in the invalid rectangle.
FALSE	Include the descendants of *hwnd* in the invalid rectangle, but only if the parent does not have a WS_CLIPCHILDREN style.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_HRGN_BUSY (0x2034)	An internal region busy error was detected. The region was locked by one thread during an attempt to access it from another thread.

PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

The update region is a subregion of a window that is deemed "invalid" or incorrect in visual terms and is in need of redrawing.

If the window has a CS_SYNCPAINT style, it is redrawn during the processing of this function and the update region should be NULL on return from this function.

If the window has a WS_CLIPCHILDREN style with part of its update region overlapping child windows with a CS_SYNCPAINT style, those children are updated before this function returns.

This function should not be called in response to a WM_PAINT request for windows of style CS_SYNCPAINT. CS_SYNCPAINT means that windows are updated synchronously when invalidated, which generates a WM_PAINT message. Thus, invalidating the window in response to a WM_PAINT message would cause another invalidate, another WM_PAINT, and so on.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Related Messages

- WM_PAINT

Example Code

This example invalidates the entire window by adding the whole window to the window's update region using WinInvalidateRegion. This single call accomplishes the same as paired calls to WinQueryWindowRect and WinInvalidateRect. If less than the entire window is desired, the NULL value in the second parameter can be replaced with a region handle that corresponds to a subregion of the window.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwnd;           /* window handle                        */

WinInvalidateRegion(hwnd, NULLHANDLE, 0);
```

WinInvertRect

This function inverts a rectangular area.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinInvertRect (HPS hps, PRECTL prclRect)
```

Parameters

hps (HPS) – input
Presentation-space handle.

The presentation space contains the rectangle to be inverted.

prclRect (PRECTL) – input
Rectangle to be inverted.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT can also be used, if supported by the language.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

Inversion is a logical-NOT operation and has the effect of flipping the bits of each pel.

Related Functions

- WinDrawBitmap
- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinFillRect
- WinGetSysBitmap
- WinQueryPresParam
- WinRemovePresParam
- WinScrollWindow
- WinSetPresParam

Example Code

This example inverts a rectangle if the mouse button is released (WM_BUTTON1UP) within the rectangle (WinPtInRect); the presentation space handle is obtained via WinBeginPaint.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

BOOL  fSuccess;          /* success indicator                   */
HAB     hab;             /* anchor-block handle                 */
RECTL prclRect1 = {0,0,100,100}; /* rectangle                   */
HWND   hwnd;             /* client window handle                */
HPS    hps;              /* presentation-space handle           */
POINTL ptl;              /* current mouse position              */
MPARAM mpParam1;         /* Parameter 1  (x,y) point value      */

case WM_BUTTON1UP:
     ptl.x = (LONG) SHORT1FROMMP(mpParam1);
     ptl.y = (LONG) SHORT2FROMMP(mpParam1);

     if (WinPtInRect(hab, &prclRect1, &ptl))
        {
        hps = WinBeginPaint(hwnd, NULLHANDLE, &prclRect1);
        fSuccess = WinInvertRect(hps, &prclRect1);
        WinEndPaint(hps);
        }
```

WinIsChild

This function tests if one window is a descendant of another window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinIsChild (HWND hwnd, HWND hwndParent)
```

Parameters

hwnd (HWND) – input
Child-window handle.

hwndParent (HWND) – input
Parent-window handle.

Returns

fRelated (BOOL) – returns
Related indicator.

TRUE Child window is a descendant of the parent window, or is equal to it

FALSE Child window is not a descendant of the parent, or is an Object Window (even if *hwndParent* is specified as the desktop or HWND_DESKTOP), or an error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001) An invalid window handle was specified.

Related Functions

- WinBeginEnumWindows
- WinEndEnumWindows
- WinEnumDlgItem
- WinGetNextWindow
- WinMultWindowFromIDs
- WinQueryWindow
- WinSetOwner
- WinSetParent

Example Code

This example uses WinIsChild to determine if one window is a descendant of another window.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwndChild;      /* child window to check                */
HWND    hwndParent;     /* parent window to check               */

if (WinIsChild(hwndChild, hwndParent))
   {
   /* hwndChild is a descendant of hwndParent */
   }
else
   {
   /* hwndChild is not a descendant of hwndParent */
   }
```

WinIsControlEnabled

This macro returns the state (enable/disable) of the specified item in the dialog template within a dialog box.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinIsControlEnabled (HWND hwndDlg, USHORT usId)
```

Parameters

hwndDlg (HWND) – input
Dialog window handle.

usId (USHORT) – input
Identity of the specified item.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This macro expands to:

```
#define WinIsControlEnabled(hwndDlg, usId)
   ((BOOL)WinIsWindowEnabled(WinWindowFromID(hwndDlg, usId)))
```

This function requires the existence of a message queue.

Related Functions

- WinIsWindowEnabled
- WinWindowFromID

Example Code

This example uses WinIsControlEnabled to determine if a selected control is valid; if it is not, an error message box is displayed.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#include <os2.h>

HWND    hwndDlg;        /* dialog window                        */
MPARAM  mp1;            /* Parameter 1                          */
USHORT  usId;           /* dialog control id                    */

case WM_CONTROL:
     usId = SHORT1FROMMP(mp1);

     if (!WinIsControlEnabled(hwndDlg, usId))
        {
        WinMessageBox(HWND_DESKTOP,
            hwndDlg,                /* client-window handle  */
            "Control is not valid",  /* body of the message   */
            "Error notification",    /* title of the message  */
            0,                       /* message box id        */
            MB_NOICON | MB_OK);      /* icon and button flags */
        }
```

WinIsMenuItemChecked

This macro returns the state (checked/not checked) of the identified menu item.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinIsMenuItemChecked (HWND hwndMenu, USHORT usId)
```

Parameters

hwndMenu (HWND) – input
Menu window handle.

usId (USHORT) – input
Identity of the menu item.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This macro expands to:

```
#define WinIsMenuItemChecked(hwndMenu, usId)
  ((BOOL)WinSendMsg(hwndMenu,
                    MM_QUERYITEMATTR,
                    MPFROM2SHORT(usId, TRUE),
                    MPFROM2SHORT(MIA_CHECKED)))
```

This function requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- MM_QUERYITEMATTR

Example Code

This example uses WinIsMenuItemChecked to query the check attribute of a selected menu item before setting the check state of that menu item.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

USHORT  usItemId;       /* menu item id                         */
HWND    hwndMenu;       /* menu handle                          */
BOOL    usChkstate;     /* new checked state                    */
BOOL    fSuccess;       /* success indicator                    */
MPARAM  mp1;            /* Parameter 1 (menu item id)           */
MPARAM  mp2;            /* Parameter 2 (menu handle)            */

case WM_MENUSELECT:
     usItemId = SHORT1FROMMP(mp1);
     hwndMenu = HWNDFROMMP(mp2);

     /* query current check state */
     usChkstate = WinIsMenuItemChecked(hwndMenu, usItemId);

     /* set menu item check state */
     fSuccess = WinCheckMenuItem(hwndMenu, usItemId, usChkstate);
```

WinIsMenuItemEnabled

This macro returns the state (enable/disable) of the menu item specified.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinIsMenuItemEnabled  (HWND hwndMenu, USHORT usId)
```

Parameters

hwndMenu (HWND) – input
Menu window handle.

usId (USHORT) – input
Identity of the menu item.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This macro expands to:

```
#define WinIsMenuItemEnabled(hwndMenu, usId)
  (!(BOOL)WinSendMsg(hwndMenu,
                     MM_QUERYITEMATTR,
                     MPFROM2SHORT(usId, TRUE),
                     MPFROMSHORT(MIA_DISABLED)))
```

This function requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- MM_QUERYITEMATTR

Example Code

This example uses WinIsMenuItemEnabled to determine if a selected menu item is available for use. If the item is not valid (WinIsMenuItemValid) or not enabled, a beep is emitted.

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINMENUS           /* Window Menu Functions        */
#define INCL_DOSPROCESS         /* OS/2 Process Functions       */
#include <os2.h>

MPARAM  mp1;            /* Parameter 1 (rectl structure)        */
MPARAM  mp2;            /* Parameter 2 (frame boolean)          */
USHORT  usItemId;       /* menu item id                         */
HWND    hwndMenu;       /* menu handle                          */

case WM_MENUSELECT:
     usItemId = SHORT1FROMMP(mp1);
     hwndMenu = HWNDFROMMP(mp2);

     /* if menu item is not valid or enabled, emit beep */
     if (!WinIsMenuItemValid(hwndMenu, usItemId) ||
         !WinIsMenuItemEnabled(hwndMenu, usItemId))
        DosBeep(800,100L);
```

WinIsMenuItemValid

This macro returns TRUE if the specified item is a valid choice.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinIsMenuItemValid  (HWND hwndMenu, USHORT usId)
```

Parameters

hwndMenu (HWND) – input
Menu window handle.

usId (USHORT) – input
Identity of the menu item.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This macro expands to:

```
#define WinIsMenuItemValid(hwndMenu, usId)
  ((BOOL)WinSendMsg(hwndMenu,
                    MM_ISITEMVALID,
                    MPFROM2SHORT(usId, TRUE),
                    MPFROMSHORT(FALSE)))
```

This function requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- MM_ISITEMVALID

Example Code

This example uses WinIsMenuItemValid to determine if a selected menu item is available for use. If the item is not enabled (WinIsMenuItemEnabled) or not valid, a beep is emitted.

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINMENUS           /* Window Menu Functions        */
#define INCL_DOSPROCESS         /* OS/2 Process Functions       */
#include <os2.h>

MPARAM  mp1;            /* Parameter 1 (rectl structure)        */
MPARAM  mp2;            /* Parameter 2 (frame boolean)          */
USHORT  usItemId;       /* menu item id                         */
HWND    hwndMenu;       /* menu handle                          */

case WM_MENUSELECT:
     usItemId = SHORT1FROMMP(mp1);
     hwndMenu = HWNDFROMMP(mp2);

     /* if menu item is not valid or enabled, emit beep */
     if (!WinIsMenuItemValid(hwndMenu, usItemId) ||
         !WinIsMenuItemEnabled(hwndMenu, usItemId))
        DosBeep(800,100L);
```

WinIsPhysInputEnabled

This function returns the status of hardware input (on/off).

Syntax

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinIsPhysInputEnabled (HWND hwndDeskTop)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP The desktop-window handle

Returns

rc (BOOL) – returns
Return value.

TRUE If input is enabled.
FALSE If input is disabled.

Related Functions

- WinEnablePhysInput

Example Code

This example uses WinIsPhysInputEnabled to determine if physical input is enabled; if it is not, then WinEnablePhysInput is called to enable it.

```
#define INCL_WININPUT           /* Window Input Functions       */
#include <os2.h>


if (!WinIsPhysInputEnabled(HWND_DESKTOP))
   /* enable queuing of physical input */
   WinEnablePhysInput(HWND_DESKTOP, TRUE);
```

WinIsRectEmpty

This function checks whether a rectangle is empty.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinIsRectEmpty (HAB hab, PRECTL prclprc)
```

Parameters

hab (HAB) – input
Anchor-block handle.

prclprc (PRECTL) – input
Rectangle to be checked.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT can also be used, if supported by the language.

Returns

rc (BOOL) – returns
Empty indicator.

TRUE	Rectangle is empty
FALSE	Rectangle is not empty.

Remarks

A rectangle has area if its left edge coordinate is less than its right edge coordinate, and its bottom edge coordinate is less than its top edge coordinate. An empty rectangle is one with no area.

Related Functions

- WinCopyRect
- WinEqualRect
- WinFillRect
- WinInflateRect
- WinIntersectRect
- WinOffsetRect
- WinPtInRect
- WinSetRect
- WinSetRectEmpty
- WinSubtractRect
- WinUnionRect

Example Code

This example checks if a rectangle is empty (i.e. it has no area).

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions   */
#include <os2.h>

BOOL  fEmpty;           /* empty indicator                      */
HAB     hab;            /* anchor-block handle                  */
RECTL prclRect1 = {0,0,100,100}; /* rectangle                   */

fEmpty = WinIsRectEmpty(hab, &prclRect1);
```

WinIsSOMDDReady

This function returns the current state of the DSOM daemon started by the Workplace Shell process using the WinRestartSOMDD.

Syntax

```
#define INCL_WPCLASS
#include <os2.h>
BOOL WinIsSOMDDReady ()
```

Parameters

None.

Returns

fReady (BOOL) – returns

Flag indicating the DSOM daemon status.

TRUE	SOMDD has been started by the Workplace Shell process.
FALSE	SOMDD has not been started by the Workplace Shell process.

Remarks

This function returns the state of the DSOM daemon started only by the Workplace Shell process using a call to WinRestartSOMDD. This does not include the status of the DSOM daemon if started by any other process.

Note: This function requires that the PM Shell is up and running.

Related Functions

- WinIsWPDServerReady
- WinRestartSOMDD
- WinRestartWPDServer

Example Code

This example starts the DSOM daemon and, a short time later, checks to see if it indeed has started successfully.

```
#define  INCL_WPCLASS
#include <os2.h>

WinRestartSOMDD();
  .
  .
  .
if ( WinIsSOMDDReady() )
   somPrintf ("SOMDD is running\n")
else
   somPrintf ("SOMDD failed to start, possiblly started already"
              "by another process\n");
```

WinIsThreadActive

This function determines whether the active window belongs to the calling execution thread.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinIsThreadActive (HAB hab)
```

Parameters

hab (HAB) – input
Anchor-block handle of calling thread.

Returns

rc (BOOL) – returns
Active-window indicator.

TRUE Active window belongs to calling thread
FALSE Active window does not belong to calling thread.

Related Functions

- WinEnableWindow
- WinIsWindow
- WinIsWindowEnabled
- WinQueryDesktopWindow
- WinQueryObjectWindow
- WinQueryWindowDC
- WinQueryWindowProcess
- WinQueryWindowRect
- WinWindowFromDC
- WinWindowFromID
- WinWindowFromPoint

Example Code

This example uses WinIsThreadActive to verify that the active window belongs to the current thread before querying and enabling the system menu window via WinIsWindowEnabled and WinEnableWindow.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINFRAMEMGR        /* Window Frame Functions       */
#include <os2.h>

HAB   hab;              /* anchor-block handle                  */
HWND    hwndSysmenu;    /* system menu window                   */
HWND    hwnd;           /* parent window                        */
BOOL  fSuccess;         /* success indicator                    */

/* if the active window belongs to the current thread, query the
   enabled status of the system menu */
if (WinIsThreadActive(hab))
   {
   /* obtain handle for system menu */
   hwndSysmenu = WinWindowFromID(hwnd,FID_SYSMENU);

   /* if system menu is not enabled, enable it */
   if (!WinIsWindowEnabled(hwndSysmenu))
      fSuccess = WinEnableWindow(hwndSysmenu, TRUE);
   }
```

WinIsWindow

This function determines if a window handle is valid.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinIsWindow (HAB hab, HWND hwnd)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hwnd (HWND) – input
Window handle.

Returns

rc (BOOL) – returns
Validity indicator.

TRUE	Window handle is valid
FALSE	Window handle is not valid.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinEnableWindow
- WinIsThreadActive
- WinIsWindowEnabled
- WinQueryDesktopWindow
- WinQueryObjectWindow
- WinQueryWindowDC
- WinQueryWindowProcess
- WinQueryWindowRect
- WinWindowFromDC
- WinWindowFromID
- WinWindowFromPoint

Example Code

This example uses WinIsWindow to verify that the parent window is valid before querying and enabling the system menu window via WinIsWindowEnabled and WinEnableWindow.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINFRAMEMGR        /* Window Frame Functions       */
#include <os2.h>

HAB   hab;              /* anchor-block handle                  */
HWND    hwndSysmenu;    /* system menu window                   */
HWND    hwnd;           /* parent window                        */
BOOL  fSuccess;         /* success indicator                    */

/* if handle specifies a valid window, query the enabled status of
   the system menu */
if (WinIsWindow(hab, hwnd))
   {
   /* obtain handle for system menu */
   hwndSysmenu = WinWindowFromID(hwnd,FID_SYSMENU);

   /* if system menu is not enabled, enable it */
   if (!WinIsWindowEnabled(hwndSysmenu))
      fSuccess = WinEnableWindow(hwndSysmenu, TRUE);
   }
```

WinIsWindowEnabled

This function returns the enabled/disabled state of a window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinIsWindowEnabled  (HWND hwnd)
```

Parameters

hwnd (HWND) -- input
Window handle.

Returns

rc (BOOL) -- returns
Enabled-state indicator.

TRUE	Window is enabled
FALSE	Window is not enabled.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinEnableWindow
- WinIsThreadActive
- WinIsWindow
- WinQueryDesktopWindow
- WinQueryObjectWindow
- WinQueryWindowDC
- WinQueryWindowProcess
- WinQueryWindowRect
- WinWindowFromDC
- WinWindowFromID
- WinWindowFromPoint

Example Code

This example uses WinIsWindowEnabled to check that the parent window is currently disabled before calling WinEnableWindow to enable the system menu window.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINFRAMEMGR        /* Window Frame Functions       */
#include <os2.h>

HAB   hab;              /* anchor-block handle                  */
HWND    hwndSysmenu;    /* system menu window                   */
HWND    hwnd;           /* parent window                        */
BOOL  fSuccess;         /* success indicator                    */

/* obtain handle for system menu */
hwndSysmenu = WinWindowFromID(hwnd,FID_SYSMENU);

/* if system menu is not enabled, enable it */
if (!WinIsWindowEnabled(hwndSysmenu))
   fSuccess = WinEnableWindow(hwndSysmenu, TRUE);
```

WinIsWindowShowing

This function determines whether any part of the window *hwnd* is physically visible.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinIsWindowShowing  (HWND  hwnd)
```

Parameters

hwnd (HWND) – input
Window handle.

Returns

rc (BOOL) – returns
Showing state indicator.

TRUE	Some part of the window is displayed on the screen
FALSE	No part of the window is displayed on the screen.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function is useful for applications that constantly output new information. If value FALSE is returned (that is, no part of the window is physically visible), the application can choose not to redraw, since redrawing is not necessary.

If an application is using WinIsWindowShowing, it must issue the call every time it has new information that needs to be updated. If this is not done, invalid screen content could result. The alternative to this approach for a constantly-updating application that has new information is for it to invalidate its window and redraw within a WinBeginPaint - WinEndPaint sequence.

FALSE is returned if the PM session is not currently visible.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion

- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Example Code

This example uses WinIsWindowShowing to check if any part of the window is physically visible before causing a redraw of the window via WinInvalidateRect.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINFRAMEMGR        /* Window Frame Functions       */
#include <os2.h>

HWND    hwnd;           /* window handle                        */
RECTL  rcl;             /* update region                        */

/* if any part of the window is visible, cause a redraw */
if (WinIsWindowShowing(hwnd))
   {
   WinQueryUpdateRect(hwnd, &rcl);

   WinInvalidateRect(hwnd,   /* window to invalidate    */
       &rcl,                 /* invalid rectangle       */
       FALSE);               /* do not include children */
   }
```

WinIsWindowVisible

This function returns the visibility state of a window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinIsWindowVisible  (HWND hwnd)
```

Parameters

hwnd (HWND) – input
Window handle.

Returns

rc (BOOL) – returns
Visibility-state indicator.

TRUE	Window and all its parents have the WS_VISIBLE style bit set on
FALSE	Window or one of its parents have the WS_VISIBLE style bit set off.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

Because *rc* reflects only the values of WS_VISIBLE style bits, *rc* may be set to TRUE even if *hwnd* is totally obscured by other windows. Use WinIsWindowShowing to determine if any part of the window is actually visible.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinLockVisRegions

- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Example Code

This example uses WinIsWindowVisible to query the visibility state of a window (i.e. the value of the WS_VISIBLE style bits) when the window is created, so that the window can be designated as visible, if necessary, by calling WinEnableWindowUpdate.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwnd;           /* parent window                        */
BOOL  fSuccess;         /* success indicator                    */

case WM_CREATE:
     /* if window has WS_VISIBLE off, set state to visible */
     if (!WinIsWindowVisible(hwnd))
        {
        /* set state to visible and cause WM_PAINT message */
        fSuccess = WinEnableWindowUpdate(hwnd, TRUE);
        }
```

WinIsWPDServerReady

This function returns the current state of the Workplace Shell DSOM Server.

Syntax

```
#define INCL_WPCLASS
#include <os2.h>
BOOL WinIsWPDServerReady ()
```

Parameters

None.

Returns

fReady (BOOL) – returns

Flag indicating the Workplace Shell DSOM Server status.

TRUE	Workplace Shell DSOM Server is ready.
FALSE	Workplace Shell DSOM Server is not ready.

Remarks

This function returns the ready status of the Workplace Shell DSOM Server.

Note: This function requires that the PM Shell is up and running.

Related Functions

- WinIsSOMDDReady
- WinRestartSOMDD
- WinRestartWPDServer

Example Code

This example stops the Workplace Shell DSOM Server, then waits until the server has terminate before stopping the DSOM daemon.

```
#define  INCL_WPCLASS
#include <os2.h>

ULONG  count=0;
enum   {ON, OFF};

WinRestartWPDServer(OFF);

/* Make sure the server thread has terminated completely before */
/* bring down the DSOM daemon                                   */
while ( WinIsWPDServerReady() )
{
   HEV hev;

   /* First create a private, reset, event semaphore.           */
   DosCreateEventSem( (PSZ)NULL, &hev, 0, FALSE);

   /* Wait for 1 second; then try again for a max. of 30 sec.   */
   DosWaitEventSem (hev, 1000);
   if (count++ > 30)
      break;
}
WinRestartSOMDD (OFF);
```

WinLoadAccelTable

This function loads an accelerator table.

Syntax

```
#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HACCEL WinLoadAccelTable  (HAB hab, HMODULE Resource,
                           ULONG idAccelTable)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Resource (HMODULE) – input
Resource identity containing the accelerator table.

Module handle returned by the DosLoadModule or DosQueryModuleHandle functions referencing a dynamic link library containing the resource or NULLHANDLE for the application's module.

idAccelTable (ULONG) – input
Accelerator-table identifier, within the resource file.

It must be greater or equal to 0 and less or equal to 0xFFFF.

Returns

haccelAccel (HACCEL) – returns
Accelerator-table handle.

Possible returns from WinGetLastError

PMERR_RESOURCE_NOT_FOUND (0x100A)	The specified resource identity could not be found.

Remarks

This function returns a different value when called twice in succession with the same parameter values.

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

Related Functions

- WinCopyAccelTable
- WinCreateAccelTable
- WinDestroyAccelTable
- WinQueryAccelTable
- WinSetAccelTable
- WinTranslateAccel

Example Code

This example loads an accelerator-table, using the application defined accelerator id, from a resource using the resource handle returned by DosLoadModule or DosQueryModuleHandle. The returned table handle is then used by WinCopyAccelTable to copy the table into an in-memory accelerator table structure.

```
#define INCL_WINACCELERATORS    /* Window Accelerator Functions */
#define INCL_DOSMODULEMGR       /* Module Manager Functions     */
#include <os2.h>
#define ACCEL_ID 1

ULONG   ulCopied;       /* bytes copied                         */
HACCEL  hAccel;         /* Accelerator-table handle             */
ACCELTABLE  pacctAccelTable;/* Accelerator-table data area      */
ULONG   ulCopyMax;      /* Maximum data area size               */
ULONG   idAccelTable=ACCEL_ID;/* Accelerator-table identifier   */
HAB     hab;            /* anchor-block handle                  */
HMODULE hmodDLL;        /* resource module                      */
CHAR    LoadError[100]; /* object name buffer for DosLoad       */
ULONG   rc;             /* return code                          */

/* obtain resource handle */
rc = DosLoadModule(LoadError, sizeof(LoadError), "RES.DLL",
                   &hmodDLL);

if (rc == 0)
   hAccel = WinLoadAccelTable(hab, hmodDLL, idAccelTable);

ulCopyMax = sizeof(pacctAccelTable);
if (hAccel)
   ulCopied = WinCopyAccelTable(hAccel, &pacctAccelTable,
                                ulCopyMax);
```

WinLoadDlg

This function creates a dialog window from the dialog template *idDlg* in *hmod* and returns the dialog window handle.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
HWND WinLoadDlg (HWND hwndParent, HWND hwndOwner, PFNWP pfnDlgProc,
                 HMODULE hmod, ULONG idDlg, PVOID pCreateParams)
```

Parameters

hwndParent (HWND) – input
Parent-window handle of the created dialog window.

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

hwndOwner (HWND) – input
Requested owner-window handle of the created dialog window.

The actual owner window is calculated using the algorithm specified below.

pfnDlgProc (PFNWP) – input
Dialog procedure for the created dialog window.

hmod (HMODULE) – input
Resource identity containing the dialog template.

NULLHANDLE	Use the application's .EXE file.
Other	Module handle returned from the DosLoadModule or DosQueryModuleHandle functions.

idDlg (ULONG) – input
Dialog-template identity within the resource file.

It is also used as the identity of the created dialog window.

pCreateParams (PVOID) – input
Pointer to application-defined data area.

This is passed to the dialog procedure in the WM_INITDLG message.

This parameter MUST be a pointer rather than a long.

Returns

hwndDlg (HWND) – returns
Dialog-window handle.

NULL	Dialog window not created
Other	Dialog window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_INTEGER_ATOM (0x1016)	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME (0x1015)	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND (0x1017)	The specified atom name is not in the atom table.
PMERR_RESOURCE_NOT_FOUND (0x100A)	The specified resource identity could not be found.

Remarks

Unless window style WS_VISIBLE is specified for the dialog window in the DIALOG statement within the dialog template, the dialog window is created as an invisible window.

The dialog window owner may be modified, in order to ensure acceptable results if it is later processed as a modal dialog using the WinProcessDlg or WinGetDlgMsg functions. A search is made up the parent hierarchy, starting at the window specified by the *hwndOwner* parameter, until a child of the window specified by the *hwndParent* is found. If such a window exists, it is made the actual owner of the dialog. If no such window exists the actual owner of the dialog is set to NULLHANDLE.

This function returns immediately after creating the dialog window. A WM_INITDLG (Default Dialogs) message is sent to the dialog procedure before this function returns.

This function should not be used while pointing device capture is set (see WinSetCapture).

As each of the controls defined within the template of this dialog window is created during the processing of this function, the dialog procedure may receive various control notifications before this function returns.

A dialog window can be destroyed with the WinDestroyWindow function.

Because windows are created from the template, strings in the template are processed with WinSubstituteStrings. Any resultant WM_SUBSTITUTESTRING messages are sent to the dialog procedure before this function returns.

When the child windows of the dialog are created, the WinSubstituteStrings function is used to allow the child windows to perform text substitutions in their window text. If any of the child window text strings contain the percent (%) substitution character, there is an upper limit of 256 on the length of the text string, after it is returned from the substitution.

If a dialog template (typically compiled using the resource compiler) references another resource (for example an icon resource for an icon static control), this function always searches for that resource in the .EXE file. If an application wishes to keep resources referenced by a dialog template in a .DLL library, these resources must be loaded by an explicit function call during the processing of the WM_INITDLG message.

Note: In general, it is better to create the dialog window invisible as this allows for optimization. In particular, an experienced user can type ahead, anticipating the processing in the dialog window.

In this instance, there may be no need to display the dialog window at all, as the user might have finished the interaction before the window can be displayed.

This is in fact how the WinProcessDlg function works; it does not display the dialog window while there are still WM_CHAR messages in the input queue, but allows these to be processed first.

Related Functions

- WinCreateDlg
- WinDefDlgProc
- WinDismissDlg
- WinDlgBox
- WinGetDlgMsg
- WinProcessDlg

Related Messages

- WM_INITDLG (Default Dialogs)
- WM_SUBSTITUTESTRING
- WM_CHAR

Example Code

This example uses WinLoadDlg to load a dialog template from the application's .EXE file.

```
#define INCL_WINDIALOGS     /* Window Dialog Mgr Functions  */
#include <os2.h>

HWND  hwndOwner;               /* owner window                */
HWND  hwndDlg;                 /* Dialog-window handle        */
PFNWP GenericDlgProc;          /* dialog process              */

hwndDlg = WinLoadDlg(HWND_DESKTOP,        /* parent is desk top */
                  hwndOwner,             /* owner is main frame */
                  GenericDlgProc,           /* dialog procedure */
                  0L,                 /* load from resource file */
                  DLG_ID,                /* dialog resource id */
                  NULL);                /* no dialog parameters */
```

WinLoadFileIcon

This function returns a pointer to an icon which is associated with the file specified by *pFileName*.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
HPOINTER WinLoadFileIcon  (PSZ pFileName, BOOL fPrivate)
```

Parameters

pFileName (PSZ) – input
Pointer to file name.

A pointer to a zero-terminated string which contains the name of the file whose icon will be loaded.

fPrivate (BOOL) – input
Icon usage flag.

TRUE A private copy of this icon is requested. This flag should be used if the application needs to modify the icon.

FALSE A shared pointer to this icon is requested. This flag should be used if application needs to display the icon without modifying it. This should be used whenever possible to optimize system resource use.

Returns

rc (HPOINTER) – returns
Success indicator.

NULL Error occurred.
OTHER Handle to an icon.

Remarks

The icon will be retrieved in the following order until an icon has been found:

- .ICON extended attribute
- .ICO file in same directory with same prefix
- Application specific icon (if PM executable or MS Windows executable*)
- PM application icon (if PM executable)
- MS Windows application icon (if MS Windows application executable)*
- OS/2 application icon (if OS/2 full-screen only executable)
- OS/2 window icon (if OS/2 window compatible executable)
- DOS windowed application icon (if DOS windowed executable)

- Program application (if unknown type executable)
- Data icon specified by associated application
- Data icon of associated application
- Data file icon (if not program or directory)
- Directory icon (if directory)

The HPOINTER returned in *fPrivate* should be freed by the caller via WinFreeFileIcon when it is no longer being used.

Related Functions

- WinSetFileIcon
- WinFreeFileIcon

WinLoadHelpTable

This function identifies the module handle and identity of the help table to the instance of the Help Manager.

Syntax

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinLoadHelpTable (HWND hwndHelpInstance, ULONG idHelpTable,
                       HMODULE Module)
```

Parameters

hwndHelpInstance (HWND) – input
Handle of an instance of the Help Manager.

This is the handle returned by the WinCreateHelpInstance call.

idHelpTable (ULONG) – input
Identity of the help table.

It must be greater or equal to 0 and less or equal to 0xFFFF.

Module (HMODULE) – input
Handle of the module which contains the help table and help subtable resources.

NULLHANDLE	Use the resources file for the application.
Other	The module handle returned by the DosLoadModule or DosQueryModuleHandle call referencing a dynamic-link library containing the help resources.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

An application specifies or changes the handle of the module which contains the help table or the identity of the help table.

This function corresponds to the HM_LOAD_HELP_TABLE message that identifies the identifier of a help table and the handle of the module which contains the help table and its associated help subtables.

Related Functions

- WinAssociateHelpInstance
- WinCreateHelpInstance
- WinCreateHelpTable
- WinDestroyHelpInstance
- WinQueryHelpInstance

Related Messages

- HM_LOAD_HELP_TABLE

Example Code

```
BOOL LoadHelpTable( HWND hWnd, USHORT usResource, PSZ pszModuleName )
{
    BOOL bSuccess = FALSE;
    HMODULE hmodule;
    HWND hwndHelp;
    PSZ  pszObjNameBuf[ 80 ];

    /* Get the DLL loaded */
    if( !DosLoadModule( pszObjNameBuf, sizeof( pszObjNameBuf ),
                  pszModuleName, &hmodule ) )
    {
        /* Get the associated help instance */
        hwndHelp = WinQueryHelpInstance( hWnd );

        if( hwndHelp )
        {
            /* Pass address of help table to the Help Manager */
            bSuccess = WinLoadHelpTable( hwndHelp, usResource, hmodule );
        }
    }

    /* Return success indicator */
    return bSuccess;
}
```

WinLoadLibrary

This function makes the library available to the application.

Syntax

```
#define INCL_WINLOAD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HLIB WinLoadLibrary  (HAB hab, PSZ pszLibname)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pszLibname (PSZ) – input
Library name.

Returns

hlibLibhandle (HLIB) – returns
Library handle.

NULLHANDLE	Library not successfully loaded
Other	Library handle.

Remarks

This function makes the library *pszLibname* (containing procedures, or resources, or both) available to the application. All of the dynamic link libraries have the .DLL filename extension by default.

Related Functions

- WinDeleteLibrary
- WinDeleteProcedure
- WinLoadProcedure

Example Code

This example loads the RES.DLL resource/procedure library, returning a library handle that is then used by WinLoadProcedure to load procedures from that library.

```
#define INCL_WINLOAD            /* Window Load Functions        */
#include <os2.h>

PFNWP   pWndproc;       /* procedure pointer                    */
HAB     hab;            /* anchor-block handle                  */
HLIB    hlib;           /* library handle                       */
char    pszLibname[10]="RES.DLL"; /* library name string        */
char    pszProcname[10]="WndProc"; /* procedure name string     */

/* load RES.DLL */
hlib = WinLoadLibrary(hab, pszLibname);

/* load WndProc */
pWndproc = WinLoadProcedure(hab, hlib, pszProcname);
```

WinLoadMenu

This function creates a menu window from the menu template *idMenu* from *hmod*, and returns in *hwndMenu* the window handle for the created window.

Syntax

```
#define INCL_WINMENUS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinLoadMenu  (HWND hwndFrame, HMODULE hmod, ULONG idMenu)
```

Parameters

hwndFrame (HWND) – input
Owner- and parent-window handle.

HWND_DESKTOP	The desktop window
HWND_OBJECT	Object window
Other	Specified window.

hmod (HMODULE) – input
Resource identifier.

NULLHANDLE	The resource is in the .EXE file of the application.
Other	The module handle returned by the DosLoadModule or DosQueryModuleHandle call.

idMenu (ULONG) – input
Menu identifier within the resource file.

It must be greater or equal to 0 and less or equal to 0xFFFF.

Returns

hwndMenu (HWND) – returns
Menu-window handle.

Remarks

The menu window is created with its parent and owner set to *hwndFrame*, and with identity FID_MENU. If *hwndFrame* is HWND_OBJECT or a window handle returned from WinQueryObjectWindow, the menu window is created as an object window.

Action bar menus are created as child windows of the frame window and are initially visible. Submenus are initially created as object windows that are owned by the window frame.

Related Functions

- WinCreateMenu
- WinPopupMenu

Example Code

This example creates a menu window from the menu template (idMenuId) located in "RES.DLL" and returns a menu handle which is used by WinPopupMenu.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINMENUS           /* Window Menu Functions        */
#include <os2.h>

HWND    hwndMenu;       /* menu window                          */
HWND    hwndOwner;      /* owner window                         */
HMODULE hmodDLL;        /* resource handle                      */
ULONG   idMenuid;       /* resource menu id                     */
BOOL  fSuccess;         /* success indicator                    */
HWND    hwndParent;     /* parent window                        */
ULONG   flOptions;      /* pop-up menu options                  */

if (DosQueryModuleHandle("RES.DLL",&hmodDLL))
   hwndMenu = WinLoadMenu(hwndOwner, hmodDLL, idMenuid);

flOptions = PU_MOUSEBUTTON1DOWN | PU_KEYBOARD | PU_MOUSEBUTTON1;
fSuccess = WinPopupMenu(hwndParent, hwndOwner, hwndMenu, 0, 50,
                        0, flOptions);
```

WinLoadMessage

This function loads a message from a resource, copies the message to the specified buffer, and appends a terminating null character.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinLoadMessage  (HAB hab, HMODULE hmodMod, ULONG ulId,
                      LONG lcchMax, PSZ pBuffer)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hmodMod (HMODULE) – input
Module handle.

NULLHANDLE	Use the application's own resources file.
Other	Module handle returned by the DosLoadModule or DosQueryModuleHandle call referencing a dynamic-link library containing the resource.

ulId (ULONG) – input
Message identifier.

It must be greater or equal to 0 and less or equal to 0xFFFF.

lcchMax (LONG) – input
Specifies the size of buffer.

The maximum length of a string is 256 character.

pBuffer (PSZ) – input
Points to the buffer that receives the message.

Returns

lLength (LONG) – returns
The length of the string returned.

This excludes the terminating null, and has the following values:

0	Error
Other	A maximum value of (*lcchMax*–1).

Remarks

Message resources contain up to 16 messages each. The resource ID is calculated from the id parameter value passed to this function as follows:

```
resource ID = (id / 16) + 1
```

To save storage on disk and in memory, applications should number their message resources sequentially, starting at some multiple of 16.

Related Functions

- WinLoadString

Example Code

This example loads an error message from ERR.DLL using the resource handle from DosLoadModule and uses the message in a message box.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_DOSMODULEMGR       /* Module Manager Functions     */
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#include <os2.h>
#define ERRMSG_ID 1

LONG    lLength;        /* length of string                     */
HAB     hab;            /* anchor-block handle                  */
HMODULE hmodDLL;        /* Handle of resource module            */
LONG    lBufferMax = 100;/* Size of buffer                      */
char    pszErrMsg[100]; /* error message                        */
CHAR    LoadError[100]; /* object name buffer for DosLoad       */
ULONG   rc;             /* return code                          */
HWND    hwnd;           /* window handle                        */

/* obtain resource handle */
rc = DosLoadModule(LoadError, sizeof(LoadError), "ERR.DLL",
                   &hmodDLL);

/* load message from resource */
if (rc == 0)
   {
   /* load error message string */
   lLength = WinLoadMessage(hab, hmodDLL, ERRMSG_ID, lBufferMax,
                            pszErrMsg);

   /* display error message box */
   WinMessageBox(HWND_DESKTOP,
       hwnd,                    /* client-window handle   */
       pszErrMsg,               /* message                */
       "Error message",         /* title of the message   */
       0,                       /* message box id         */
       MB_NOICON | MB_OK);      /* icon and button flags  */
   }
```

WinLoadPointer

This function loads a pointer from a resource file into the system.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HPOINTER WinLoadPointer  (HWND hwndDeskTop, HMODULE Resource,
                          ULONG idPointer)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop window
Other	Desktop-window handle returned by WinQueryDesktopWindow.

Resource (HMODULE) – input
Resource identity containing the pointer definition.

NULLHANDLE	Use the resources file for the application.
Other	Module handle returned by the DosLoadModule or DosQueryModuleHandle call referencing a dynamic-link library containing the resource.

idPointer (ULONG) – input
Identifier of the pointer to be loaded.

It must be greater or equal to 0 and less or equal to 0xFFFF.

Returns

hptr (HPOINTER) – returns
Pointer handle.

NULLHANDLE	Error has occurred
Other	Handle of loaded pointer.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_RESOURCE_NOT_FOUND (0x100A)	The specified resource identity could not be found.

Remarks

A new copy of the pointer is created each time this function is called. The pointer created by this function can be destroyed using the WinDestroyPointer function. To get one of the standard system pointers, use the WinQuerySysPointer function.

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

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinQuerySysPointerData
- WinSetPointer
- WinSetPointerPos
- WinSetSysPointerData
- WinShowPointer

Example Code

This example calls WinLoadPointer to load an application defined pointer. When processing the WM_MOUSEMOVE message, the loaded pointer is displayed by calling WinSetPointer.

```
#define INCL_WINPOINTERS        /* Window Pointer Functions     */
#include <os2.h>

HPOINTER  hptrCrossHair;/* pointer handle                       */

case WM_CREATE:
    hptrCrossHair = WinLoadPointer(HWND_DESKTOP,
        0L,                /* load from .exe file    */
        IDP_CROSSHAIR);    /* identifies the pointer */

case WM_MOUSEMOVE:
    WinSetPointer(HWND_DESKTOP, hptrCrossHair);
```

WinLoadProcedure

This function loads the window or dialog procedure from a specified dynamic link library.

Syntax

```
#define INCL_WINLOAD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
PFNWP WinLoadProcedure  (HAB hab, HLIB hlibLibhandle, PSZ pszProcname)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hlibLibhandle (HLIB) – input
Library handle.

pszProcname (PSZ) – input
Procedure name.

Returns

Wndproc (PFNWP) – returns
Window-procedure identifier.

NULL	Procedure not successfully loaded
Other	Window-procedure identifier.

Remarks

This function loads the window or dialog procedure *pszProcname* from the library *hlibLibhandle*.

Related Functions

- WinDeleteLibrary
- WinDeleteProcedure
- WinLoadLibrary

Example Code

This example loads the WndProc procedure, returning a pointer to the procedure, from the RES.DLL library, based on the library handle returned by WinLoadLibrary.

```
#define INCL_WINLOAD            /* Window Load Functions        */
#include <os2.h>

PFNWP   pWndproc;       /* procedure pointer                    */
HAB     hab;            /* anchor-block handle                  */
HLIB    hlib;           /* library handle                       */
char    pszLibname[10]="RES.DLL"; /* library name string        */
char    pszProcname[10]="WndProc"; /* procedure name string     */

/* load RES.DLL */
hlib = WinLoadLibrary(hab, pszLibname);

/* load WndProc */
pWndproc = WinLoadProcedure(hab, hlib, pszProcname);
```

WinLoadString

This function loads a string from a resource.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinLoadString  (HAB hab, HMODULE Resource, ULONG idString,
                     LONG lBufferMax, PSZ pszBuffer)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Resource (HMODULE) – input
Resource identity containing the string.

NULLHANDLE	Use the application's own resources file.
Other	Module handle returned by the DosLoadModule or DosQueryModuleHandle call referencing a dynamic-link library containing the resource.

idString (ULONG) – input
String identifier.

It must be greater or equal to 0 and less or equal to 0xFFFF.

lBufferMax (LONG) – input
Size of buffer.

The maximum length of a string is 256 characters.

pszBuffer (PSZ) – output
Buffer that is to receive the string.

Returns

lLength (LONG) – returns
The length of the string returned.

This excludes the terminating null, and has the following values:

0	Error
Other	A maximum value of (*lBufferMax*–1).

Possible returns from WinGetLastError

PMERR_RESOURCE_NOT_FOUND (0x100A) The specified resource identity could not be found.

Remarks

This function loads a string resource identified by the *idString* and the *Resource* parameters into the *pszBuffer* parameter, and appends a terminating null character.

RT_STRING resources (string resources) contain up to 16 strings each (see "Resource (.RES) File Specification" in the *Presentation Manager Programming Reference Volume II*). The resource ID is calculated from the idString passed to this function as follows:

```
resource ID = (idString / 16) + 1
```

To save storage on disk and in memory, applications should number their string resources sequentially, starting at some multiple of 16.

Related Functions

- WinCompareStrings
- WinNextChar
- WinPrevChar
- WinSubstituteStrings
- WinUpper
- WinUpperChar

Example Code

This example loads a string from RES.DLL using the resource handle from DosLoadModule.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_DOSMODULEMGR       /* Module Manager Functions     */
#include <os2.h>
#define STRING_ID 1

LONG    lLength;        /* length of string                     */
HAB     hab;            /* anchor-block handle                  */
HMODULE hmodDLL;        /* Handle of resource module            */
ULONG   idString = STRING_ID; /* String identifier              */
LONG    lBufferMax = 10;/* Size of buffer                       */
char    pszString1[10]; /* first string                         */
CHAR    LoadError[100]; /* object name buffer for DosLoad       */
ULONG   rc;             /* return code                          */

/* obtain resource handle */
rc = DosLoadModule(LoadError, sizeof(LoadError), "RES.DLL",
                   &hmodDLL);

/* load string from resource */
if (rc == 0)
   lLength = WinLoadString(hab, hmodDLL, idString, lBufferMax,
                           pszString1);
```

WinLockPointerUpdate

This function is specific to version 2.1, or higher, of the OS/2 operating system.

This function causes the mouse pointer to change into the symbol described by *hptrNew* for the period of time indicated by *ulTimeInterval*.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinLockPointerUpdate (HWND hwndDesktop, HPOINTER hptrNew,
                           ULONG ulTimeInterval)
```

Parameters

hwndDesktop (HWND) – input
Handle to the desktop window.

hptrNew (HPOINTER) – input
Pointer handle to be displayed.

ulTimeInterval (ULONG) – input
Time interval.

The time (0 to 10000ms) during which changes to the mouse pointer shape will be locked out.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful.
FALSE	An error occurred.

Remarks

This function causes the mouse pointer to be changed into the symbol described by *hptrNew* for the period of time indicated by *ulTimeInterval*. The mouse remains fully functional during the lock time, however the pointer shape is not updated.

This function can be used to convey visual feedback to the user, that some mouse action has been recognized. For example, that a gesture has been recognized from the last few mouse movements.

WinLockupSystem

This function is specific to version 2.1, or higher, of the OS/2 operating system.

This function locks up the system.

Syntax

```
#define INCL_WINMESSAGEMGR
#include <os2.h>
BOOL WinLockupSystem  (HAB hab)
```

Parameters

hab (HAB) – input
The application anchor block.

Returns

rc (BOOL) – returns
Return code.

TRUE	The system was successfully locked.
FALSE	An error occurred or the system was already in a locked state.

Remarks

This function allows an application program to cause the system to lock up at any point in time. For example, a programmer might wish to lock up the system if any tampering is detected, such as a password being misspelled more than four times in a row.

In order to execute WinUnlockSystem after a WinLockupSystem has been called, the WinUnlockSystem must be called from a separate thread because WinLockupSystem will not return until a password has been entered from the keyboard.

The LockupHook hook allows a PM application to customize system lockups.

In order for the window to appear as the top most window on the lockup screen, the WS_CLIPSIBLINGS flag must be used for the *flStyle* parameter in the WinCreateWindow or WinCreateStdWindow call.

Related Functions

- LockupHook
- WinUnlockSystem

WinLockVisRegions

This function locks or unlocks the visible regions of all the windows on the screen, preventing any of the visible regions from changing.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinLockVisRegions  (HWND hwndDesktop, BOOL fLock)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle or HWND_DESKTOP.

fLock (BOOL) – input
Indicates whether the visible regions are being locked or unlocked.

TRUE	Lock the visible regions
FALSE	Unlock the visible regions.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful.
FALSE	An error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function is useful to threads that need to prevent window visible regions from changing while some screen operation, such as copying screen pels into a memory bit map, is being performed.

Any other thread that tries to alter the visible regions is blocked while the visible regions are locked. While the visible regions are locked, no messages must be sent and no functions called that can send messages.

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.

Note: Locking the visible regions does not prevent painting of a window by another process.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Example Code

This example uses WinLockVisRegions to prevent any window's visible region from changing while a screen operation is executing. WinLockVisRegions is called before the screen operation to lock the visible regions and again after the operation to unlock the regions.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

BOOL  fSuccess;          /* success indicator                   */

/* lock visible regions */
fSuccess = WinLockVisRegions(HWND_DESKTOP, TRUE);

/*
 .
 . executing screen operation
 .
 */

/* unlock visible regions */
fSuccess = WinLockVisRegions(HWND_DESKTOP, FALSE);
```

WinLockWindowUpdate

This function disables or enables output to a window and its descendants.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinLockWindowUpdate (HWND hwndDeskTop, HWND hwndLockUpdate)
```

Parameters

hwndDeskTop (HWND) – input
Desktop handle of the screen containing the window to be locked.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

hwndLockUpdate (HWND) – input
Handle of window in which output is to be prevented.

NULLHANDLE	Enable output in the locked window and its descendants.
Other	Handle of the window in which output is to be prevented. Output is also prevented in the descendants of the window.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful operation.
FALSE	Error occurred.

Remarks

This function is used by threads that need to draw on an area of the screen over which they have no control. For example, the user interface sizing and moving calls use this call when drawing the shadow box, as a window is sized or moved.

All threads continue to run while the window is disabled; only output is prevented.

If one thread disables the window, other threads using this function are blocked until the first enables the window, although they can still receive messages.

This function does not prevent screen group switches, because these may be necessary to handle "hard errors" in other screen groups.

Example Code

This example disables output to a window and its children during a move operation (WM_MOVE) and then re-enables output once the move is finished.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND  hwndLock;          /* handle of window to be (un)locked    */
BOOL  fSuccess;          /* success indicator                    */

case WM_MOVE:
     /* lock output */
     fSuccess = WinLockWindowUpdate(HWND_DESKTOP, hwndLock);

     /*
      .
      . execute window move
      .
      */

     /* unlock output */
     fSuccess = WinLockWindowUpdate(HWND_DESKTOP, NULLHANDLE);
```

WinMakePoints

This function converts points to graphics points.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinMakePoints  (HAB hab, PPOINTL pwpt, ULONG cwpt)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pwpt (PPOINTL) – in/out
Points to be converted.

The data type of these points after conversion is POINTL.

cwpt (ULONG) – input
Number of points to be converted.

Must be positive.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function converts the array of points from a WPOINT data structure into a POINTL data structure.

Example Code

This example calls WinMakePoints to convert a 3-element array of points from window points (WPOINT structure) to graphics points (POINTL structure).

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions   */
#include <os2.h>

HAB     hab;            /* anchor-block handle                  */
BOOL  fSuccess;         /* success indicator                    */
/* array of window points */
WPOINT pwptppt[3] = {0,0,0,0,20,0,50,0,100,0,60,0};

/* convert points */
fSuccess = WinMakePoints(hab, pwptppt, 3);
```

WinMakeRect

This function converts a rectangle to a graphics rectangle.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinMakeRect (HAB hab, PRECTL pwrc)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pwrc (PRECTL) – in/out
Rectangle to be converted.

The data type of the rectangle after conversion is RECTL.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function converts a rectangle from a WRECT data structure into a RECTL data structure.

Related Functions

- WinCopyRect
- WinEqualRect
- WinFillRect
- WinInflateRect
- WinIntersectRect
- WinIsRectEmpty
- WinOffsetRect
- WinPtInRect
- WinSetRect
- WinSetRectEmpty
- WinSubtractRect
- WinUnionRect

Example Code

This example calls WinMakeRect to convert a window rectangle (WRECT structure) to a graphics rectangle (RECTL structure).

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions   */
#include <os2.h>

HAB     hab;            /* anchor-block handle                  */
BOOL  fSuccess;         /* success indicator                    */
/* window rectangle */
WRECT  pwrcprc = {0,0,0,0,50,0,50,0};

/* convert rectangle */
fSuccess = WinMakeRect(hab, &pwrcprc);
```

WinMapDlgPoints

This function maps points from dialog coordinates to window coordinates, or from window coordinates to dialog coordinates.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinMapDlgPoints  (HWND hwndDlg, PPOINTL prgwptl, ULONG cwpt,
                       BOOL fCalcWindowCoords)
```

Parameters

hwndDlg (HWND) – input
Dialog-window handle.

prgwptl (PPOINTL) – in/out
Coordinate points to be mapped.

The mapped points are substituted.

cwpt (ULONG) – input
Number of coordinate points.

fCalcWindowCoords (BOOL) – input
Calculation control.

TRUE The points are in dialog coordinates and are to be mapped into window coordinates relative to the window specified by the *hwndDlg* parameter.

FALSE The points are in window coordinates relative to the window specified by the *hwndDlg* parameter and are to be mapped into dialog coordinates.

Returns

rc (BOOL) – returns
Coordinates-mapped indicator.

TRUE Coordinates successfully mapped
FALSE Coordinates not successfully mapped.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001) An invalid window handle was specified.

Related Functions

- WinMapWindowPoints

Example Code

This example calls WinMapDlgPoints to map a point from dialog coordinates to window coordinates relative to the dialog window.

```
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#include <os2.h>

HWND  hwndDlg;          /* handle of dialog window              */
BOOL  fSuccess;         /* success indicator                    */
POINTL aptlPoint;       /* point to be mapped                   */

/* map point to relative window coordinates */
fSuccess = WinMapDlgPoints(hwndDlg, &aptlPoint, 1, TRUE);
```

WinMapWindowPoints

This function maps a set of points from a coordinate space relative to one window into a coordinate space relative to another window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinMapWindowPoints  (HWND hwndFrom, HWND hwndTo,
                          PPOINTL prgptl, LONG cwpt)
```

Parameters

hwndFrom (HWND) – input
Handle of the window from whose coordinates points are to be mapped.

HWND_DESKTOP	Points are mapped from screen coordinates
Other	Points are mapped from window coordinates.

hwndTo (HWND) – input
Handle of the window to whose coordinates points are to be mapped.

HWND_DESKTOP	Points are mapped into screen coordinates
Other	Points are mapped into window coordinates.

prgptl (PPOINTL) – in/out
Points to be mapped to the new coordinate system.

cwpt (LONG) – input
Number of points to be mapped.

prgptl can be a RECTL structure, in which case this parameter should have the value 2.

Note: This is not supported in all languages.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinMapDlgPoints

Example Code

This example calls WinMapWindowPoints to map a mouse point on the desktop window to a mouse point in the client window and then checks whether the mouse pointer is inside the client area or not.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINRECTANGLES      /* Window Rectangle Functions   */
#define INCL_WINPOINTERS        /* Window Pointer Functions     */
#include <os2.h>

HAB     hab;            /* anchor-block handle                  */
HWND  hwndClient;       /* handle of client window              */
BOOL  fSuccess;         /* success indicator                    */
POINTL ptlMouse;        /* mouse pointer position               */
RECTL rclWork;          /* client area                          */

/* get current mouse pointer position */
WinQueryPointerPos(HWND_DESKTOP, &ptlMouse);

/* map from desktop to client window */
fSuccess = WinMapWindowPoints(HWND_DESKTOP, hwndClient,
                              &ptlMouse, 1);

/* check if new mouse position is inside the client area */
WinQueryWindowRect(hwndClient, &rclWork);
if (WinPtInRect(hab, &rclWork, &ptlMouse))
   {
   /* pointer is in client area */
   }
```

WinMessageBox

This function creates, displays, and operates a message box window.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
ULONG WinMessageBox (HWND hwndParent, HWND hwndOwner, PSZ pszText,
                     PSZ pszCaption, ULONG idWindow, ULONG flStyle)
```

Parameters

hwndParent (HWND) – input
Parent-window handle of the created message-box window.

HWND_DESKTOP	The message box is to be main window.
Other	Parent-window handle.

hwndOwner (HWND) – input
Requested owner-window handle of the created message-box window.

The actual owner window is calculated using the algorithm specified in the description of the WinLoadDlg function.

pszText (PSZ) – input
Message-box window message.

The text of the message to be displayed within the message-box window. If multiple lines are required, carriage-return characters must be inserted into the text at appropriate points.

pszCaption (PSZ) – input
Message-box window title.

The text for the title should not be longer than 40 characters. If text longer than this is supplied, text centering is still performed, even though the beginning and end of the string are not visible.

NULL	The text Error is to be displayed as the title of the message-box window.
Other	The text to be displayed as the title of the message-box window.

idWindow (ULONG) – input
Message-box window identity.

This value is passed to the HK_HELP hook if the WM_HELP message is received by the message-box window. It must be greater or equal to 0 and less or equal to 0xFFFF.

flStyle (ULONG) – input
Message-box window style.

These values may be combined using the logical-OR operation but only one value can be taken from each of the following groups.

Button or Button Group:

MB_OK	Message box contains an OK push button.
MB_OKCANCEL	Message box contains both OK and CANCEL push buttons.
MB_CANCEL	Message box contains a CANCEL push button.
MB_ENTER	Message box contains an ENTER push button.
MB_ENTERCANCEL	Message box contains both ENTER and CANCEL push buttons.
MB_RETRYCANCEL	Message box contains both RETRY and CANCEL push buttons.
MB_ABORTRETRYIGNORE	Message box contains ABORT, RETRY, and IGNORE push buttons.
MB_YESNO	Message box contains both YES and NO push buttons.
MB_YESNOCANCEL	Message box contains YES, NO, and CANCEL push buttons.

Help button:

MB_HELP'.	Message box contains a HELP push button. When this is selected a WM_HELP message is sent to the window procedure of the message box.

Color or Icon:

MB_NOICON	Message box is not to contain an icon.
MB_ICONHAND	Message box contains a hand icon.
MB_ICONQUESTION	Message box contains a question mark (?) icon.
MB_ICONEXCLAMATION	Message box contains an exclamation point (!) icon.
MB_ICONASTERISK	Message box contains an asterisk (*) icon.
MB_INFORMATION	Message box contains a black information "i" in a square box.
MB_QUERY	Message box contains a question mark in a square box.
MB_WARNING	Message box contains a black '!' in a square box.
MB_ERROR	Message box contains a STOP sign on a white background.

Default action:

MB_DEFBUTTON1	The first button is the default selection. This is the default case, if none of MB_DEFBUTTON1, MB_DEFBUTTON2, and MB_DEFBUTTON3 is specified.

MB_DEFBUTTON2	The second button is the default selection.
MB_DEFBUTTON3	The third button is the default selection.
Modality indicator:	
MB_APPLMODAL	Message box is application modal. This is the default case. Its owner is disabled; therefore, do not specify the owner as the parent if this option is used.
MB_SYSTEMMODAL	Message box is system modal.
Mobility indicator:	
MB_MOVEABLE	Message box is moveable. The message box is displayed with a title bar and a system menu, which shows only the Move, Close, and Task Manager choices, which can be selected either by use of the pointing device or by accelerator keys. If the user selects Close, the message box is removed and the *usResponse* is set to MBID_CANCEL, whether or not a cancel button existed within the message box.

Returns

usResponse (ULONG) – returns
User-response value.

MBID_ENTER	ENTER push button was selected
MBID_OK	OK push button was selected
MBID_CANCEL	CANCEL push button was selected
MBID_ABORT	ABORT push button was selected
MBID_RETRY	RETRY push button was selected
MBID_IGNORE	IGNORE push button was selected
MBID_YES	YES push button was selected
MBID_NO	NO push button was selected
MBID_ERROR	Function not successful; an error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

The message box consists of a message and a simple dialog with the user.

This function behaves in a similar way to WinDlgBox, and the remarks concerning modality which are documented under that call, and also under the WinLoadDlg and WinProcessDlg functions, also apply here.

This function should not be used while pointing device capture is set (see WinSetCapture).

If the keyboard is used to cycle from one window to the next, the message box and its parent window are considered to be next to each other in the sequence.

If a message box is created as part of the processing of a dialog window, where the dialog window has not been dismissed, the dialog window should be made the owner of the message box window.

If a system modal message box is created to indicate to the user that the system is running out of memory, the strings passed into this call must not be taken from a resource file, as an attempt to load the resource file could fail because of the lack of memory. However, such a message box can safely use the hand icon because this icon is always memory-resident.

The size of the message box is determined as follows:

- The minimum width of a message box is enough to display 40 characters of average width.
- The minimum height of a message box is enough to display 2 lines.
- The text of a message box is word-wrapped by default. If more than two lines are required to display the text, the height of the message box is increased up to a maximum of two thirds of the screen height. The height of a message box can never exceed this value.
- If necessary, the width of a message box is increased to allow room to display the title.

Text is wrapped at word boundaries (spaces). If a word is too big to fit on one line, the start of the word is not wrapped to the next line, but stays adjacent to the text it follows, and the word is split at the box boundary.

The message box is centered on the screen.

If a message box window has a CANCEL button, the MBID_CANCEL value is returned if either the Escape key is pressed or the Cancel button is selected. If the message box window has no CANCEL button, pressing the Escape key has no effect.

Note: If a dialog box or a message box is up for a window, and the parent or owner of that window is destroyed, the code following the WinDlgBox or WinMessageBox call is executed even though the parent/owner window no longer exists. This can result in accessing data that no longer exists; especially data referenced in the window words. Therefore, it is extremely important to determine the state of your child-window procedure after this function returns. The most straightforward method for doing this is to call WinQueryWindowPtr to get a pointer to the window words. If the returned

pointer is NULL, then you should exit immediately. Should this be the case, the bottom-up rule (that is, the child window gets WM_DESTROY messages first, then the parent window) still applies, and it becomes the child window procedure's responsibility to exit gracefully.

Related Functions

- WinAlarm
- WinFlashWindow

Example Code

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.

```
#define INCL_WINDIALOGS         /* Window Dialog Mgr Functions  */
#define INCL_WINPOINTERS        /* Window Pointer Functions     */
#include <os2.h>

CHAR szMsg[100];        /* message                              */
POINTL ptl;             /* message data                         */
HWND   hwndClient;      /* client window handle                 */

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 */
```

WinMessageBox2

This function creates a message-box window that can be used to display error messages and ask questions.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_GPI, INCL_WINWINDOWMGR, */
#include <os2.h>
ULONG WinMessageBox2 (HWND hwndParent, HWND hwndOwner, PSZ pszText,
                      PSZ pszTitle, ULONG ulWindow, PMB2INFO pmb2info)
```

Parameters

hwndParent (HWND) – input
Parent-window handle of the message-box window to be created.

HWND-DESKTOP	The message box is to be main window.
Other	Parent-window handle.

hwndOwner (HWND) – input
Requested owner-window handle of the message-box window to be created.

The actual owner window is calculated using the algorithm specified in the description of the WinLoadDlg function.

pszText (PSZ) – input
Message-box window message.

The text of the message to be displayed within the message-box window. If multiple lines are required, carriage-return characters must be inserted into the text at appropriate points.

pszTitle (PSZ) – input
Message-box window title.

The text for the title should not be longer than 40 characters. If text longer than this is supplied, text centering is still performed, even though the beginning and end of the string are not visible.

A value if NULL indicates that the text "Error" is to be displayed as the title of the message-box window.

ulWindow (ULONG) – input
Message-box window identity.

This value is passed to the HK_HELP hook if the WM_HELP message is received by the message-box window.

pmb2info (PMB2INFO) – input
Input structure for mesage-box window.

Returns

ulButtonId (ULONG) – returns
Id of the button that was clicked, or MBID_ERROR.

Possible returns from WinGetLastError

PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.
PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The function is a more powerful version of WinMessageBox. It supports custom icons and custom text within buttons, and can be transformed into a non-modal message box. It can also be displayed with a title bar and a system menu containing Move, Size, Close and Window items.

Related Functions

- WinAlarm
- WinFlashWindow
- WinMessageBox

Example Code

The following example uses WinMessageBox2 to create a message box containing a customized icon.

```
#define  INCL_WINDIALOGS     /* Window Dialog Mgr Functions        */
#define  INCL_WINPOINTERS    /* Window Pointer Functions           */

#include <os2.h>
#include <stdio.h>
#include <string.h>

CHAR      szMsg[100];        /* message                            */
HWND      hwndClient;        /* client-window handle               */
MB2INFO   mb2info;           /* message box input structure        */

MB2D mb2d[4] = {              /* Array of button defs     */
    { "AAAA", ID_BUTTON1, BS_DEFAULT},
    { "BBBBBBBBBBBB", ID_BUTTON2, 0},
    { "CCCCCCCCC",  ID_BUTTON3, 0},
    { "D", ID_BUTTON4, 0}
};

mb2info.hIcon = WinLoadPointer(HWND_DESKTOP, 0, ID_ICON1);
mb2info.cButtons = 4;                          /* number of buttons   */
mb2info.flStyle = MB_CUSTOMICON | MB_MOVEABLE; /* icon style flags    */
mb2info.hwndNotify = NULLHANDLE;               /* Reserved            */
mb2info.cb = sizeof(MB2INFO) + ((mb2info.cButons >1) ?
             (mb2info.cButtons -1) * sizeof (MB2D) : 0);

memcpy (&mb2info.mb2d, &mb2d, mb2info.cb);

sprintf (&szMsg, %s, "Error condition exists");

WinMessageBox2(HWND_DESKTOP,
               hwndClient,              /* client-window handle     */
               &szMsg,                  /* body of the message      */
               "Debugging Information", /* title of the message     */
               0,                       /* message box id           */
               &mb2info);            /* message box input structure */
```

WinMoveObject

This function is specific to version 3, or higher, of the OS/2 operating system.

This function moves an object from its existing folder to a specified new destination.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
HOBJECT WinMoveObject  (HOBJECT hObjectofObject, HOBJECT hObjectofDest,
                        ULONG ulFlags)
```

Parameters

hObjectofObject (HOBJECT) – input
Handle of the object being moved.

hObjectofDest (HOBJECT) – input
Handle of the folder into which *hObjectofObject* is to be moved.

ulFlags (ULONG) – input
Flags.

MOVE_FAILIFEXSTS

Returns

hwndDlg (HOBJECT) – returns
Handle of the newly created object.

A return value of NULLHANDLE indicates that either *hObjectofDest* is NULLHANDLE or an object with the same name as *hObjectofObject* exists in the destination folder.

Possible returns from WinGetLastError

WPERR_INVALID_FLAGS (0x1719) An invalid flag was specified.

Remarks

Using HOBJECT for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

Related Functions

- WinCopyObject
- WinCreateObject.
- WinDestroyObject
- WinQueryObjectWindow
- WinSaveObject

WinMultWindowFromIDs

This function finds the handles of child windows that belong to a specified window and have window identities within a specified range.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinMultWindowFromIDs (HWND hwndParent, PHWND prghwnd,
                           ULONG idFirst, ULONG idLast)
```

Parameters

hwndParent (HWND) – input
Parent-window handle.

prghwnd (PHWND) – output
Window handles.

Array of window handles. The array must contain (*idLast* – *idFirst* + 1) elements. The handle of a window, whose identity is WID (in the range *idFirst* to *idLast*), has a zero-based index in the array of (WID – *idFirst*). If there is no window for a window identity within the range, the corresponding element in the array is NULLHANDLE.

idFirst (ULONG) – input
First window identity value in the range (inclusive).

It must be greater or equal to 0 and less or equal to 0xFFFF.

idLast (ULONG) – input
Last window identity value in the range (inclusive).

It must be greater or equal to 0 and less or equal to 0xFFFF.

Returns

lWindows (LONG) – returns
Number of window handles returned.

0	No window handles returned
Other	Number of window handles returned.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function can be used to enumerate all the items in a dialog group, or to enumerate all the frame controls of a standard window. This function is faster than individual calls to the WinWindowFromID function.

Related Functions

- WinBeginEnumWindows
- WinEndEnumWindows
- WinEnumDlgItem
- WinGetNextWindow
- WinIsChild
- WinQueryWindow
- WinSetOwner
- WinSetParent

Example Code

This example finds the handles of all frame controls of a specified window via the WinMultWindowFromIDs call. The handles are returned in an array of window handles, and after the call completes, the handle for the minmax control window is assigned to a variable if a handle for it was found (i.e. handle not equal to NULLHANDLE).

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINFRAMEMGR        /* Window Frame Functions       */
#include <os2.h>

HWND  hwndParent;       /* parent window                        */
HWND  ahwnd[FID_CLIENT-FID_SYSMENU]; /* window handle array     */
HWND  hwndMinMax;       /* minmax control window handle         */
LONG  lHandles;         /* number of handles returned           */

/* get all control handles between and including system menu and
   client windows */
lHandles = WinMultWindowFromIDs(hwndParent, ahwnd, FID_SYSMENU,
                                FID_CLIENT);

/* if any handles returned and the handle for the minmax control is
   not null, assign a variable to the minmax handle */
if (lHandles > 0 && ahwnd[FID_MINMAX -
   FID_SYSMENU] != NULLHANDLE)
   hwndMinMax = ahwnd[FID_MINMAX - FID_SYSMENU];
```

WinNextChar

This function moves to the next character in a string.

Syntax

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
PSZ WinNextChar  (HAB hab, ULONG ulCodepage, ULONG ulCountry,
                  PSZ pszCurrentChar)
```

Parameters

hab (HAB) – input
Anchor-block handle.

ulCodepage (ULONG) – input
Code page.

If 0 is specified for *ulCodepage*, the code page specified by the queue associated with the calling thread is used.

ulCountry (ULONG) – input
Reserved value, must be 0.

pszCurrentChar (PSZ) – input
Current character in a null-terminated string.

Returns

pszNextChar (PSZ) – returns
Next character in the null-terminated string.

NULL	End of string reached
Other	Next character.

Possible returns from WinGetLastError

PMERR_INVALID_STRING_PARM (0x100B)	The specified string parameter is invalid.

Remarks

This function handles DBCS strings.

The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865
Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft Windows.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data.
The following is the list of valid country codes:

Country	Code
Arabic	785
Australian	61
Belgian	32
Canadian-French	2
Danish	45
Finnish	358

French	33
German	49
Hebrew	972
Italian	39
Japanese	81
Korean	82
Latin-American	3
Netherlands	31
Norwegian	47
Portuguese	351
Simpl. Chinese	86
Spanish	34
Swedish	46
Swiss	41
Trad. Chinese	88
UK-English	44
US-English	1
Other country	0

Related Functions

- WinCompareStrings
- WinLoadString
- WinPrevChar
- WinSubstituteStrings
- WinUpper
- WinUpperChar

Example Code

This example uses WinNextChar to traverse a string until a specified character is found, while maintaining an index to point to the character's position.

```
#define INCL_WINCOUNTRY          /* Window Country Functions     */
#include <os2.h>

HAB   hab;               /* anchor-block handle                  */
ULONG   idCodepage=437; /* Code page identity of both strings   */
ULONG   idCountryCode=1;/* Country code                         */
char  pszString1[10];    /* first string                         */
char  *pszNextChar;      /* next character                       */
char  *pszCurrentChar;   /* current character                    */
ULONG   ulIndex;         /* array index                          */

/* set string */
strcpy(pszString1,"Compare");

pszCurrentChar = pszString1;
do
   {
   pszNextChar = WinNextChar(hab, idCodepage, idCountryCode,
                             (psz)pszCurrentChar);

   if (pszCurrentChar[0] == 'p')
      break;

   ulIndex++;
   }
while (pszNextChar != NULL);
```

WinOffsetRect

This function offsets a rectangle.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinOffsetRect (HAB hab, PRECTL prcl, LONG cx, LONG cy)
```

Parameters

hab (HAB) – input
Anchor-block handle.

prcl (PRECTL) – in/out
Rectangle to be offset.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT may also be used, if supported by the language.

cx (LONG) – input
x-value of offset.

cy (LONG) – input
y-value of offset.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function offsets the coordinates of *prcl* by adding the value of the *cx* parameter to both the left and right coordinates, and the value of the *cy* to both the top and bottom coordinates.

Related Functions

- WinCopyRect
- WinEqualRect
- WinFillRect
- WinInflateRect
- WinIntersectRect
- WinIsRectEmpty
- WinPtInRect

- WinSetRect
- WinSetRectEmpty
- WinSubtractRect
- WinUnionRect

Example Code

This example moves a rectangle in response to the movement of the mouse (WM_MOUSEMOVE); the rectangle is moved (offset) based on the distance moved by the mouse since its previous position.

```
#define INCL_WINRECTANGLES      /* Window Rectangle Functions   */
#include <os2.h>

int main(void)
{
BOOL  fSuccess;         /* success indicator                    */
HAB     hab;            /* anchor-block handle                  */
RECTL prclRect1 = {0,0,100,100}; /* rectangle                   */
LONG   lcx;             /* Horizontal expansion                 */
LONG   lcy;             /* Vertical expansion                   */
POINTL ptlPrev;         /* previous mouse position              */
POINTL ptlCurr;         /* current mouse position               */
MPARAM mp1;             /* Parameter 1  (x,y) point value       */

case WM_MOUSEMOVE:
     ptlCurr.x = (LONG) SHORT1FROMMP(mp1);
     ptlCurr.y = (LONG) SHORT2FROMMP(mp1);

     /* calculate distance from previous mouse position */
     lcx =   (LONG)(ptlPrev.x - ptlCurr.x);
     lcy =   (LONG)(ptlPrev.y - ptlCurr.y);

     fSuccess = WinOffsetRect(hab, &prclRect1, lcx, lcy);
```

WinOpenClipbrd

This function opens the clipboard.

Syntax

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinOpenClipbrd  (HAB hab)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Clipboard successfully opened
FALSE	Error occurred.

Remarks

The process reading the clipboard does not become the owner of the object in it; it must not update or free the object.

This function 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.

Related Functions

- WinCloseClipbrd
- WinEmptyClipbrd
- WinEnumClipbrdFmts
- WinQueryClipbrdData
- WinQueryClipbrdFmtInfo
- WinQueryClipbrdOwner
- WinQueryClipbrdViewer
- WinSetClipbrdData

- WinSetClipbrdOwner
- WinSetClipbrdViewer

Example Code

This example opens the clipboard for use by the current process.

```
#define INCL_WINCLIPBOARD       /* Window Clipboard Functions   */
#include <os2.h>

BOOL  fSuccess;          /* success indicator                   */
HAB   hab;               /* anchor-block handle                 */

fSuccess = WinOpenClipbrd(hab);
```

WinOpenObject

This function is specific to version 3, or higher, of the OS/2 operating system.

This function eithers opens a view of the given object or surfaces an existing view.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinOpenObject  (HOBJECT hObjectofObject, ULONG ulView,
                     ULONG ulFlags)
```

Parameters

hObjectofObject (HOBJECT) – input
Handle to a Workplace Shell object to be opened.

ulView (ULONG) – input
View to open this object.

OPEN_SETTINGS
OPEN_TREE
OPEN_DEFAULT
OPEN_CONTENTS
OPEN_DETAILS

ulFlags (ULONG) – input
Flags.

TRUE	Open a view of the object which already exists and supports concurrent views.
FALSE	Open a view of this object by calling wpOpen.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

Using HOBJECT for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

Related Functions

- WinCreateObject
- WinDestroyObject
- WinSaveObject

WinOpenWindowDC

This function opens a device context for a window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
HDC WinOpenWindowDC  (HWND  hwnd)
```

Parameters

hwnd (HWND) – input
Window handle.

Returns

hdc (HDC) – returns
Device-context handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

hdc is used to associate a presentation space with the window.

Note: The window device context is automatically closed when its associated window is destroyed. It must not be closed with the DevCloseDC call.

The visible region of the device context is updated automatically as windows are rearranged.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Example Code

This example calls WinOpenWindowDC to open a device context for a window, the handle to which is then used to associate a presentation space with the window.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_GPICONTROL         /* Gpi Control Functions        */
#include <os2.h>

HWND  hwnd;             /* window handle                        */
HPS    hps;             /* presentation-space handle            */
SIZEL pagesize={0L,0L}; /* Presentation page size               */
HAB   hab;              /* Anchor-block handle                  */
HDC   hdc;              /* device-context handle                */

case WM_CREATE:                    /* Window just created       */

   hdc = WinOpenWindowDC(hwnd);   /* Open window device context */

   hps = GpiCreatePS(hab,              /* Create GPI PS and */
                     hdc,              /* associate with DC */
                     &pagesize,        /* default size */
                     PU_PELS |         /* pixel units */
                     GPIF_LONG |       /* 4-byte coordinates */
                     GPIA_ASSOC);      /* associate with device */
```

WinPeekMsg

This function inspects the thread's message queue and returns to the application with or without a message.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinPeekMsg  (HAB hab, PQMSG pqmsg, HWND hwndFilter,
                  ULONG ulFirst, ULONG ulLast, ULONG flOptions)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pqmsg (PQMSG) – output
Message structure.

Note: If the function returns FALSE, the state of the QMSG structure is undefined. Applications should always check the return code before examining this structure.

hwndFilter (HWND) – input
Window filter.

ulFirst (ULONG) – input
First message identity.

ulLast (ULONG) – input
Last message identity.

flOptions (ULONG) – input
Options.

If neither of the following flags is specified, the message is not removed. If both of the following flags are specified, the message is removed:

PM_REMOVE	Remove message from queue
PM_NOREMOVE	Do not remove message from queue.

Returns

rc (BOOL) – returns
Message-available indicator.

TRUE	Message available
FALSE	No message available.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

This function is identical to the WinGetMsg function, except that it does not wait for the arrival of a message. The message can be left on the queue, by using *flOptions*.

For details of *hwndFilter*, *ulFirst*, and *ulLast*, see the WinGetMsg function.

The window handle within *pqmsg* is null if the message is posted to the queue with a *hwnd* that is null.

Related Functions

- WinBroadcastMsg
- WinCancelShutdown
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example uses WinPeekMsg to count the total number of pending messages for the window corresponding to hwndFilter.

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HAB     hab;            /* anchor-block handle                  */
QMSG    qmsg;           /* message                              */
HWND    hwndFilter;     /* message queue filter                 */
ULONG   flOptions;      /* peek options                         */
ULONG   ulMsgCount=0;   /* message count                        */

/* don't remove messages */
flOptions = PM_NOREMOVE;

/* count number of messages for filter window */
while (WinPeekMsg (hab, &qmsg, hwndFilter, 0, 0, flOptions))
   ulMsgCount++;
```

WinPopupMenu

This function causes a pop-up menu to be presented.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinPopupMenu  (HWND hwndParent, HWND hwndOwner,
                    HWND hwndMenu, LONG x, LONG y, LONG idItem,
                    ULONG fs)
```

Parameters

hwndParent (HWND) – input
Parent-window handle.

hwndOwner (HWND) – input
Owner-window handle.

The owner window receives all the notification messages generated by the pop-up menu.

hwndMenu (HWND) – input
Pop-up menu-window handle.

The pop-up menu must have been created, by use of either the WinCreateMenu or WinLoadMenu functions.

x (LONG) – input
x-coordinate of the pop-up menu position.

The value is in window coordinates relative to the origin of the parent window.

The x-coordinate of the origin of the pop-up menu can be affected, if either of the PU_POSITIONONITEM or PU_HCONSTRAIN values of the *fs* parameter is also set.

y (LONG) – input
y-coordinate of the pop-up menu position.

The value is in window coordinates relative to the origin of the parent window.

The y-coordinate of the origin of the pop-up menu can be affected, if either of the PU_POSITIONONITEM or PU_VCONSTRAIN values of the *fs* parameter is also set.

idItem (LONG) – input
Item identity.

This is used if either the PU_POSITIONONITEM or PU_SELECTITEM of the *fs* parameter is also set. It must be greater or equal to 0 and less or equal to 0xFFFF.

fs (ULONG) – input
Options.

Position
Pop-up menu position.

PU_POSITIONONITEM — Position the pop-up menu so that the item identified by the *idItem* parameter of the top-level menu specified by the *hwndMenu* parameter lies directly under the *x* \ *y* coordinates.

The position of the pop-up menu can be affected, if either the PU_HCONSTRAIN or or PU_VCONSTRAIN values of the *fs* parameter is also set.

This value also causes the pop-up menu item identified by the *idItem* to be selected.

Restrain
Pop-up menu position constraints.

These options allow the application to ensure that the pop-up menu is visible on the desktop.

PU_HCONSTRAIN — Constrain the pop-up menu so that its width is wholly visible on the desktop.

If necessary the position of the pop-up menu will be adjusted so that its left edge is coincident with the left edge of the desktop or that its right edge is coincident with the right edge of the desktop.

PU_VCONSTRAIN — Constrain the pop-up menu so that its height is wholly visible on the desktop.

If necessary the position of the pop-up menu will be adjusted so that its top edge is coincident with the top edge of the desktop or that its bottom edge is coincident with the bottom edge of the desktop.

InitialState
Initial input state of the pop-up menu.

This allows the user interaction which caused the application to summon the pop-up menu to be carried through as the initial user interaction with the pop-up menu.

For example, this permits the application to support the user interface in which mouse button 1 can be depressed to cause the pop-up menu to be presented and held down while moving the mouse over the menu in order to select another menu item and then released to dismiss the menu.

Only one of the following values can be selected:

PU_MOUSEBUTTON1DOWN — The pop-up menu is initialized with mouse button 1 depressed.

PU_MOUSEBUTTON2DOWN — The pop-up menu is initialized with mouse button 2 depressed.

PU_MOUSEBUTTON3DOWN — The pop-up menu is initialized with mouse button 3 depressed.

PU_NONE — The pop-up menu is to be presented uninfluenced by the user interaction which caused it to be summoned.

This is the default value.

Select

Item selection.

PU_SELECTITEM — The item identified by *idItem* is to be selected. This is only valid if PU_NONE is set in the *InitialState* parameter.

If the identified item is in a submenu of the pop-up menu, then all the previous submenus in the menu hierarchy are presented with the correct path to the identified item.

Usage

Input device usage.

The window procedure controlling the pop-up menu must be informed of which input devices are available for interaction with the pop-up menu.

These options are independent to those of the *InitialState* parameter. Therefore, if an application indicates in the *InitialState* parameter that the pop-up menu is to be initialized with a particular user interaction, then the mechanism which permits that user interaction would usually be specified in this parameter. In this way the user's expectation, that once a device has been employed for the manipulation of the pop-up menu then that device can continue to be used for that purpose, is fulfilled.

It is valid to specify a user interaction as an initialization of the pop-up menu by an input mechanism which is not identified as available for interaction with the pop-up menu. This implies that the user cannot necessarily complete the interaction with the pop-up menu with that input mechanism.

For example, if a pop-up menu is initialized with a mouse button depressed but that mouse button is not identified as available for manipulating the pop-up menu, then that mouse button can manipulate the pop-up menu until it is released. Assuming that the pop-up menu is not dismissed when that mouse button is released, then the mouse button cannot be used for further interaction with the pop-up menu, since it is not identified as available for that use.

The following list shows the input device valid for interaction with the pop-up menu with each option:

PU_KEYBOARD — The keyboard.
PU_MOUSEBUTTON1 — Mouse button 1.
PU_MOUSEBUTTON2 — Mouse button 2.
PU_MOUSEBUTTON3 — Mouse button 3.

Returns

rc (BOOL) – returns

Pop-up menu invoked indicator.

This function returns as soon as the pop-up menu has been invoked, which might be before the user has completed interacting with the pop-up menu.

TRUE	Pop-up menu successfully invoked
FALSE	Pop-up menu not successfully invoked.

Remarks

A pop-up menu is the unanchored equivalent of a pull-down menu, that is it can be positioned anywhere rather than being associated with an action bar. Typically, pop-up menus are related to specific objects, such as an icon, or with a particular area of the application's presentation space.

Once invoked, a pop-up menu behaves in exactly the same way as a pull-down menu.

Related Functions

- WinCreateMenu
- WinLoadMenu

Example Code

This example presents a pop-up menu (loaded from RES.DLL by WinLoadMenu) with the following characteristics: located at (0,50); initialized with mouse button 1 depressed; allowing keyboard and mouse button 1 interaction.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINMENUS           /* Window Menu Functions        */
#include <os2.h>

HWND    hwndMenu;       /* menu window                          */
HWND    hwndOwner;      /* owner window                         */
HMODULE hmodDLL;        /* resource handle                      */
ULONG   idMenuid;       /* resource menu id                     */
BOOL    fSuccess;       /* success indicator                    */
HWND    hwndParent;     /* parent window                        */
ULONG   flOptions;      /* pop-up menu options                  */

if (DosQueryModuleHandle("RES.DLL",&hmodDLL))
   hwndMenu = WinLoadMenu(hwndOwner, hmodDLL, idMenuid);

flOptions = PU_MOUSEBUTTON1DOWN | PU_KEYBOARD | PU_MOUSEBUTTON1;
fSuccess = WinPopupMenu(hwndParent, hwndOwner, hwndMenu, 0, 50,
                        0, flOptions);
```

WinPostMsg

This function posts a message to the message queue associated with the window defined by *hwnd*.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinPostMsg  (HWND hwnd, ULONG ulMsgid, MPARAM mpParam1,
                  MPARAM mpParam2)
```

Parameters

hwnd (HWND) – input
Window handle.

NULL The message is posted into the queue associated with the current thread. When the message is received by using the WinGetMsg or WinPeekMsg functions, the *hwnd* parameter of the QMSG structure is NULL.

Other Window handle.

ulMsgid (ULONG) – input
Message identity.

mpParam1 (MPARAM) – input
Parameter 1.

mpParam2 (MPARAM) – input
Parameter 2.

Returns

rc (BOOL) – returns
Message-posted indicator.

TRUE Message successfully posted
FALSE Message could not be posted; for example, because the message queue was full.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001) An invalid window handle was specified.

Remarks

The message contains *hwnd*, *ulMsgid*, *mpParam1*, *mpParam2*, and the time and pointer position when this function is called.

WinPostMsg returns immediately, while WinSendMsg waits for the receiver to return.

A thread which does not have a message queue can still call WinPostMsg but cannot call WinSendMsg.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example posts a Set menu item checked attribute message (MM_SETITEMATTR) to the message queue associated with the window handle in response to a menu select message (WM_MENUSELECT).

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINMENUS           /* Window Menu Functions        */
#include <os2.h>

BOOL    fResult;        /* message-posted indicator             */
ULONG   ulMsgid;        /* message id                           */
MPARAM  mp1;            /* Parameter 1 (rectl structure)        */
MPARAM  mp2;            /* Parameter 2 (frame boolean)          */
USHORT  usItemId;       /* menu item id                         */
HWND    hwndMenu;       /* menu handle                          */

case WM_MENUSELECT:
     usItemId = SHORT1FROMMP(mp1);
     hwndMenu = HWNDFROMMP(mp2);

     /* initialize message id, parameters */
     ulMsgid = MM_SETITEMATTR;
     mp1 = MPFROM2SHORT(usItemId, TRUE);
     mp2 = MPFROM2SHORT(MIA_CHECKED, TRUE);

     fResult = WinPostMsg(hwndMenu, ulMsgid, mp1, mp2);
```

WinPostQueueMsg

This function posts a message to a message queue.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinPostQueueMsg  (HMQ hmq, ULONG msg, MPARAM mp1,
                       MPARAM mp2)
```

Parameters

hmq (HMQ) – input
Message-queue handle.

msg (ULONG) – input
Message identifier.

mp1 (MPARAM) – input
Parameter 1.

mp2 (MPARAM) – input
Parameter 2.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or the queue was full.

Possible returns from WinGetLastError

PMERR_INVALID_HMQ (0x1002)	An invalid message-queue handle was specified.

Remarks

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

It constructs a QMSG structure by setting its *hwnd* parameter to NULL, setting its *msg*, *mp1*, and *mp2* parameters from the corresponding parameters of this function, and by deriving its *time* and *ptl* parameters from the system time and pointer position when this function was called. The QMSG structure is then placed on the specified queue.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example posts a Set menu item checked attribute message (MM_SETITEMATTR) to the specified message queue in response to a menu select message (WM_MENUSELECT).

```
#define INCL_WINMESSAGEMGR      /* Window Message Functions     */
#define INCL_WINMENUS           /* Window Menu Functions        */
#include <os2.h>

BOOL    fResult;        /* message-posted indicator             */
ULONG   ulMsgid;        /* message id                           */
HMQ     hmq;            /* message queue handle                 */
MPARAM  mp1;            /* Parameter 1 (rectl structure)        */
MPARAM  mp2;            /* Parameter 2 (frame boolean)          */
USHORT  usItemId;       /* menu item id                         */

case WM_MENUSELECT:
     usItemId = SHORT1FROMMP(mp1);

     /* initialize message id, parameters */
     ulMsgid = MM_SETITEMATTR;
     mp1 = MPFROM2SHORT(usItemId, TRUE);
     mp2 = MPFROM2SHORT(MIA_CHECKED, TRUE);

     fResult = WinPostQueueMsg(hmq, ulMsgid, mp1, mp2);
```

WinPrevChar

This function moves to the previous character in a string.

Syntax

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
PSZ WinPrevChar (HAB hab, ULONG ulCodepage, ULONG ulCountry,
                 PSZ pszStart, PSZ pszCurrentChar)
```

Parameters

hab (HAB) – input
Anchor-block handle.

ulCodepage (ULONG) – input
Code page.

If 0 is specified for *ulCodepage*, the code page specified by the queue associated with the calling thread is used.

ulCountry (ULONG) – input
Reserved value, must be 0.

pszStart (PSZ) – input
Character string that contains *pszCurrentChar*.

pszCurrentChar (PSZ) – input
Current character.

Returns

pszPrevChar (PSZ) – returns
Previous character.

The previous character, or the first character if *pszCurrentChar* is the first character of *pszStart*.

Possible returns from WinGetLastError

PMERR_INVALID_STRING_PARM (0x100B)	The specified string parameter is invalid.

Remarks

This function handles DBCS strings.

The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865
Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft Windows.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data. The following is the list of valid country codes:

Country	Code
Arabic	785
Australian	61
Belgian	32
Canadian-French	2
Danish	45
Finnish	358

French	33
German	49
Hebrew	972
Italian	39
Japanese	81
Korean	82
Latin-American	3
Netherlands	31
Norwegian	47
Portuguese	351
Simpl. Chinese	86
Spanish	34
Swedish	46
Swiss	41
Trad. Chinese	88
UK-English	44
US-English	1
Other country	0

Related Functions

- WinCompareStrings
- WinLoadString
- WinNextChar
- WinSubstituteStrings
- WinUpper
- WinUpperChar

Example Code

This example uses WinPrevChar to return a pointer to the previous character in a string.

```
#define INCL_DOSNLS
#define INCL_WINCOUNTRY
#include <OS2.H>
#include <stdio.h>
#define CURRENT_COUNTRY 0

main()
{
HAB hab;                /* anchor-block handle. */
char string[] = "ABCDEFGHIJ";
char *ptoE = &string[4];
char *ptoD;
ULONG   CodePage;       /* List (returned) */
ULONG  DataLength;      /* Length of list (returned) */
COUNTRYCODE Country;
COUNTRYINFO CtryBuffer;

Country.country = CURRENT_COUNTRY;

DosQueryCp((ULONG)2,
           &CodePage,    /* get code page identifier of calling */
                         /* process.                            */
          &DataLength);

          /* first WORD contains the codepage.        */
Country.codepage = (ULONG)HIUSHORT(CodePage);

         /* get corresponding country code */
DosQueryCtryInfo(sizeof(CtryBuffer),  /* Length of data area */
                                      /*          provided */
               &Country,              /* Input data structure */
               &CtryBuffer,           /* Data area to be filled */
                                      /* by function */
             &DataLength);            /* Length of data */

/* should return a pointer to character "D" in the string */

ptoD = WinPrevChar(hab,
                   (ULONG)CodePage,
                   (ULONG)CtryBuffer.country,
                  (PSZ)string,
                  ptoE);  /* pointer to character "E" in the */
                          /* string.                         */

printf(ptoE);
}
```

WinProcessDlg

This function dispatches messages while a modal dialog window is displayed.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinProcessDlg  (HWND hwndDlg)
```

Parameters

hwndDlg (HWND) – input
Dialog-window handle.

Returns

ulReply (ULONG) – returns
Reply value.

Value established by the WinDismissDlg function.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

If the dialog has an owner window, that window is disabled. This means that all user input to the owner, and its descendants, is prevented.

This function then dispatches messages from the queue to the appropriate window or dialog procedure until the dialog is dismissed by the WinDismissDlg function. This is usually done by the dialog procedure on receipt of an appropriate message, but also occurs if the dialog procedure passes a WM_COMMAND message to WinDefDlgProc or if a WM_QUIT message is encountered before the dialog window is dismissed. In this latter case, WinProcessDlg itself issues a WinDismissDlg function, and posts the WM_QUIT message back to the queue so that the application main loop terminates in the normal way.

This function shows the window, if it is hidden, when the queue is empty. It is therefore possible for the experienced user to type ahead and cause the dialog to be dismissed before it becomes visible.

The WinDismissDlg function hides the dialog window without destroying it, and also re-enables any window that was disabled by this function.

This function does not return until a WinDismissDlg call is issued in one of the ways listed above. This is true even if the application main window has not been disabled, for example because the dialog window has no owner. In this case, the dialog will appear to the user to be modeless; the user will continue to be able to interact with the application, and possibly create multiple instances of the dialog. In such circumstances the operating system calls the application main window procedure recursively before WinProcessDlg returns.

It is not possible to temporarily disable more than one window using this function; a dialog window can have at most one owner. If an application has more than one main window which should be disabled while the modal dialog is displayed, it can be done by setting appropriate hooks using the WinSetHook function.

If the dialog window is a descendant of its owner, this function disables input to the dialog itself. However, this situation can only occur by explicitly changing the window hierarchy. Dialog windows are created using the WinLoadDlg or WinCreateDlg functions, which modify the owner window specified on their parameter lists.

Related Functions

- WinCreateDlg
- WinDefDlgProc
- WinDismissDlg
- WinDlgBox
- WinGetDlgMsg
- WinLoadDlg

Related Messages

- WM_COMMAND
- WM_QUIT

Example Code

This function is used to process messages while a dialog is active.

```
#define INCL_WIN
#define INCL_WINDIALOGS
#include <OS2.H>
#define IDD_OPEN WM_USER+200
#define IDM_OPEN WM_USER+201

HWND hwndDlg;
HWND hwndFrame;
PFNWP OpenDlg;


/* Inside client procedure. */

switch(msg)
{
 case WM_COMMAND:
 /* The user has chosen a menu item.  Process the selection */
 /* accordingly.                                            */

  switch ( SHORT1FROMMP( mp1 ) )
  {
  case IDM_OPEN:
     if (WinDlgBox(HWND_DESKTOP,
              hwndFrame,    /* handle of the owner         */
             OpenDlg,      /* dialog procedure address    */
          (ULONG)0,         /* location of dialog resource */
              IDD_OPEN,     /* resource identifier         */
              NULL)) {      /* application-specific data   */

    WinProcessDlg(hwndDlg);
                    }
    break;

   }
 break;
```

WinPtInRect

This function queries whether a point lies within a rectangle.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinPtInRect  (HAB hab, PRECTL prcl, PPOINTL pptl)
```

Parameters

hab (HAB) – input
Anchor-block handle.

prcl (PRECTL) – input
Rectangle to be queried.

Note: The value of each field in this structure must be in the range -32768 through 32767. The data type WRECT may also be used, if supported by the language.

pptl (PPOINTL) – input
Point to be queried.

Returns

rc (BOOL) – returns
Success indicator.

TRUE *pptl* lies within *prcl*.
FALSE *pptl* does not lie within *prcl*, or an error occurred.

Related Functions

- WinCopyRect
- WinEqualRect
- WinFillRect
- WinInflateRect
- WinIntersectRect
- WinIsRectEmpty
- WinOffsetRect
- WinSetRect
- WinSetRectEmpty
- WinSubtractRect
- WinUnionRect

Example Code

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.

```
#define INCL_WIN
#define INCL_WINRECTANGLES
#include <OS2.H>

HAB hab;          /* anchor-block handle */
/* . */
/* . */
RECTL rclGlobal;
POINTL ptl;
HPS hps;

USHORT msg;
MPARAM mp1;

/* inside client window function. */

switch(msg)
{
 case WM_COMMAND:
 /* The user has chosen a menu item.  Process the selection */
 /* accordingly.                                            */

  switch ( SHORT1FROMMP( mp1 ) )
  {

   case WM_BUTTON1UP:
     ptl.x = (LONG) SHORT1FROMMP(mp1);
     ptl.y = (LONG) SHORT2FROMMP(mp1);
        WinPtInRect(hab,   /* anchor-block handle      */
         &rclGlobal,       /* address of the rectangle */
         &ptl);            /* address of the point     */
   break;
   }
 break;
}
```

WinQueryAccelTable

This function queries the window or queue accelerator table.

Syntax

```
#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HACCEL WinQueryAccelTable  (HAB hab, HWND hwndFrame)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hwndFrame (HWND) – input
Frame-window handle.

NULLHANDLE	Return queue accelerator.
Other	Return the window accelerator table, by sending the WM_QUERYACCELTABLE message to *hwndFrame*.

Returns

haccelAccel (HACCEL) – returns
Accelerator-table handle.

NULLHANDLE	Error occurred
Other	Accelerator-table handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinCopyAccelTable
- WinCreateAccelTable
- WinDestroyAccelTable
- WinLoadAccelTable
- WinSetAccelTable
- WinTranslateAccel

Related Messages

- WM_QUERYACCELTABLE

Example Code

This example shows how to get the accelerator table for the frame window.

```
#define INCL_WINWINDOWMGR
#define INCL_WINACCELERATORS
#include <OS2.H>

HACCEL haccel;
HWND hwndFrame, hwndClient; /* window handles. */
HAB hab;                    /* anchor block.   */

hwndFrame = WinQueryWindow(hwndClient,
                           QW_PARENT); /* get handle of parent, */
                                       /* which is frame window. */


    /* Now  get the accel table for the frame window  */
haccel = WinQueryAccelTable(hab,
                            hwndFrame);
```

WinQueryActiveDesktopPathname

This function is specific to version 3, or higher, of the OS/2 operating system.

This function returns the directory specification of the active desktop.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinQueryActiveDesktopPathname  (PSZ pszPathName, ULONG ulSize)
```

Parameters

pszPathName (PSZ) – output
Memory allocated by caller in which directory specification is written.

ulSize (ULONG) – input
Number of bytes pointed to by *pszPathName*.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETER (0x1645)
An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32,768 to +32,767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

Remarks

This function is used to find the directory specification of the current desktop. The current desktop is not always \DESKTOP of the boot drive.

Example Code

This example finds the directory specification of the current desktop.

```
#define INCL_WINWORKPLACE
#include <os2.h>

CHAR    szPath[CCHMAXPATH + 1];
BOOL    fSuccess;

fSuccess = WinQueryActiveDesktopPathname(szPath,sizeof(szPath));
if (fSuccess)
{
  /* WinQueryActiveDesktopPathname was successful */
}
else
{
  /* WinQueryActiveDesktopPathname failed */
}
```

WinQueryActiveWindow

This function returns the active window for HWND_DESKTOP, or other parent window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinQueryActiveWindow  (HWND hwndParent)
```

Parameters

hwndParent (HWND) – input
Parent-window handle for which the active window is required.

HWND_DESKTOP	The desktop-window handle that causes this function to return the top-level frame window.
Other	Specified parent-window handle.

Returns

hwndActive (HWND) – returns
Active-window handle.

NULLHANDLE	No window is active
Other	Active-window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinGetMinPosition
- WinQueryWindowPos
- WinSaveWindowPos
- WinSetActiveWindow
- WinSetMultWindowPos
- WinSetWindowPos

Example Code

This example shows how the WinQueryActiveWindow can be used to find the active window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwndActive;

hwndActive = WinQueryActiveWindow(HWND_DESKTOP)
```

WinQueryAnchorBlock

This function returns the anchor block handle of the caller.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
HAB WinQueryAnchorBlock  (HWND hwnd)
```

Parameters

hwnd (HWND) – input
Window handle.

Returns

hab (HAB) – returns
Anchor block handle.

NULLHANDLE	Invalid *hwnd* parameter
Other	Anchor block handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Example Code

This function obtains the anchor block handle of the caller.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HAB hab;
HWND hwnd;

hab =  WinQueryAnchorBlock(hwnd);
```

WinQueryAtomLength

This function queries the length of an atom represented by the specified atom.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryAtomLength  (HATOMTBL hatomtblAtomTbl, ATOM atom)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

The handle returned from a previous WinCreateAtomTable or WinQuerySystemAtomTable function.

atom (ATOM) – input
Atom whose associated character-string length is to be returned.

Returns

ulretlen (ULONG) – returns
String length.

0	The specified atom or the atom table is invalid.
Other	The length of the character string associated with the atom **excluding** the null terminating byte. Integer atoms always return a length of six.

Possible returns from WinGetLastError

PMERR_INVALID_HATOMTBL (0x1013)	An invalid atom-table handle was specified.
PMERR_INVALID_ATOM (0x1014)	The specified atom does not exist in the atom table.

Remarks

The purpose of this function is to allow an application to determine the size of buffer to use in the WinQueryAtomName call.

Related Functions

- WinAddAtom
- WinCreateAtomTable
- WinDeleteAtom
- WinDestroyAtomTable

- WinFindAtom
- WinQueryAtomName
- WinQueryAtomUsage
- WinQuerySystemAtomTable

Example Code

This function queries the length of an atom.

```
#define INCL_WINATOM
#include <OS2.H>

HATOMTBL atomtbl;
ATOM atom = 25;

      WinQueryAtomlength(atomtbl,  /* atom handle. */
                         atom);
```

WinQueryAtomName

This function returns an atom name associated with an atom.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryAtomName (HATOMTBL hatomtblAtomTbl, ATOM atom,
                        PSZ pszBuffer, ULONG ulBufferMax)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

The handle returned from a previous WinCreateAtomTable or WinQuerySystemAtomTable function.

atom (ATOM) – input
Identifies the character string to be retrieved.

pszBuffer (PSZ) – output
Buffer to receive the character string.

ulBufferMax (ULONG) – input
Buffer size in bytes.

Returns

ulretlen (ULONG) – returns
Length of retrieved character string.

0	The specified atom or the atom table is invalid.
Other	The number of bytes copied to the buffer **excluding** the terminating zero.

Possible returns from WinGetLastError

PMERR_INVALID_HATOMTBL (0x1013)	An invalid atom-table handle was specified.
PMERR_INVALID_ATOM (0x1014)	The specified atom does not exist in the atom table.
PMERR_INVALID_STRING_PARM (0x100B)	The specified string parameter is invalid.

Remarks

For integer atoms, the format of the string is "#ddddd" where "ddddd" are decimal digits in the system code page (an ASCII code page). No leading zeros are generated, and the length can be from 3 through 7 characters.

Related Functions

- WinAddAtom
- WinCreateAtomTable
- WinDeleteAtom
- WinDestroyAtomTable
- WinFindAtom
- WinQueryAtomLength
- WinQueryAtomUsage
- WinQuerySystemAtomTable

Example Code

This function obtains the name of an atom given the atom id.

```
#define INCL_WINATOM
#include <OS2.H>
HATOMTBL atomtbl;
char atomname[256];
ATOM atom = 25;

        WinQueryAtomName(atomtbl,
                         atom,
                         atomname,
                         sizeof(atomname));
```

WinQueryAtomUsage

This function returns the number of times an atom has been used.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryAtomUsage (HATOMTBL hatomtblAtomTbl, ATOM atom)
```

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

The handle returned from a previous WinCreateAtomTable or WinQuerySystemAtomTable function.

atom (ATOM) – input
Atom whose use count is to be returned.

Returns

ulcount (ULONG) – returns
Use count of the atom.

65535	Integer atom
0	The specified atom or the atom table is invalid
Other	Use count.

Possible returns from WinGetLastError

PMERR_INVALID_HATOMTBL (0x1013)	An invalid atom-table handle was specified.
PMERR_INVALID_ATOM (0x1014)	The specified atom does not exist in the atom table.

Related Functions

- WinAddAtom
- WinCreateAtomTable
- WinDeleteAtom
- WinDestroyAtomTable
- WinFindAtom
- WinQueryAtomLength
- WinQueryAtomName
- WinQuerySystemAtomTable

Example Code

This function returns the number of times an atom has been used.

```
#define INCL_WINATOM
#include <OS2.H>

HATOMTBL atomtbl;
ATOM atom = 25;

      WinQueryAtomlength(atomtbl,
                         atom);
```

WinQueryButtonCheckstate

This macro returns the checked state of the button control specified.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
USHORT WinQueryButtonCheckstate (HWND hwndDlg, USHORT usId)
```

Parameters

hwndDlg (HWND) – input
Dialog window handle.

usId (USHORT) – input
Button control identity.

Returns

usRetChkst (USHORT) – returns
Returns the checkstate of the specified button control.

Remarks

This macro expands to:

```
#define WinQueryButtonCheckstate(hwndDlg, usId)
  ((USHORT)WinSendDlgItemMsg(hwndDlg,
                             usId,
                             BM_QUERYCHECK,
                             (MPARAM)NULL,
                             (MPARAM)NULL))
```

This function requires the existence of a message queue.

Related Functions

- WinSendDlgItemMsg

Related Messages

- BM_QUERYCHECK

Example Code

This function returns the checked state of the button control specified.

```
#define INCL_WINWINDOWMGR
#define INCL_WINBUTTONS
#include <OS2.H>

#define IDM_BUTTONA 900

HWND hwndDlg;
USHORT ChkState;


ChkState = WinQueryButtonCheckstate(hwndDlg,
                                    IDM_BUTTONA);

switch (ChkState)
{


  case 0:

     /* Unchecked */
   break;
  case 1:

     /* Checked */
     break;
  case 2:

     /* Indeterminate. */
     break;
}
```

WinQueryCapture

This function returns the handle of the window that has the pointer captured.

Syntax

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinQueryCapture (HWND hwndDesktop)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Returns

hwnd (HWND) – returns
Handle of the window with the pointer captured.

NULLHANDLE	No window has the pointer captured, or an error occurred
Handle	Handle of the window with the pointer captured.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinSetCapture

Example Code

This function returns the handle of the window that has the pointer captured.

```
#define INCL_WININPUT
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND hwnd;    /* handle of window that has pointer captured */

hwnd = WinQueryCapture(HWND_DESKTOP);  /* window that has  */
                                       /* pointer captured */
```

WinQueryClassInfo

This function returns window class information.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinQueryClassInfo  (HAB hab, PSZ PSZClassName,
                         PCLASSINFO PclsiClassInfo)
```

Parameters

hab (HAB) – input
Anchor-block handle.

PSZClassName (PSZ) – input
Class name.

PclsiClassInfo (PCLASSINFO) – output
Class information structure.

Returns

rc (BOOL) – returns
Class-exists indicator.

TRUE	Class does exist
FALSE	Class does not exist.

Possible returns from WinGetLastError

PMERR_INVALID_INTEGER_ATOM (0x1016)	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME (0x1015)	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND (0x1017)	The specified atom name is not in the atom table.

Remarks

PSZClassName is either an application-specified name (as defined by the WinRegisterClass call) or the name of a preregistered WC_* class. see "Control Window Message Processing" in the *Presentation Manager Programming Reference Volume I*. Preregistered class names are of the form #nnnnn, where nnnnn is up to five digits corresponding to the value of the WC_* class name constant.

This function provides information that is needed to create a subclass of a given class (see WinSubclassWindow).

Related Functions

- WinCalcFrameRect
- WinCreateFrameControls
- WinCreateStdWindow
- WinCreateWindow
- WinDefWindowProc
- WinDestroyWindow
- WinQueryClassName
- WinRegisterClass
- WinSubclassWindow

Example Code

This example obtains a pointer to the window procedure of the window class WC_COMBOBOX.

```
#define INCL_WINWINDOWMGR
#define INCL_WINENTRYFIELDS
#include <OS2.H>
HAB hab;
/* . */
CLASSINFO classinfo;
PFNWP pWindowProc;


WinQueryClassInfo(hab,
                  WC_COMBOBOX,
                  &classinfo);

pWindowProc = classinfo.pfnWindowProc;
```

WinQueryClassName

This function copies the window class name, as a null-terminated string, into a buffer.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinQueryClassName  (HWND hwnd, LONG lLength, PCH PCHBuffer)
```

Parameters

hwnd (HWND) – input
Window handle.

If this window is of any of the preregistered WC_* classes (see "Control Window Message Processing" in the *Presentation Manager Programming Reference Volume I*) the class name returned in the *PCHBuffer* parameter is in the form "#nnnnn," where "nnnnn" is a group of up to five digits that corresponds to the value of the WC_* class name constant.

lLength (LONG) – input
Length of *PCHBuffer*.

It must be greater than 0.

PCHBuffer (PCH) – output
Class name.

If the class name is longer than (*lLength*–1) only the first (*lLength*–1) characters of class name are copied.

Returns

lRetLen (LONG) – returns
Returned class name length.

This is the length, *excluding* the null-termination character.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_STRING_PARM (0x100B)	The specified string parameter is invalid.

Related Functions

- WinCalcFrameRect
- WinCreateFrameControls
- WinCreateStdWindow
- WinCreateWindow
- WinDefWindowProc
- WinDestroyWindow
- WinQueryClassInfo
- WinRegisterClass
- WinSubclassWindow

Example Code

This example obtains a pointer to the window procedure of the window class, given that we know the window handle.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HAB hab;
/* . */
HWND hwnd;
CLASSINFO classinfo;
PFNWP pWindowProc;
char *classname;


WinQueryClassName(hwnd,
                  sizeof(classname),
                  classname);


WinQueryClassInfo(hwnd,
                  classname,
                  &classinfo);

pWindowProc = classinfo.pfnWindowProc;
```

WinQueryClassThunkProc

This call queries the pointer-conversion procedure associated with a class.

Syntax

```
#define INCL_WINTHUNKAPI /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
PFN WinQueryClassThunkProc  (PSZ pszClassName)
```

Parameters

pszClassName (PSZ) – input
Window-class name.

Returns

thunkpr (PFN) – returns
Pointer-conversion procedure identifier.

NULL	No pointer-conversion procedure is associated with this class.
Other	Identifier of the pointer-conversion procedure associated with this class.

Related Functions

- WinQueryWindowModel
- WinQueryWindowThunkProc
- WinSetClassThunkProc
- WinSetWindowThunkProc

Example Code

This example obtains the pointer conversion procedure of the window class, given that we have an anchor-block handle.

```
#define INCL_WINWINDOWMGR
#define INCL_WINTHUNKAPI
#include <OS2.H>
HWND hwnd;
/* . */
PFN pfn;
char *classname;


WinQueryClassName(hwnd,
                  sizeof(classname),
                  classname);


pfn = WinQueryClassThunkProc(classname);
```

WinQueryClipbrdData

This function obtains a handle to the current clipboard data with a specified format.

Syntax

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryClipbrdData (HAB hab, ULONG fmt)
```

Parameters

hab (HAB) – input
Anchor-block handle.

fmt (ULONG) – input
Format of the data to be accessed.

In addition to the predefined formats, private formats can be created using the standard system atom manager. The following is the list of standard clipboard formats:

CF_TEXT	Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.
CF_DSPTEXT	Text display format associated with private format.
CF_BITMAP	Bit map.
CF_DSPBITMAP	Bit-map display format associated with private format.
CF_METAFILE	Metafile.
CF_DSPMETAFILE	Metafile display format associated with private format.
CF_PALETTE	Palette.

Returns

ulRet (ULONG) – returns
Handle to the clipboard data.

0	Format does not exist, or an error occurred
Other	Handle to the clipboard data.

Remarks

The returned data handle must not be used after the WinCloseClipbrd function is called. For this reason, the application must either copy the data (if required for long-term use) or process the data before the WinCloseClipbrd function is called. The application must neither free the data handle itself, nor leave it locked in any way.

Information about the format of the data in the clipboard can be obtained from WinQueryClipbrdFmtInfo.

Related Functions

- WinCloseClipbrd
- WinEmptyClipbrd
- WinEnumClipbrdFmts
- WinOpenClipbrd
- WinQueryClipbrdFmtInfo
- WinQueryClipbrdOwner
- WinQueryClipbrdViewer
- WinSetClipbrdData
- WinSetClipbrdOwner
- WinSetClipbrdViewer

Example Code

This example obtains a handle to the current clipboard data of text format.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
HAB hab;
/* . */
ULONG hclipbrdData;

hclipbrdData = WinQueryClipbrdData(hab, CF_TEXT);
```

WinQueryClipbrdFmtInfo

This function determines whether a particular format of data is present in the clipboard, and if so, provides information about that format.

Syntax

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinQueryClipbrdFmtInfo (HAB hab, ULONG fmt, PULONG prgfFmtInfo)
```

Parameters

hab (HAB) – input
Anchor-block handle.

fmt (ULONG) – input
Format of the data to be queried.

In addition to the predefined formats, private formats can be created using the standard system atom manager. The following is the list of standard clipboard formats:

CF_TEXT	Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.
CF_DSPTEXT	Text display format associated with private format.
CF_BITMAP	Bit map.
CF_DSPBITMAP	Bit-map display format associated with private format.
CF_METAFILE	Metafile.
CF_DSPMETAFILE	Metafile display format associated with private format.
CF_PALETTE	Palette.

prgfFmtInfo (PULONG) – output
Memory model and usage flags.

These are the usage flags set by the setting application; that is, the CFI_* flags of the *flFmtInfo* parameter of the WinSetClipbrdData function.

If the format is CF_BITMAP, CF_DSPBITMAP, CF_METAFILE or CF_DSPMETAFILE, *prgfFmtInfo* is set to CFI_HANDLE. If the format is CF_TEXT or CF_DSPTEXT, *prgfFmtInfo* is set to CFI_POINTER. If the format is user-defined, *prgfFmtInfo* is set to the value used in the WinSetClipbrdData function.

Returns

rc (BOOL) – returns
Format-exists indicator.

TRUE *fmt* exists in the clipboard and *prgfFmtInfo* is set
FALSE *fmt* does not exist in the clipboard and *prgfFmtInfo* is not set.

Possible returns from WinGetLastError

PMERR_INVALID_FLAG (0x1019) An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

This function does not cause the data to be rendered.

Applications can put information in the clipboard in their own private format. Use the following function calls to create and query a private format:

```
fDataPlaced = WinSetClipbrdData(hab, ulh,
                                WinAddAtom(WinQuerySystemAtomTable(),
                                           "Private Data Type"),
                                flFmtInfo);


ulData = WinQueryClipbrdData(hab,
                             WinFindAtom(WinQuerySystemAtomTable(),
                                         "Private Data Type") );
```

Related Functions

- WinCloseClipbrd
- WinEmptyClipbrd
- WinEnumClipbrdFmts
- WinOpenClipbrd
- WinQueryClipbrdData
- WinQueryClipbrdOwner
- WinQueryClipbrdViewer
- WinSetClipbrdData
- WinSetClipbrdOwner
- WinSetClipbrdViewer

Example Code

This example obtains a handle to the current clipboard data of text format if that format is present in the clipboard.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
HAB hab;
/* . */
ULONG  format;
ULONG hclipbrdData;


if (WinQueryClipbrdData(hab,CF_TEXT))
{
 hclipbrdData = WinQueryClipbrdFmfInfo(hab,
                                       CF_TEXT
                                       &format);
}
```

WinQueryClipbrdOwner

This function obtains any current clipboard owner window.

Syntax

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinQueryClipbrdOwner  (HAB hab)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

hwndClipbrdOwner (HWND) – returns
Window handle of the current clipboard owner.

NULLHANDLE	If the clipboard is not owned by any window, or if an error occurred.
Other	Window handle of the current clipboard owner.

Related Functions

- WinCloseClipbrd
- WinEmptyClipbrd
- WinEnumClipbrdFmts
- WinOpenClipbrd
- WinQueryClipbrdData
- WinQueryClipbrdFmtInfo
- WinQueryClipbrdViewer
- WinSetClipbrdData
- WinSetClipbrdOwner
- WinSetClipbrdViewer

Example Code

This example finds out which window currently owns the clipboard.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
HAB hab;
/* . */
HWND  hwndClipbrdOwner;


hwndClipbrdOwner = WinQueryClipbrdOwner(hab);
```

WinQueryClipbrdViewer

This function obtains any current clipboard viewer window.

Syntax

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinQueryClipbrdViewer  (HAB hab)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

hwndClipbrdViewer (HWND) – returns
Current clipboard viewer window handle.

NULLHANDLE	Clipboard does not have a current viewer window, or an error occurred
Other	Current clipboard viewer window handle.

Related Functions

- WinCloseClipbrd
- WinEmptyClipbrd
- WinEnumClipbrdFmts
- WinOpenClipbrd
- WinQueryClipbrdData
- WinQueryClipbrdFmtInfo
- WinQueryClipbrdOwner
- WinSetClipbrdData
- WinSetClipbrdOwner
- WinSetClipbrdViewer

Example Code

This example finds out which window currently owns the clipboard.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
HAB hab;
/* . */
HWND  hwndClipbrdViewer;


hwndClipbrdViewer = WinQueryClipbrdViewer(hab);
```

WinQueryCp

This function returns the queue code page for the specified message queue.

Syntax

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryCp  (HMQ hmq)
```

Parameters

hmq (HMQ) – input
Message queue.

Returns

ulCodePage (ULONG) – returns
Code page.

0	Error occurred
Other	Queue code page for the specified message queue.

Possible returns from WinGetLastError

PMERR_INVALID_HMQ (0x1002)	An invalid message-queue handle was specified.

Related Functions

- WinCpTranslateChar
- WinCpTranslateString
- WinQueryCpList
- WinSetCp

Example Code

This example returns the queue code page for the specified queue.

```
#define INCL_WINCOUNTRY
#include <OS2.H>

HMQ hmq;
/* . */
ULONG  cp;

cp = WinQueryCp(hmq);
```

WinQueryCpList

This function queries available code pages.

Syntax

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryCpList (HAB hab, ULONG ulcount, PULONG aulCodepage)
```

Parameters

hab (HAB) – input
Anchor-block handle.

ulcount (ULONG) – input
Maximum number of code pages returned.

aulCodepage (PULONG) – output
Code page list.

An array of *ulcount* elements, that contains a list of code pages available to the program.

For more information about code pages, see "Code Pages" in the *Presentation Manager Programming Reference Volume II*.

Returns

ulTotCount (ULONG) – returns
Total number of code pages available.

0	An error occurred
Other	Total number of code pages available.

Possible returns from WinGetLastError

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Related Functions

- WinCpTranslateChar
- WinCpTranslateString
- WinQueryCp
- WinSetCp

Example Code

This example queries available code pages.

```
#define INCL_WINCOUNTRY
#include <OS2.H>
#define maxcount 8
HAB hab;
/* . */
ULONG  aulCodepage[maxcount];


WinQueryCpList(hab,
               (ULONG)maxcount,
               (PULONG) aulCodepage);
```

WinQueryCursorInfo

This function obtains information about any current cursor.

Syntax

```
#define INCL_WINCURSORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinQueryCursorInfo  (HWND hwndDeskTop,
                          PCURSORINFO pcsriCursorInfo)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

pcsriCursorInfo (PCURSORINFO) – output
Cursor information.

The values are equivalent to the parameters of the WinCreateCursor function except that *ulrgf* never includes the CURSOR_SETPOS option.

The size and position of the cursor are returned in window coordinates relative to the window identified by the *hwnd* parameter of the structure.

Returns

rc (BOOL) – returns
Current-cursor indicator.

TRUE	Cursor exists
FALSE	Cursor does not exist, *pcsriCursorInfo* is not updated by this call.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinCreateCursor
- WinDestroyCursor
- WinShowCursor

Example Code

This example obtains information about any current cursor.

```
#define INCL_WINCURSORS
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND hwnd;    /* handle of window that has pointer captured */
CURSORINFO cursorinfo;


WinQueryCursorInfo(hwnd_DESKTOP,    /*  get cursor info  */
                   &cursorinfo);
```

WinQueryDesktopBkgnd

This function returns the desktop structure, which contains the information about the current state of the desktop background.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinQueryDesktopBkgnd (HWND hwndDesktop, PDESKTOP pdsk)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop window
Other	Specified desktop window.

pdsk (PDESKTOP) – output
Desktop-state structure.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Desktop-window status provided
FALSE	Desktop-window status not provided.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function allows an application to query the background information of the desktop window. This application must be acting as the OS/2 PM shell in place of the IBM supplied shell. If the IBM supplied shell is executing it maintains control of the background of the desktop window, and WinQueryDesktopBkgnd will have no effect on the desktop window background, but will indicate a successful return code.

Related Functions

- WinSetDesktopBkgnd

Example Code

This example uses WinQueryDesktopBkgnd to query the current desktop background bit map before setting it to a new bit map with WinSetDesktopBkgnd.

```
#define INCL_WINDESKTOP
#define INCL_WINWINDOWMGR
#include <OS2.H>
HAB hab;
HWND    hwndDeskTop;
DESKTOP DeskTopState;
HBITMAP  hbm;
HBITMAP hbm_user;


WinQueryDesktopBkgnd(HWND_DESKTOP,
                     &DeskTopState);

if (hbm_user != DeskTopState.hbm)
{

 DeskTopState.fl = SDT_LOADFILE;
              /* the szFile is used to load the bit map because */
              /* the fl parameter is set to SDT_LOADFILE.      */
 strcpy(DeskTopState.szFile,"fruit.bmp");
 DeskTopState.hbm = hbm_user;
 WinSetDesktopBkgnd(hwndDeskTop,
                    &DeskTopState);
}
```

WinQueryDesktopWindow

This function returns the desktop-window handle.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinQueryDesktopWindow  (HAB hab, HDC hdc)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hdc (HDC) – input
Device-context handle.

NULLHANDLE	Default device (the screen).

Returns

hwndDeskTop (HWND) – returns
Desktop-window handle.

NULLHANDLE	Error occurred
Other	Desktop-window handle.

Possible returns from WinGetLastError

PMERR_INV_HDC (0x207C)	An invalid device-context handle or (micro presentation space) presentation-space handle was specified.

Remarks

Only the screen device supports windowing.

Many of the calls that require a desktop-window handle accept HWND_DESKTOP instead. For example, WinCreateWindow accepts HWND_DESKTOP for the parent-window handle to create a main window that is a child of the desktop window.

Related Functions

- WinEnableWindow
- WinIsThreadActive
- WinIsWindow
- WinIsWindowEnabled

- WinQueryObjectWindow
- WinQueryWindowDC
- WinQueryWindowProcess
- WinQueryWindowRect
- WinWindowFromDC
- WinWindowFromID
- WinWindowFromPoint

Example Code

This function is used to find the desktop window handle. For most calls however, the parameter HWND_DESKTOP can be used.

```
#define INCL_WINDESKTOP
#include <OS2.H>
HAB hab;
HWND  hwndDeskTop;

hwndDeskTop = WinQueryDesktopWindow(hab,
                                    NULLHANDLE);
```

WinQueryDlgItemShort

This function converts the text of a dialog item into an integer value.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinQueryDlgItemShort (HWND hwndDlg, ULONG idItem,
                           PSHORT psResult, BOOL fSigned)
```

Parameters

hwndDlg (HWND) – input
Parent-window handle.

idItem (ULONG) – input
Identity of the child window whose text is to be converted.

It must be greater or equal to 0 and less or equal to 0xFFFF.

psResult (PSHORT) – output
Integer value resulting from the conversion.

fSigned (BOOL) – input
Sign indicator.

TRUE	Signed text. It is inspected for a minus sign (-).
FALSE	Unsigned text.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful conversion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function is useful for converting a numerical input field into a binary number for further processing. The text of a dialog item is assumed to be an ASCII string.

This function is valid for any window with children. However, it is typically used for dialog items in a dialog window.

Related Functions

- WinQueryDlgItemText
- WinQueryDlgItemTextLength
- WinQueryWindowText
- WinQueryWindowTextLength
- WinSetDlgItemShort
- WinSetDlgItemText
- WinSetWindowText

Example Code

This example gets the text from a Dialog Box entry field as an integer value.

```
#define INCL_WINDIALOGS

#include <OS2.H>

#define ID_ENTRYFLD 900

HAB hab;
HWND  hwnd;
ULONG  msg;
MPARAM mp1;
SHORT iconverted;

/* . */
 switch(msg)
  {
   case WM_INITDLG:

   case WM_COMMAND:
   switch(SHORT1FROMMP(mp1))
    {
     case DID_OK:
      WinQueryDlgItemShort(hwnd,
                           ID_ENTRYFLD,
                           &iconverted, /* integer result */
                           TRUE); /* Get the short      */

    }
   }
```

WinQueryDlgItemText

This function queries a text string in a dialog item.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
ULONG WinQueryDlgItemText (HWND hwndDlg, ULONG idItem, LONG lMaxText,
                           PSZ pszText)
```

Parameters

hwndDlg (HWND) – input
Parent-window handle.

idItem (ULONG) – input
Identity of the child window whose text is to be queried.

It must be greater or equal to 0 and less or equal to 0xFFFF.

lMaxText (LONG) – input
Length of *pszText*.

It must be greater or equal to 0.

pszText (PSZ) – output
Output string.

This is the text string that is obtained from the dialog item.

Returns

ulRetLen (ULONG) – returns
Actual number of characters returned.

0	Error occurred
Other	Actual number of characters returned, not including the null-terminating character. The maximum value is (*lMaxText*–1).

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function is valid for any window with children. However, it is typically used for dialog items in a dialog window.

Related Functions

- WinQueryDlgItemShort
- WinQueryDlgItemTextLength
- WinQueryWindowText
- WinQueryWindowTextLength
- WinSetDlgItemShort
- WinSetDlgItemText
- WinSetWindowText

Example Code

This example is the beginning of a function which processes the text which is displayed in the message text line.

```
#define INCL_WINDIALOGS
#include <OS2.H>
#define DID_MSGEDIT 900
void SelectMessageFromText(hwndDlg)
HWND    hwndDlg;
{
    char    szTemp[80];

    /* First get the edit text from the string */
    WinQueryDlgItemText(hwndDlg, DID_MSGEDIT, sizeof(szTemp),
            (PSZ)szTemp);
      /* . */
      /* . */
}
```

WinQueryDlgItemTextLength

This function queries the length of the text string in a dialog item, not including any null termination character.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
LONG WinQueryDlgItemTextLength  (HWND hwndDlg, ULONG idItem)
```

Parameters

hwndDlg (HWND) – input
Parent-window handle.

idItem (ULONG) – input
Identity of the child window whose text is to be queried.

It must be greater or equal to 0 and less or equal to 0xFFFF.

Returns

lRetLen (LONG) – returns
Length of text.

0	Error occurred
Other	Length of text.

Remarks

This function is valid for any window with children. However, it is typically used for dialog items in a dialog window.

Related Functions

- WinQueryDlgItemShort
- WinQueryDlgItemText
- WinQueryWindowText
- WinQueryWindowTextLength
- WinSetDlgItemShort
- WinSetDlgItemText
- WinSetWindowText

Example Code

This example is the beginning of a function which processes the text which is displayed in the message text line.

```
#define INCL_WINDIALOGS
#define INCL_DOSMEMMGR
#include <OS2.H>
#define DID_MSGEDIT 900
void SelectMessageFromText(hwndDlg)
HWND    hwndDlg;
{

char *szTemp;
LONG  length;

    /* First get the edit text from the string */

 length = WinQueryDlgItemTextLength(hwndDlg,
                                    DID_MSGEDIT);
   /* now we know the buffer size needed. */

DosAllocMem((PPVOID)szTemp,
            (ULONG)length,
             PAG_READ |
             PAG_WRITE |
             PAG_COMMIT);

 WinQueryDlgItemText(hwndDlg,
                     DID_MSGEDIT,
                     sizeof(szTemp),
                     (PSZ)szTemp);
      /* . */
      /* . */
}
```

WinQueryFocus

This function returns the focus window. It is NULLHANDLE if there is no focus window.

Syntax

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinQueryFocus  (HWND  hwndDeskTop)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Returns

hwndFocus (HWND) – returns
Focus-handle.

NULL	Error occurred or no focus window.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinEnablePhysInput
- WinFocusChange
- WinGetKeyState
- WinGetPhysKeyState
- WinSetFocus
- WinSetKeyboardStateTable

Example Code

This example checks to see if the menu has the focus.

```
#define INCL_WININPUT
#include <OS2.H>
#define SYS_MENU 900
HWND hwndFrame;

if (WinQueryFocus(HWND_DESKTOP) ==
    WinWindowFromID(hwndFrame, SYS_MENU))
{
 /* . */
}
```

WinQueryHelpInstance

This function enables the application to query the instance of the Help Manager associated with the application-supplied window handle.

Syntax

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinQueryHelpInstance  (HWND  hwndApp)
```

Parameters

hwndApp (HWND) – input
Handle of the application window.

Returns

hwndHelp (HWND) – returns
Help Manager window handle.

NULLHANDLE	No Help Manager instance is associated with the application window.
Other	Help Manager window handle.

Remarks

The Help Manager first traces the parent window chain until it is NULLHANDLE or HWND_DESKTOP. Then Help Manager traces the owner chain. If a parent of the owner window exists, the trace begins again with the parent chain.

The window chain will be traced until the Help Manager finds an instance of the Help Manager or until both the parent and owner windows are NULLHANDLE or HWND_DESKTOP.

Related Functions

- WinAssociateHelpInstance
- WinCreateHelpInstance
- WinCreateHelpTable
- WinDestroyHelpInstance
- WinLoadHelpTable

Example Code

This example shows the use of the WinQueryHelpInstance call during the processing of a WM_INITMENU message in order to obtain the handle for sending an HM_SET_ACTIVE_WINDOW message.

```
#define INCL_WIN
#include <os2.h>

MRESULT wm_initmenu( HWND hWnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
   /* Send message to establish the current window's parent      */
   /* as the active help window.                                 */
   WinSendMsg( WinQueryHelpInstance( hWnd ),
               HM_SET_ACTIVE_WINDOW,
               (MPARAM)WinQueryWindow( hWnd, QW_PARENT ),
               (MPARAM)WinQueryWindow( hWnd, QW_PARENT ) );

   /* Pass message on for default processing                     */
   return WinDefWindowProc( hWnd, ulMsg, mp1, mp2 );
}
```

WinQueryLboxCount

This macro returns the number of items in the List Box.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinQueryLboxCount (HWND hwndLbox)
```

Parameters

hwndLbox (HWND) – input
Listbox handle.

Returns

lRetNumIt (LONG) – returns
Number of items in the list box.

Remarks

This macro is defined as:

```
#define WinQueryLboxCount(hwndLbox)             \
        ((LONG)WinSendMsg(hwndLbox,             \
                          LM_QUERYITEMCOUNT,    \
                          (MPARAM)NULL,         \
                          (MPARAM)NULL))
```

This macro requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- LM_QUERYITEMCOUNT

Example Code

This example uses WinQueryLboxCount to find the number of list box items and selects them all.

```
#define INCL_WINLISTBOXES
#define INCL_WINWINDOWMGR
#include <OS2.H>

LONG cWindows;
HWND hwndWindowLB;

cWindows = WinQueryLboxCount(hwndWindowLB);

/* Loop through all windows, selecting them all */
while (cWindows)
{
   WinSendMsg(hwndWindowLB,
              LM_SELECTITEM,
              (MPARAM)--cWindows,
              (MPARAM)TRUE);
}
```

WinQueryLboxItemText

This macro fills the buffer with the text of the indexed item. It returns the length of the text.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinQueryLboxItemText (HWND hwndLbox, SHORT index, PSZ psz,
                           SHORT cchMax)
```

Parameters

hwndLbox (HWND) – input
List box handle.

index (SHORT) – input
Index of the listbox item.

psz (PSZ) – input
Pointer to a null terminated string.

cchMax (SHORT) – input
Maximum number of characters allocated to the string.

Returns

lRetTxtL (LONG) – returns
Actual text length copied.

Remarks

This macro is defined as:

```
#define WinQueryLboxItemText(hwndLbox, index, psz, cchMax)  \
        ((LONG)WinSendMsg(hwndLbox,                         \
                          LM_QUERYITEMTEXT,                 \
                          MPFROM2SHORT((index), (cchMax)),  \
                          MPFROMP(psz)))
```

This macro requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- LM_INSERTITEM

Example Code

This example uses WinQueryLboxItemText to copy all of the list box items into a buffer.

```
#define INCL_WINLISTBOXES
#define INCL_WINWINDOWMGR
#include <OS2.H>

LONG  cWindows;
char *szTemp;
HWND hwndLB;
SHORT maxchar, index = 0;

cWindows = WinQueryLboxCount(hwndLB);

/* allocate a buffer for cWindows items. */

DosAllocMem((PPVOID)&szTemp,
            (ULONG)cWindows*256*sizeof(char),
             PAG_READ |
             PAG_WRITE |
             PAG_COMMIT);

/* loop through all of the items;  copying each */
/* one the buffer.                              */

while (index <= cWindows)
{
   maxchar = WinQueryLboxItemTextLength(hwndLB,index);
   WinQueryLboxItemText(hwndLB,
                        index++,
                        szTemp,
                        maxchar);
   (*szTemp)+=maxchar*sizeof(char); /* increment pointer by number */
                                    /* of bytes copied. */
}
```

WinQueryLboxItemTextLength

This macro returns the length of the text of the indexed item in the List Box.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
SHORT WinQueryLboxItemTextLength  (HWND hwndLbox, SHORT index)
```

Parameters

hwndLbox (HWND) – input
Listbox handle.

index (SHORT) – input
Index of the item in the List Box.

Returns

sRetLen (SHORT) – returns
Text length of the indexed item.

Remarks

This macro is defined as:

```
#define WinQueryLboxItemTextLength(hwndLbox, index)  \
        ((SHORT)WinSendMsg(hwndLbox,                  \
                           LM_QUERYITEMTEXTLENGTH,    \
                           MPFROMSHORT(index),        \
                           (MPARAM)NULL))
```

This macro requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- LM_QUERYITEMTEXTLENGTH

Example Code

This example uses WinQueryLboxItemText to copy all of the list box items into a buffer.

```
#define INCL_WINLISTBOXES
#define INCL_WINWINDOWMGR
#include <OS2.H>

LONG  cWindows;
char *szTemp;
HWND  hwndLB;
SHORT maxchar, index = 0;

cWindows = WinQueryLboxCount(hwndLB);

/* allocate a buffer for cWindows items. */

DosAllocMem((PPVOID)&szTemp,
            (ULONG)cWindows*256*sizeof(char),
             PAG_READ |
             PAG_WRITE |
             PAG_COMMIT);

/* loop through all of the items;  copying each */
/* one the buffer.                              */

while (index <= cWindows)
{
   maxchar = WinQueryLboxItemTextLength(hwndLB,index);
   WinQueryLboxItemText(hwndLB,
                        index++,
                        szTemp,
                        maxchar);
   (*szTemp)+=maxchar*sizeof(char); /* increment pointer by number */
                                    /* of bytes copied. */
}
```

WinQueryLboxSelectedItem

This macro returns the index of the selected item in the List Box (for single selection only).

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinQueryLboxSelectedItem  (HWND hwndLbox)
```

Parameters

hwndLbox (HWND) – input
List box handle.

Returns

lRetIndex (LONG) – returns
Index of the selected item.

Remarks

This macro is defined as:

```
#define WinQueryLBoxSelectedItem (hwndLbox)          \
        ((LONG)WinSendMsg(hwndLbox,                  \
                          LM_QUERYSELECTION,         \
                          MPFROMLONG(LIT_FIRST),     \
                          (MPARAM)NULL))
```

This macro requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- LM_QUERYSELECTION

Example Code

This example copies the text from the selected item in a list box to a buffer. Note that while WinQueryLboxSelectedItem returns a LONG value, WinQueryLboxItemText takes a SHORT parameter.

```
#define INCL_WINLISTBOXES
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwndLB;
LONG index;
char szTemp[256];

index = WinQueryLboxSelectedItem(hwndLB);

WinQueryLboxItemText(hwndLB,
                     (SHORT) index,
                     szTemp,
                     WinQueryLboxItemTextLength(hwndLB,index));
```

WinQueryMsgPos

This function returns the pointer position, in screen coordinates, when the last message obtained from the current message queue is posted.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinQueryMsgPos  (HAB hab, PPOINTL pptl)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pptl (PPOINTL) – output
Pointer position in screen coordinates.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The pointer position is the same as that in the *ptl* parameter of a QMSG structure.

To obtain the current position of the pointer, use the WinQueryPointerPos function.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgTime
- WinQueryQueueInfo

- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example returns position and time of the the last message obtained from the current message queue.

```
#define INCL_WINMESSAGEMGR
#define INCL_WINDIALOGS
#include <OS2.H>
#include <stdio.h>
HAB hab;
POINTL ptl;
CHAR szMsg[100];
HWND hwnd;
ULONG ulTime;


WinQueryMsgPos(hab, &ptl);


ulTime = WinQueryMsgTime(hab);



 sprintf(szMsg, "x = %ld   y = %ld\n\ntime = %ld",
        ptl.x, ptl.y, ulTime);
 WinMessageBox(HWND_DESKTOP,
     hwnd,                      /* 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 */
```

WinQueryMsgTime

This function returns the message time for the last message retrieved by the WinGetMsg or WinPeekMsg functions from the current message queue.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryMsgTime  (HAB hab)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

ulTime (ULONG) – returns
Time in milliseconds.

Remarks

The message time is the time the message is posted, measured in milliseconds, from the time the system is started. Its value is the same as that in the *time* parameter of the QMSG structure.

To calculate time delays between messages, the time of the first message is subtracted from the time of the second message.

Time values do not always increase because the value is the number of milliseconds since the system was started, and the system accumulator for this count can wrap through zero.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos

- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example returns position and time of the the last message obtained from the current message queue.

```
#define INCL_WINMESSAGEMGR
#define INCL_WINDIALOGS
#include <OS2.H>
#include <stdio.h>
HAB hab;
POINTL ptl;
CHAR szMsg[100];
HWND hwnd;
ULONG ulTime;


WinQueryMsgPos(hab, &ptl);


ulTime = WinQueryMsgTime(hab);



 sprintf(szMsg, "x = %ld   y = %ld\n\ntime = %ld",
         ptl.x, ptl.y, ulTime);
 WinMessageBox(HWND_DESKTOP,
     hwnd,                      /* 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 */
```

WinQueryObject

The WinQueryObject function returns a handle to the given object

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
HOBJECT WinQueryObject  (PSZ pObjectID)
```

Parameters

pObjectID (PSZ) – input

The ObjectID of an existing object.

The ObjectID of an existing object, for example <LOCATION_DESKTOP>, or alternatively the fully qualified filename of any file or directory.

Returns

hObject (HOBJECT) – returns

MRESULT.

Persistent object handle, or NULLHANDLE if the object does not exist or could not be awakened.

Remarks

This function allows you to obtain the persistant object handle for any file object, by passing the fully qualified filename. Similarly any objects' handle can be retrieved if its ObjectID string is passed. Once a program has an object handle, it is able to change the objects state by using the WinSetObjectData function or delete the object using the WinDestroyObject function. Note that valid ObjectID strings must always start with the "<" character and be terminated by the ">" character, and are thus invalid file system names.

Using HOBJECT for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

Related Functions

- WinCreateObject
- WinDestroyObject
- WinSetObjectData

WinQueryObjectPath

This function is specific to version 3, or higher, of the OS/2 operating system.

This function returns the directory specification of a given object handle.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinQueryObjectPath  (HOBJECT hObject, PSZ pszPathname,
                          ULONG ulSize)
```

Parameters

hObject (HOBJECT) – input
Object handle of the object whose file/directory specification is to be returned.

pszPathname (PSZ) – output
Memory allocated by caller in which directory specification is written.

ulSize (ULONG) – input
Number of bytes pointed to by *pszPathname*.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS (0x1208)
An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

Remarks

This function is used to find the file/directory specification of a given object handle.

Using HOBJECT for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

Example Code

This example finds the file/directory of the <WP_OS2SYS> object..

```
#define INCL_WINWORKPLACE
#include <os2.h>

HOBJECT hObject;
CHAR    szPath[CCHMAXPATH + 1];
BOOL    fSuccess;

hObject = WinQueryObject("<WP_OS2SYS>");
if (hObject != NULL)
{
  /* WinQueryObject was successful */

  fSuccess = WinQueryObjectPath(hObject,szPath,sizeof(szPath));
  if (fSuccess)
  {
    /* WinQueryObjectPath was successful */
  }
  else
  {
    /* WinQueryObjectPath failed */
  }
}
else
{
  /* WinQueryObject fialed */
}
```

WinQueryObjectWindow

This function returns the desktop object window handle.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinQueryObjectWindow (HWND hwndDesktop)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Returns

hwndObject (HWND) – returns
Object-window handle.

NULLHANDLE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

Any window created as a descendant of *hwndObject* is an object window.

Related Functions

- WinEnableWindow
- WinIsThreadActive
- WinIsWindow
- WinIsWindowEnabled
- WinQueryDesktopWindow
- WinQueryWindowDC
- WinQueryWindowProcess
- WinQueryWindowRect
- WinWindowFromDC
- WinWindowFromID
- WinWindowFromPoint

Example Code

This example calls WinQueryObjectWindow to return the desktop object window handle. All windows created as descendants of this object window—as in the example—will be object windows.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#include <os2.h>

HWND    hwndObject;     /* desktop object window                */
HWND    hwndObject1;    /* descendant object window             */
USHORT  WindowId;
hwndObject = WinQueryObjectWindow(HWND_DESKTOP);

/* create object window */
hwndObject1 = WinCreateWindow(hwndObject,  /* parent window     */
                      "NewClass",    /* class name              */
                      "new button",  /* window text             */
                      WS_VISIBLE,    /* window style            */
                      0, 0,          /* position (x,y)          */
                      200, 100,      /* size (width,height)     */
                      0L,            /* owner window            */
                      HWND_TOP,      /* sibling window          */
                      WindowId,      /* window id               */
                      NULL,          /* control data            */
                      NULL);         /* presentation parms      */
```

WinQueryPointer

This function returns the pointer handle for *hwndDeskTop*.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HPOINTER WinQueryPointer (HWND hwndDeskTop)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Returns

hptrPointer (HPOINTER) – returns
Pointer handle.

NULLHANDLE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinQuerySysPointerData
- WinSetPointer
- WinSetPointerPos
- WinSetSysPointerData
- WinShowPointer

Example Code

This example obtains the pointer handle from the desktop window handle.

```
#define INCL_WINPOINTERS
#define INCL_WINDESKTOP
#include <OS2.H>

HAB      hab;
HPOINTER hpointer;

hpointer = WinQueryPointer(HWND_DESKTOP);
```

WinQueryPointerInfo

This function returns pointer information.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinQueryPointerInfo (HPOINTER hptr, PPOINTERINFO pptriPointerInfo)
```

Parameters

hptr (HPOINTER) – input
Pointer handle.

pptriPointerInfo (PPOINTERINFO) – output
Pointer-information structure.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HPTR (0x101B)	An invalid pointer handle was specified.

Remarks

The pointer information structure contains information such as the bit-map handle of the pointer and action point coordinates. The values returned for the *xHotSpot* and the *yHotSpot* parameters are in units relative to the size of the system icon or system pointer.

For example, if the application creates a pointer out of a bit map xWide units wide and positions the x-coordinate of the pointer's action point at xHot, then this function will return the value of the *xHotSpot* as:

```
xHotspot = (xHot * SystemPointerWidth) / xWide
```

where SystemPointerWidth can be obtained by using the WinQuerySysValue function.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerPos
- WinQuerySysPointer
- WinQuerySysPointerData
- WinSetPointer
- WinSetPointerPos
- WinSetSysPointerData
- WinShowPointer

Example Code

This example uses the WinQueryPointerInfo call to obtain the bit-map handle of the color bit map.

```
#define INCL_WINPOINTERS
#define INCL_WINDESKTOP
#include <OS2.H>

HAB         hab;
HPOINTER    hpointer;
POINTERINFO pointerinfo;
HBITMAP     hbm;            /* Bit-map handle of color bit map  */

hpointer = WinQueryPointer(HWND_DESKTOP);

WinQueryPointerInfo(hpointer,
                    &pointerinfo);

hbm = pointerinfo.hbmColor;
```

WinQueryPointerPos

This function returns the pointer position.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinQueryPointerPos  (HWND hwndDeskTop, PPOINTL pptlPoint)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

pptlPoint (PPOINTL) – output
Pointer position in screen coordinates.

Returns

rc (BOOL) – returns
Pointer position returned indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The WinQueryMsgPos is used to get the pointer position of the last message obtained by means of the WinGetMsg or WinPeekMsg functions.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo

- WinQuerySysPointer
- WinQuerySysPointerData
- WinSetPointer
- WinSetPointerPos
- WinSetSysPointerData
- WinShowPointer

Example Code

This example displays the pointer position.

```
#define INCL_WINWINDOWMGR
#define INCL_WINPOINTERS
#include <OS2.H>

HWND   hwndClient;
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 */
```

WinQueryPresParam

This function queries the values of presentation parameters for a window

Syntax

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryPresParam  (HWND hwnd, ULONG idAttrType1,
                          ULONG idAttrType2, PULONG pidAttrTypeFound,
                          ULONG cbAttrValueLen, PPVOID pAttrValue,
                          ULONG flOptions)
```

Parameters

hwnd (HWND) – input
Window handle.

idAttrType1 (ULONG) – input
First attribute type identity.

This identifies the first presentation parameter attribute to be queried It can be zero to reference no presentation parameter attribute.

idAttrType2 (ULONG) – input
Second attribute type identity.

This identifies the second presentation parameter attribute to be queried. It can be zero to reference no presentation parameter attribute.

pidAttrTypeFound (PULONG) – in/out
Attribute type identity found.

This identifies which of the presentation parameter attributes *idAttrType1* and *idAttrType2* has been found. This parameter can be passed as NULL (if, for example, only one attribute is being queried).

cbAttrValueLen (ULONG) – input
Byte count of the size of the *pAttrValue* parameter.

pAttrValue (PPVOID) – output
Attribute value.

The value of the presentation parameter attribute found.

flOptions (ULONG) – input
Options controlling the query.

Any of the following values can be ORed together:

QPF_NOINHERIT	For the purposes of this query, presentation parameters are not inherited from the owners of the window specified by *hwnd*. If not specified (default), presentation parameters are inherited.
QPF_ID1COLORINDEX	*idAttrType1* refers to a color index presentation parameter attribute, the value of which, if found, is to be converted to RGB before being passed back in *pAttrValue*.
QPF_ID2COLORINDEX	*idAttrType2* refers to a color index presentation parameter attribute, the value of which, if found, is to be converted to RGB before being passed back in *pAttrValue*.
QPF_PURERGBCOLOR	Specifies that either or both of *idAttrType1* and *idAttrType2* reference RGB color, and that these colors must be pure. This is necessary when specifying text foreground and background colors. This is applied after QPF_ID1COLORINDEX and QPF_ID2COLORINDEX.

Returns

cbRetLen (ULONG) – returns
Length of presentation parameter value. passed back.

Zero	Presentation parameter not found or error occurred
Other	Length of presentation parameter value passed back in *pAttrValue*.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

Two presentation parameter attribute identities can be passed, and both will be searched for along the chain of owners of the window *hwnd* (subject to QPF_NOINHERIT). The first one found satisfies the query. If both *idAttrType1* and *idAttrType2* are present for the same window, *idAttrType1* takes precedence.

If the presentation parameter attribute value is too long to fit in the *pAttrValue* buffer provided, it is truncated, and the number of bytes copied is returned in *cbRetLen*. (See also WinSetPresParam and WinRemovePresParam).

Related Functions

- WinDrawBitmap
- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinFillRect
- WinGetSysBitmap
- WinInvertRect
- WinRemovePresParam
- WinScrollWindow
- WinSetPresParam

Example Code

This example queries the disable-foreground attribute; if it is a valid attribute of the window, it is removed via WinRemovePresParam.

```
#define INCL_WINSYS
#include <OS2.H>

HWND  hwnd;
ULONG AttrFound;
ULONG AttrValue[32];
ULONG cbRetLen;

cbRetLen = WinQueryPresParam(hwnd,
                             PP_DISABLEDFOREGROUNDCOLORINDEX,
                             0,
                             &AttrFound,
                             sizeof(AttrValue),
                             &AttrValue,
                             QPF_ID1COLORINDEX | QPF_NOINHERIT);

if(PP_DISABLEDFOREGROUNDCOLORINDEX == AttrFound);

WinRemovePresParam(hwnd,
                   PP_DISABLEDFOREGROUNDCOLORINDEX);
```

WinQueryQueueInfo

This function returns the information for the specified queue.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinQueryQueueInfo (HMQ hmq, PMQINFO pmqiMqinfo, ULONG cbCopied)
```

Parameters

hmq (HMQ) – input
Queue handle.

It must be created by a previous call to WinCreateMsgQueue or HMQ_CURRENT.

pmqiMqinfo (PMQINFO) – output
Message queue information structure to contain the queue information.

cbCopied (ULONG) – input
Size of message queue information structure that is provided (in bytes).

Specifies the maximum number of bytes to be copied into the *pmqiMqinfo* parameter. This should be the size of an MQINFO structure.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueStatus

- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example retrieves the process identity from a queue by passing the queue handle to WinQueryQueueInfo.

```
#define INCL_WINMESSAGEMGR
#include <OS2.H>

HMQ    hmq;
MQINFO mqinfo;
PID    pid;

WinQueryQueueInfo(hmq,
                  &mqinfo,
                  sizeof(MQINFO));

pid = mqinfo.pid;
```

WinQueryQueueStatus

This function returns a code indicating the status of the message queue associated with the caller.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryQueueStatus  (HWND hwndDesktop)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Desktop-window handle returned by WinQueryDesktopWindow.

Returns

flStatus (ULONG) – returns
Status information.

Summary
Summary of message types existing on the queue.

This field contains a combination of the following values:

QS_KEY	An input event (keyboard or journaling) has caused a WM_CHAR message to be placed in the queue.
QS_MOUSE	An input event has caused a WM_MOUSEMOVE, WM_BUTTON1UP, WM_BUTTON1DOWN, WM_BUTTON1DBLCLK, WM_BUTTON2UP, WM_BUTTON2DOWN, WM_BUTTON2DBLCLK, WM_BUTTON3UP, WM_BUTTON3DOWN, or WM_BUTTON3DBLCLK message to be placed in the queue.
QS_MOUSEBUTTON	An input event has caused a WM_BUTTON1UP, WM_BUTTON1DOWN, WM_BUTTON1DBLCLK, WM_BUTTON2UP, WM_BUTTON2DOWN, WM_BUTTON2DBLCLK, WM_BUTTON3UP, WM_BUTTON3DOWN, or WM_BUTTON3DBLCLK message to be placed in the queue.
QS_MOUSEMOVE	An input event has caused a WM_MOUSEMOVE message to be placed in the queue.

QS_TIMER	A timer event has caused a WM_TIMER message to be placed in the queue.
QS_PAINT	A WM_PAINT message is available.
QS_SEM1	A WM_SEM1 message is available.
QS_SEM2	A WM_SEM2 message is available.
QS_SEM3	A WM_SEM3 message is available.
QS_SEM4	A WM_SEM4 message is available.
QS_POSTMSG	A message has been posted to the queue. Note that this message is probably not one of the messages listed above, but could be a WM_CHAR, WM_MOUSEMOVE or similar message if an application has posted one of these. In this case, the corresponding input status flag (QS_KEY, QS_MOUSE, and so on) is not set.
QS_SENDMSG	A message has been sent by another application to a window associated with the current queue.

Added

Message type additions.

Message types added to the queue since the last use of this function. The value of this field is a subset of the *Summary* field.

Remarks

This function is an efficient method for determining whether input is available for processing by the WinGetMsg or WinPeekMsg functions.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode

- WinSetSynchroMode
- WinWaitMsg

Related Messages

- WM_BUTTON1UP
- WM_BUTTON1DOWN
- WM_BUTTON1DBLCLK
- WM_BUTTON2UP
- WM_BUTTON2DOWN
- WM_BUTTON2DBLCLK
- WM_BUTTON3UP
- WM_BUTTON3DOWN
- WM_BUTTON3DBLCLK
- WM_CHAR
- WM_MOUSEMOVE
- WM_PAINT
- WM_SEM1
- WM_SEM2
- WM_SEM3
- WM_SEM4
- WM_TIMER

Example Code

This example uses the WinQueryQueueStatus to see if a WM_MOUSEMOVE message has been placed in the queue.

```
#define INCL_WINMESSAGEMGR
#include <OS2.H>

if(WinQueryQueueStatus(HWND_DESKTOP) == QS_MOUSEMOVE)
{
   /* . */
   /* . */
}
```

WinQuerySessionTitle

This function obtains the title under which a specified application is started, or is added to the Window List.

Syntax

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQuerySessionTitle  (HAB hab, ULONG ulSession, PSZ pszTitle,
                             ULONG ulTitlelen)
```

Parameters

hab (HAB) – input
Anchor-block handle.

ulSession (ULONG) – input
Session identity of application whose title is requested.

0	Use the session identity of the caller
Other	Use the specified session identity.

pszTitle (PSZ) – output
Window List title.

This is the title of the application with a process identity, if the application is present in the Window List.

ulTitlelen (ULONG) – input
Maximum length of data returnable, in bytes.

If the *pszTitle* parameter is longer than this value, the title is truncated. However, the terminating null character is left at the end of the string. The maximum number of title characters copied is (*ulTitlelen*–1).

Returns

rc (ULONG) – returns
Return code.

0	Successful completion
Other	Error occurred.

Remarks

This function is useful when an application uses the same name in its window title (and in its entry in the Window List) as the end user invokes to start the application. This provides a visual link for the end user.

If this function is used after a Window List entry is created for the application, the title in the Window List entry is obtained. (See also WinQueryTaskTitle.)

Related Functions

- WinAddSwitchEntry
- WinChangeSwitchEntry
- WinCreateSwitchEntry
- WinQuerySwitchEntry
- WinQuerySwitchHandle
- WinQuerySwitchList
- WinQueryTaskSizePos
- WinQueryTaskTitle
- WinRemoveSwitchEntry
- WinSwitchToProgram

Example Code

This example calls WinQuerySessionTitle to retrieve the application's title, and then sets the title bar of the frame window to that title.

```
#define INCL_WINMESSAGEMGR
#define INCL_WINWINDOWMGR
#include <OS2.H>

HAB  hab;
HWND hwndFrame, hwndClient;
CHAR szTitle[MAXNAMEL + 1];

WinQuerySessionTitle(hab,
                     0,
                     szTitle,
                     sizeof(szTitle));

hwndFrame = WinQueryWindow(hwndClient,
                           QW_PARENT); /* get handle of parent, */
                                       /* which is frame window. */
WinSetWindowText(hwndFrame, szTitle);
```

WinQuerySwitchEntry

This function obtains a copy of the Window List data for a specific application.

Syntax

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQuerySwitchEntry  (HSWITCH hswitchSwitch,
                            PSWCNTRL pswctlSwitchData)
```

Parameters

hswitchSwitch (HSWITCH) – input
Handle to the Window List entry.

This can be obtained using the WinQuerySwitchHandle function.

pswctlSwitchData (PSWCNTRL) – output
Switch control data.

Contains information about the specified Window List entry. The *hprog* field contains the program handle used to start the program.

Returns

rc (ULONG) – returns
Return code.

0	Successful completion
Other	Error occurred.

Remarks

This function is available to PM and non-PM applications.

Related Functions

- WinAddSwitchEntry
- WinChangeSwitchEntry
- WinCreateSwitchEntry
- WinQuerySessionTitle
- WinQuerySwitchHandle
- WinQuerySwitchList
- WinQueryTaskSizePos
- WinQueryTaskTitle
- WinRemoveSwitchEntry
- WinSwitchToProgram

Example Code

This example calls WinQuerySwitchHandle to get the Task List handle of a frame window, and then calls WinQuerySwitchEntry to retrieve information about that application.

```
#define INCL_WINSWITCHLIST
#include <OS2.H>

HAB     hab;
HWND    hwndFrame;
HSWITCH hswitch;
SWCNTRL swctl;

hswitch = WinQuerySwitchHandle(hwndFrame, 0);
WinQuerySwitchEntry(hswitch, &swctl);
```

WinQuerySwitchHandle

This function obtains the Window List handle belonging to a window.

Syntax

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HSWITCH WinQuerySwitchHandle  (HWND hwnd, PID idProcess)
```

Parameters

hwnd (HWND) – input
Window handle of an application.

Window handle of an application running in the OS/2 session for which the Window List handle is required.

NULLHANDLE	Application is not an OS/2 application
Other	Window handle of an application.

idProcess (PID) – input
Process identity of the application.

Returns

hswitchSwitch (HSWITCH) – returns
Switch list handle for the specified application.

NULLHANDLE	Application is not in the switch list, or an error occurred
Other	Switch list handle.

Remarks

If both a window handle and a process identity are supplied, they must be consistent.

If the window handle is NULL and the process identity supplied cannot be found in the switch list then the switch handle returned is the handle for the most proximal ancestor process. Once the switch list handle is obtained, it may be used in various other calls to manipulate the switch list entry or the program which it references.

Related Functions

- WinAddSwitchEntry
- WinChangeSwitchEntry
- WinCreateSwitchEntry
- WinQuerySessionTitle
- WinQuerySwitchEntry

- WinQuerySwitchList
- WinQueryTaskSizePos
- WinQueryTaskTitle
- WinRemoveSwitchEntry
- WinSwitchToProgram

Example Code

This example calls WinQuerySwitchHandle to get the Task List handle of a frame window, and then calls WinQuerySwitchEntry to retrieve information about that application.

```
#define INCL_WINSWITCHLIST
#include <OS2.H>

HAB     hab;
HWND    hwndFrame;
HSWITCH hswitch;
SWCNTRL swctl;

hswitch = WinQuerySwitchHandle(hwndFrame, 0);
WinQuerySwitchEntry(hswitch, &swctl);
```

WinQuerySwitchList

This function obtains information about the entries in the Window List.

Syntax

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQuerySwitchList (HAB hab, PSWBLOCK pswblkBlock,
                          ULONG ulLength)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pswblkBlock (PSWBLOCK) – in/out
Switch entries block.

Contains a description of all the entries in the current switch list. This is held in a SWBLOCK structure, which has a count of the number of switch list entries, plus a record for each entry containing data such as the process and session identities, the icon handle, and the window handle for the running program.

NULL No information returned; the return parameter however contains the total number of switch list entries.

Other Switch entries block.

ulLength (ULONG) – input
Maximum length of data returnable in bytes.

This is the maximum length in bytes of the data that can be returned in the *pswblkBlock* parameter.

0 No information returned, however the return parameter contains the total number of switch list entries.

Other Maximum length of data returnable.

Returns

ulCount (ULONG) – returns
Total number of switch list entries present in the system.

0 Error occurred
Other Total number of switch list entries present in the system.

Remarks

It is possible to obtain information about all the programs currently executing in a single operation, with one array entry for each program.

Related Functions

- WinAddSwitchEntry
- WinChangeSwitchEntry
- WinCreateSwitchEntry
- WinQuerySwitchEntry
- WinQuerySwitchHandle
- WinRemoveSwitchEntry
- WinSwitchToProgram

Example Code

This example determines the number of items in the Task List, allocates memory for the required buffer, and then calls WinQuerySwitchList again to fill the buffer with the information about each item in the Task List.

```
#define INCL_DOSMEMMGR
#define INCL_DOSERRORS
#define INCL_WINSWITCHLIST  /* Or INCL_WIN, INCL_PM */
#include <OS2.H>

#define MEMPOOL  40000   /* Size of local memory pool area */

HAB      hab        = NULLHANDLE;
HWND     hwndFrame  = NULLHANDLE;
ULONG    cbItems    = 0,            /* Number of items in list */
         ulBufSize  = 0;            /* Size of buffer for information */
PVOID    pvBase     = NULL;         /* Pointer to local memory pool */
PSWBLOCK pswblk     = NULL;         /* Pointer to information returned */
APIRET   rc         = NO_ERROR;     /* Return code from Dos APIs */

     /* Allocate a large block of memory (uncommitted) and make
        it available for suballocation.  This allows the system
        to commit memory only when it is actually needed.       */

rc = DosAllocMem( &pvBase, MEMPOOL, PAG_READ | PAG_WRITE );
rc = DosSubSetMem( pvBase, DOSSUB_INIT | DOSSUB_SPARSE_OBJ, MEMPOOL );

     /* Determine the number of items in the list and calculate
        the size of the buffer needed.                          */

cbItems = WinQuerySwitchList(hab, NULL, 0);
ulBufSize = (cbItems * sizeof(SWENTRY)) + sizeof(HSWITCH);

     /* Allocate the buffer from our memory pool */

rc = DosSubAllocMem( pvBase, (PVOID) &pswblk, ulBufSize);

   /* Call WinQuerySwitchList again to fill our buffer with information */

cbItems = WinQuerySwitchList(hab, pswblk, ulBufSize);
```

WinQuerySysColor

This function returns the system color.

Syntax

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinQuerySysColor (HWND hwndDesktop, LONG clr, LONG lReserved)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

clr (LONG) – input
System color-index value.

Must be one of the SYSCLR_* index values.

lReserved (LONG) – input
Reserved value, must be 0.

Returns

lRgbColor (LONG) – returns
RGB value.

i1.RGB (red-green-blue) RGB value corresponding to the *clr* parameter.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Remarks

This function returns the color value that corresponds to the specified color index of the specified color palette.

The following are the available SYSCLR_* values:

System Color Index	Default Color	Default RGB Values
SYSCLR_ACTIVEBORDER	Pale yellow	255 255 128
SYSCLR_ACTIVETITLE	Teal	64 128 128
SYSCLR_ACTIVETITLETEXT	White	255 255 255
SYSCLR_ACTIVETITLETEXTBGND	Teal	64 128 128
SYSCLR_APPWORKSPACE	Off-white	255 255 224
SYSCLR_BACKGROUND	Light gray	204 204 204
SYSCLR_BUTTONDARK	Dark gray	128 128 128
SYSCLR_BUTTONDEFAULT	Black	0 0 0
SYSCLR_BUTTONLIGHT	White	255 255 255
SYSCLR_BUTTONMIDDLE	Light gray	204 204 204
SYSCLR_DIALOGBACKGROUND	Light gray	204 204 204
SYSCLR_ENTRYFIELD	Pale yellow	255 255 204
SYSCLR_FIELDBACKGROUND	Light gray	204 204 204
SYSCLR_HELPBACKGROUND	White	255 255 255
SYSCLR_HELPHILITE	Blue green	0 128 128
SYSCLR_HELPTEXT	Dark blue	0 0 128
SYSCLR_HILITEBACKGROUND	Dark gray	128 128 128
SYSCLR_HILITEFOREGROUND	White	255 255 255
SYSCLR_ICONTEXT	Black	0 0 0
SYSCLR_INACTIVEBORDER	Light gray	204 204 204
SYSCLR_INACTIVETITLE	Light gray	204 204 204
SYSCLR_INACTIVETITLETEXT	Dark gray	128 128 128
SYSCLR_INACTIVETITLETEXTBGND	Light gray	204 204 204
SYSCLR_MENU	Light gray	204 204 204
SYSCLR_MENUDISABLEDTEXT	Dark gray	128 128 128
SYSCLR_MENUHILITE	Black	0 0 0
SYSCLR_MENUHILITEBGND	Light grey	204 204 204
SYSCLR_MENUTEXT	Black	0 0 0
SYSCLR_OUTPUTTEXT	Black	0 0 0
SYSCLR_PAGEBACKGROUND	White	255 255 255
SYSCLR_SCROLLBAR	Pale gray	192 192 192
SYSCLR_SHADOW	Dark gray	128 128 128
SYSCLR_SHADOWHILITEBGND	Dark gray	128 128 128
SYSCLR_SHADOWHILITEFGND	White	255 255 255

System Color Index	Default Color	Default RGB Values
SYSCLR_SHADOWTEXT	Dark gray	128 128 128
SYSCLR_TITLEBOTTOM	Dark gray	128 128 128
SYSCLR_TITLETEXT	White	255 255 255
SYSCLR_WINDOW	White	255 255 255
SYSCLR_WINDOWFRAME	Dark gray	128 128 128
SYSCLR_WINDOWSTATICTEXT	Blue	0 0 128
SYSCLR_WINDOWTEXT	Black	0 0 0

Related Functions

- WinSetSysColors

Example Code

This example uses the WinQuerySysColor to find the RGB index of the system push button, SYSCLR_BUTTONDEFAULT.

```
#define INCL_WINSYS
#define INCL_WINDESKTOP
#include <OS2.H>

HAB  hab;
LONG lRgbColor;

lRgbColor = WinQuerySysColor(HWND_DESKTOP,
                             SYSCLR_BUTTONDEFAULT,
                             0L);
```

WinQuerySysModalWindow

This function returns the current system modal window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinQuerySysModalWindow (HWND hwndDesktop)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Returns

hwndSysModal (HWND) – returns
Handle of system modal window.

NULLHANDLE	No system modal window
Other	Handle of system modal window.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

For a full description of the operation of the system modal window, see the WinSetSysModalWindow function.

Related Functions

- WinSetSysModalWindow

Example Code

This example uses the WinQuerySysModalWindow to find the handle of the system modal window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HAB  hab;
HWND hwndDeskTop, hwndSysModal;
LONG lRgbColor;

/*  Input processing can enter a "system modal" state. In */
/*  this state, all pointing  device and keyboard input   */
/*  is directed to a special window, known as the         */
/*  system-modal window. Typically, this will be a dialog */
/*  window requiring input.                               */

hwndSysModal = WinQuerySysModalWindow(hwndDeskTop);
```

WinQuerySysPointer

This function returns the system-pointer handle.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HPOINTER WinQuerySysPointer (HWND hwndDeskTop, LONG lIdentifier,
                             BOOL fCopy)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

lIdentifier (LONG) – input
System-pointer identifier.

SPTR_ARROW	Arrow pointer
SPTR_TEXT	Text I-beam pointer
SPTR_WAIT	Hourglass pointer
SPTR_SIZE	Size pointer
SPTR_MOVE	Move pointer
SPTR_SIZENWSE	Downward-sloping, double-headed arrow pointer
SPTR_SIZENESW	Upward-sloping, double-headed arrow pointer
SPTR_SIZEWE	Horizontal, double-headed arrow pointer
SPTR_SIZENS	Vertical, double-headed arrow pointer
SPTR_APPICON	Standard application icon pointer
SPTR_ICONINFORMATION	Information icon pointer
SPTR_ICONQUESICON	Question mark icon pointer
SPTR_ICONERROR	Exclamation mark icon pointer
SPTR_ICONWARNING	Warning icon pointer
SPTR_ILLEGAL	Illegal operation icon pointer
SPTR_FILE	Single file icon pointer
SPTR_MULTFILE	Multiple files icon pointer
SPTR_FOLDER	Folder icon pointer
SPTR_PROGRAM	Application program icon pointer

fCopy (BOOL) – input
Copy indicator.

TRUE — Create a copy of the default system pointer and return its handle. Specify this value if the system pointer is to be modified. The application should destroy the copy of the pointer created. This can be done by using the WinDestroyPointer function.

FALSE — Return the handle of the current system pointer.

Returns

hptrPointer (HPOINTER) – returns
Pointer handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Remarks

Take care when using the pointer bit-map handles returned by the WinQueryPointerInfo function in the POINTERINFO structure. If the handle is a system-pointer handle, or is returned by the WinQueryPointerInfo function, it is possible that another application is also accessing the bit-map handle. If this is so, selecting the bit map into a presentation space may fail. Only the active thread may use the bit-map handle returned by either the WinQuerySysPointer function, when *fCopy* is FALSE, or by the WinQueryPointerInfo function.

Note: This rule is not enforced by the system; therefore, ensure that the program handles selection failures correctly.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointerData
- WinSetPointer
- WinSetPointerPos
- WinSetSysPointerData
- WinShowPointer

Example Code

This example calls WinQuerySysPointer to get a handle to the system pointer, and then loads an application-defined pointer. After it has finished using the application-defined pointer, it restores the system pointer.

```
#define INCL_WINPOINTERS
#include <OS2.H>
#define IDP_CROSSHAIR 900

HWND hptrDefault, hptrCrossHair;

 /* get the system pointer */
 hptrDefault = WinQuerySysPointer(HWND_DESKTOP, SPTR_ARROW, FALSE);

 /* load an application-defined pointer */
 hptrCrossHair = WinLoadPointer(HWND_DESKTOP, (ULONG)0, IDP_CROSSHAIR);

 /* change the pointer to the application pointer */
 WinSetPointer(HWND_DESKTOP, hptrCrossHair);

 /* restore the system pointer */
 WinSetPointer(HWND_DESKTOP, hptrDefault);
```

WinQuerySysPointerData

This function is specific to version 2.1, or higher, of the OS/2 operating system.

This function returns the icon data for the specified system pointer for application use.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinQuerySysPointerData  (HWND hwndDesktop, ULONG iptr,
                              PICONINFO pIconInfo)
```

Parameters

hwndDesktop (HWND) – input
Handle to the desktop window.

iptr (ULONG) – input
The index of the desired system pointer.

Possible values are in the following list:

SPTR_ARROW	Arrow pointer
SPTR_TEXT	Text I-beam pointer
SPTR_WAIT	Hourglass pointer
SPTR_SIZE	Size pointer
SPTR_MOVE	Move pointer
SPTR_SIZENWSE	Downward-sloping, double-headed arrow pointer
SPTR_SIZENESW	Upward-sloping, double-headed arrow pointer
SPTR_SIZEWE	Horizontal, double-headed arrow pointer
SPTR_SIZENS	Vertical, double-headed arrow pointer
SPTR_APPICON	Standard application icon pointer
SPTR_ICONINFORMATION	Information icon pointer
SPTR_ICONQUESICON	Question mark icon pointer
SPTR_ICONERROR	Exclamation mark icon pointer
SPTR_ICONWARNING	Warning icon pointer
SPTR_ILLEGAL	Illegal operation icon pointer
SPTR_FILE	Single file icon pointer
SPTR_MULTFILE	Multiple files icon pointer
SPTR_FOLDER	Folder icon pointer
SPTR_PROGRAM	Application program icon pointer

pIconInfo (PICONINFO) – in/out
Icon data buffer.

The pointer to the buffer the icon data is to be written into. The buffer should be large enough to accommodate the icon data.

Returns

rc (BOOL) – returns
Return value.

TRUE	Successful.
FALSE	An error occurred.

Remarks

The icon data is always returned in ICON_DATA format. The buffer supplied in IconInfo must be large enough to accommodate the icon data. If the buffer size provided is 0 or not large enough, the actual buffer size needed is returned in the ICONINFO structure.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinSetPointer
- WinSetPointerPos
- WinSetSysPointerData
- WinShowPointer

WinQuerySystemAtomTable

This function returns the handle of the system atom table.

Syntax

```
#define INCL_WINATOM /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HATOMTBL WinQuerySystemAtomTable ()
```

Parameters

None.

Returns

hatomtblAtomTbl (HATOMTBL) – returns
System atom-table handle.

Remarks

The system atom table can be accessed by any process in the system. It is created at boot time and cannot be destroyed.

Related Functions

- WinAddAtom
- WinCreateAtomTable
- WinDeleteAtom
- WinDestroyAtomTable
- WinFindAtom
- WinQueryAtomLength
- WinQueryAtomName
- WinQueryAtomUsage

Example Code

This function queries the length of an atom.

```
#define INCL_WINATOM
#include <OS2.H>

HATOMTBL      hatomtbl;
ATOM          atom;
unsigned char szAtomName;

hatomtbl = WinQuerySystemAtomTable();

atom = WinFindAtom(hatomtbl, &szAtomName);
```

WinQuerySysValue

This function returns a system value.

Syntax

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinQuerySysValue  (HWND hwndDeskTop, LONG iSysValue)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	Return the system values for the desktop-window handle
Other	Return the system values for the specified desktop-window handle.

iSysValue (LONG) – input
System-value identity.

This must be one of the following SV_* constants.

Note: Not all system values can be set with the WinSetSysValue function; those that can be set are marked with an asterisk (*).

SV_CXSCREEN	Width of the screen.
SV_CYSCREEN	Height of the screen.
SV_CXVSCROLL	Width of the vertical scroll-bar.
SV_CYHSCROLL	Height of the horizontal scroll-bar.
SV_CYVSCROLLARROW	Height of the vertical scroll-bar arrow bit maps.
SV_CXHSCROLLARROW	Width of the horizontal scroll-bar arrow bit maps.
SV_CYTITLEBAR	Height of the caption.
SV_CXBORDER	Width of the nominal-width border.
SV_CYBORDER	Height of the nominal-width border.
SV_CXSIZEBORDER	(*) Width of the sizing border.
SV_CYSIZEBORDER	(*) Height of the sizing border.
SV_CXDLGFRAME	Width of the dialog-frame border.
SV_CYDLGFRAME	Height of the dialog-frame border.
SV_CYVSLIDER	Height of the vertical scroll-bar thumb.
SV_CXHSLIDER	Width of the horizontal scroll-bar thumb.

SV_CXMINMAXBUTTON	Width of the minimize/maximize buttons.
SV_CYMINMAXBUTTON	Height of the minimize/maximize buttons.
SV_CYMENU	Height of the single-line menu height.
SV_CXFULLSCREEN	Width of the client area when the window is full screen.
SV_CYFULLSCREEN	Height of the client area when the window is full screen (excluding menu height).
SV_CXICON	Icon width.
SV_CYICON	Icon height.
SV_CXPOINTER	Pointer width.
SV_CYPOINTER	Pointer height.
SV_DEBUG	FALSE indicates this is not a debug system.
SV_CMOUSEBUTTONS	The number of buttons on the pointing device (zero if no pointing device is installed).
SV_POINTERLEVEL	Pointer hide level. If the pointer level is zero, the pointer is visible. If it is greater than zero, the pointer is not visible. The WinShowPointer call is invoked to increment and decrement the SV_POINTERLEVEL, but its value cannot become negative.
SV_CTIMERS	Count of available timers.
SV_SWAPBUTTON	(*) TRUE if pointing device buttons are swapped. Normally, the pointing device buttons are set for right-handed use. Setting this value changes them for left-handed use. If TRUE, WM_LBUTTON* messages are returned when the user presses the right button, and WM_RBUTTON* messages are returned when the left button is pressed. Modifying this value affects the entire system. Applications should not normally read or set this value; users update this value by means of the user interface shell to suit their requirements.
SV_CURSORRATE	(*) Cursor blink rate, in milliseconds.
SV_DBLCLKTIME	(*) Pointing device double-click time, in milliseconds.
SV_CXDBLCLK	(*) Width of the pointing device double-click sensitive area. The default is the system-font character width.
SV_CYDBLCLK	(*) Height of the pointing device double-click sensitive area. The default is half the height of the system font character height.

SV_CXMOTIONSTART	(*) The number of pels that a pointing device must be moved in the horizontal direction, while the button is depressed, before a WM_BUTTONxMOTIONSTR message is sent.
SV_CYMOTIONSTART	(*) The number of pels that a pointing device must be moved in the vertical direction, while the button is depressed, before a WM_BUTTONxMOTIONSTR message is sent.
SV_ALARM	(*) TRUE if the alarm sound generated by WinAlarm is enabled; FALSE if the alarm sound is disabled.
SV_WARNINGFREQ	(*) Frequency for warning alarms generated by WinAlarm.
SV_WARNINGDURATION	(*) Duration for warning alarms generated by WinAlarm.
SV_NOTEFREQ	(*) Frequency for note alarms generated by WinAlarm.
SV_NOTEDURATION	(*) Duration for note alarms generated by WinAlarm.
SV_ERRORFREQ	(*) Frequency for error alarms generated by WinAlarm.
SV_ERRORDURATION	(*) Duration for error alarms generated by WinAlarm.
SV_FIRSTSCROLLRATE	(*) The delay (in milliseconds) before autoscrolling starts, when using a scroll bar.
SV_SCROLLRATE	(*) The delay (in milliseconds) between scroll operations, when using a scroll bar.
SV_CURSORLEVEL	The cursor hide level.
SV_TRACKRECTLEVEL	The hide level of the tracking rectangle (zero if visible, greater than zero if not).
SV_CXBYTEALIGN	Horizontal count of pels for alignment.
SV_CYBYTEALIGN	Vertical count of pels for alignment.
SV_SETLIGHTS	(*) When TRUE, the appropriate light is set when the keyboard state table is set.
SV_INSERTMODE	(*) TRUE if the system is in insert mode (for edit and multi-line edit controls); FALSE if in overtype mode. This system value is toggled by the system when the insert key is toggled, regardless of which window has the focus at the time.
SV_MENUROLLDOWNDELAY	(*) The delay in milliseconds before displaying a pull down referred to from a submenu item, when the button is already down as the pointer moves onto the submenu item.

SV_MENUROLLUPDELAY	(*) The delay in milliseconds before hiding a pull down referred to from a submenu item, when the button is already down as the pointer moves off the submenu item.
SV_MOUSEPRESENT	When TRUE a mouse pointing device is attached to the system.
SV_ANIMATION	(*) TRUE when animation is set on. FALSE when animation is set off.
SV_MONOICONS	(*) When TRUE preference is given to black and white icons when selecting which icon resource definition to use on the screen. Black and white icons may have more clarity than color icons on LCD and Plasma display screens.
SV_KBDALTERED	(*) Hardware ID of the newly attached keyboard. **Note:** The OS/2 National Language Support is only loaded once per system IPL. The OS/2 NLS translation is based partially on the type of keyboard device attached to the system. There are two main keyboard device types: PC AT styled and Enhanced styled. Hot Plugging between these two types of devices may result in typing anomalies due to a mismatch in the NLS device tables loaded and that of the attached device. It is strongly recommended that keyboard hot plugging be limited to the device type that the system was IPL'd with. In addition, OS/2 support will default to the 101/102 key Enhanced keyboard if no keyboard or a NetServer Mode password was in use during system IPL. (See Category 4, IOCtls 77h and 7Ah for more information on keyboard devices and types.)
SV_PRINTSCREEN	(*) TRUE when the Print Screen function is enabled; FALSE when the Print Screen function is disabled.
SV_BEGINDRAG	(*) Mouse begin drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*))
SV_ENDDRAG	(*) Mouse end drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*))
SV_BEGINSELECT	(*) Mouse begin swipe select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*))
SV_ENDSELECT	(*) Mouse select or end swipe select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*))

SV_OPEN	(*) Mouse open (low word=mouse message id (WM_*), high word=keyboard control code (KC_*))
SV_CONTEXTMENU	(*) Mouse request popup menu (low word=mouse message id (WM_*), high word=keyboard control code (KC_*))
SV_TEXTEDIT	(*) Mouse begin direct name edit (low word=mouse message id (WM_*), high word=keyboard control code (KC_*))
SV_CONTEXTMENUKB	(*) Keyboard request popup menu (low word=virtual key code (VK_*), high word=keyboard control code (KC_*))
SV_TEXTEDITKB	(*) Keyboard begin direct name edit (low word=virtual key code (VK_*), high word=keyboard control code (KC_*))
SV_ALTMNEMONIC	(*)
SV_CHORDTIME	(*)
SV_CICONTEXTLINES	(*)
SV_CXCHORD	(*)
SV_CXICONTEXTWIDTH	(*)
SV_CYCHORD	(*)
SV_EXTRAKEYBEEP	(*)
SV_NUMBEREDLISTS	(*)
SV_TASKLISTMOUSEACCESS	(*)
SV_SINGLESELECT	(*)
SV_CONTEXTHELP	(*)
SV_BEGINDRAGKB	(*)
SV_ENDDRAGKB	(*)
SV_SELECTKB	(*)
SV_OPENKB	(*)
SV_CONTEXTHELPKB	(*)
SV_BEGINSELECTKB	(*)
SV_ENDSELECTKB	(*)
SV_LOCKSTARTINPUT	(*)

Returns

lValue (LONG) – returns
System value.

0	Error occurred
Other	System value. Dimensions are in pels and times are in milliseconds.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Related Functions

- WinSetSysValue

Example Code

This example uses the WinQuerySysValue function to query the sizing border dimensions.

```
#define INCL_WINSYS
#include <OS2.H>

LONG vlXBorder, vlYBorder;

vlXBorder = WinQuerySysValue(HWND_DESKTOP,
                             SV_CXSIZEBORDER);
vlYBorder = WinQuerySysValue(HWND_DESKTOP,
                             SV_CYSIZEBORDER);
```

WinQueryTaskSizePos

This function obtains the recommended size, position and status for the first window of a newly started application (typically the main window).

Syntax

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryTaskSizePos  (HAB hab, ULONG ulID, PSWP pswp)
```

Parameters

hab (HAB) – input
Anchor-block handle.

ulID (ULONG) – input
Session.

If zero is specified, the session number of the caller is used.

pswp (PSWP) – output
Window position and size data.

Contains the recommended size and position for the first (main) window of the application. The window flags are set to indicate whether this window should be activated, minimized, or maximized.

Returns

rc (ULONG) – returns
Return code.

0	Successful completion
Other	Error occurred.

Remarks

The recommended size, position, and status for the program which is starting up, may be contained in the initialization file. However, if no data is available in the initialization file, the system generates values.

The coordinates returned are screen coordinates.

Note: For a standard window, the values returned apply to the frame window, not the client window. Where generated values are supplied, they are such as to guarantee a non null client window area within a FS_STANDARD frame window.

Related Functions

- WinAddSwitchEntry
- WinChangeSwitchEntry
- WinCreateSwitchEntry
- WinQuerySessionTitle
- WinQuerySwitchEntry
- WinQuerySwitchHandle
- WinQuerySwitchList
- WinQueryTaskTitle
- WinRemoveSwitchEntry
- WinSwitchToProgram

Example Code

This example uses the recommended size, position and status from the WinQueryTaskSize function to position the first window of a newly-started application (typically the main window).

```
#define INCL_WINSWITCHLIST
#define INCL_WINFRAMEMGR
#include <OS2.H>

HAB  hab;
SWP  winpos;
HWND hwndFrame;

WinQueryTaskSizePos(hab,
                    0,
                    &winpos);

 WinSetWindowPos(hwndFrame, HWND_TOP,
     winpos.x,                                        /* x pos */
     winpos.y,                                        /* y pos */
     winpos.cx,                                       /* x size */
     winpos.cy,                                       /* y size */
     SWP_ACTIVATE | SWP_MOVE | SWP_SIZE | SWP_SHOW); /* flags*/
```

WinQueryTaskTitle

This function obtains the title under which a specified application is started, or is added to the Window List. (See also WinQuerySessionTitle, which you should use for preference.)

Syntax

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryTaskTitle (ULONG ulSession, PSZ pszTitle, ULONG ulTitlelen)
```

Parameters

ulSession (ULONG) – input
Session identity of application whose title is requested.

0	Use the session identity of the caller
Other	Use the specified session identity.

pszTitle (PSZ) – output
Window List title.

This is the title of the application with a process identity, if the application is present in the Window List.

ulTitlelen (ULONG) – input
Maximum length of data returnable, in bytes.

If the *pszTitle* parameter is longer than this, the title is truncated. However, the terminating null character is left at the end of the string. The maximum number of title characters copied is (*ulTitlelen*–1).

Returns

rc (ULONG) – returns
Return code.

0	Successful completion
Other	Error occurred.

Remarks

This function is useful when an application uses the same name in its window title (and in its entry in the Window List) as the end user invokes to start the application. This provides a visual link for the end user.

If this function is used after a Window List entry is created for the application, the title in the Window List entry is obtained.

Related Functions

- WinAddSwitchEntry
- WinChangeSwitchEntry
- WinCreateSwitchEntry
- WinQuerySessionTitle
- WinQuerySwitchEntry
- WinQuerySwitchHandle
- WinQuerySwitchList
- WinQueryTaskSizePos
- WinRemoveSwitchEntry
- WinSwitchToProgram

Example Code

This example calls WinQueryTaskTitle to retrieve the application's title, and then sets the title bar of the frame window to that title. (The WinQuerySessionTitle could be used instead).

```
#define INCL_WINSWITCHLIST
#include <OS2.H>

HAB     hab;
HWND    hwndFrame, hwndClient;
CHAR    szTitle[MAXNAMEL + 1];
HSWITCH hswitch;
SWCNTRL swctl;

hswitch = WinQuerySwitchHandle(hwndFrame, 0);
WinQuerySwitchEntry(hswitch, &swctl);

WinQueryTaskTitle(0,
                  szTitle,
                  sizeof(szTitle));

hwndFrame = WinQueryWindow(hwndClient,
                           QW_PARENT); /* get handle of parent,  */
                                       /* which is frame window. */
WinSetWindowText(hwndFrame, szTitle);
```

WinQueryUpdateRect

This function returns the rectangle that bounds the update region of a specified window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinQueryUpdateRect  (HWND hwnd, PRECTL prclPrc)
```

Parameters

hwnd (HWND) – input
Handle of window whose update rectangle is to be queried.

prclPrc (PRECTL) – output
Update region that bounds the rectangle (in window coordinates).

Note: The value of each field in this structure must be in the range -32 768 through 32 767. The data type WRECT may also be used, if supported by the language.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or window has no update region; it is wholly valid, therefore *prclPrc* is NULL.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function is useful for implementing an incremental update scheme as an alternative to the WinBeginPaint and WinEndPaint functions.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS

- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Example Code

This example assumes that WinInvalidateRect has been called on a window owned by this window procedure. The application is sent the WM_PAINT message with the entire window as the update rectangle. The window procedure gets the dimensions of the window and calls WinValidateRect to revalidate the window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND  hwnd;
RECTL rcl;
     .
     .
     .
CASE WM_PAINT:

   WinQueryUpdateRect(hwnd, &rcl);
   WinValidateRect(hwnd,&rcl,FALSE);
   WinFillRect(hwnd,&rcl,CLR_RED);
   return 0L;
```

WinQueryUpdateRegion

This call obtains an update region of a window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinQueryUpdateRegion  (HWND hwnd, HRGN hrgn)
```

Parameters

hwnd (HWND) – input
Handle of window whose update region is to be queried.

hrgn (HRGN) – input
Handle of the window's update region.

The window's update region, in window coordinates, is copied into *hrgn*.

Returns

lComplexity (LONG) – returns
Complexity of resulting region/error indicator.

RGN_NULL	Null region
RGN_RECT	Rectangular region
RGN_COMPLEX	Complex region
RGN_ERROR	Error.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_HRGN_BUSY (0x2034)	An internal region busy error was detected. The region was locked by one thread during an attempt to access it from another thread.

Remarks

This call is useful for implementing an alternate update scheme to those used by the WinBeginPaint, and WinEndPaint functions, together with the WinValidateRegion function.

The application can use the returned update region as the clip region for a presentation space, so that drawing output can be clipped to the window's update region.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Example Code

This example gets the region that needs to be updated and then repaints the invalid region, if necessary.

```
#define INCL_WINWINDOWMGR
#define INCL_GPIREGIONS
#include <OS2.H>

HWND hwnd;
HRGN hrgn; /* region handle. */

if (RGN_NULL != WinQueryUpdateRegion(hwnd, hrgn)) {
          /* repaint the invalid region  */
               .
               .
               .

}
```

WinQueryVersion

This function is included for compatibility purposes only, and it is not recommended for use. It is recommended that DosQuerySysInfo (see the *Control Program Programming Reference*) be used to return the version, the revision level, and the environment of PM.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
ULONG WinQueryVersion  (HAB hab)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

fSysInf (ULONG) – returns
System information within which the application is operating.

WinQueryVisibleRegion

This function is similar to WinQueryUpdateRegion It returns a region which could be NULL, rectangular, or complex, and represents the visible region of the window.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryVisibleRegion  (HWND hwnd, HRGN hrgn)
```

Parameters

hwnd (HWND) – input
The window handle.

hrgn (HRGN) – input
Region handle to receive the current visible region.

Returns

ulretn (ULONG) – returns

RGN_ERROR	An error occurred.
RGN_NULL	The window currently has no visible area on screen.
RGN_RECT	The region contains a series of rectangular areas.
RGN_COMPLEX	The region contains a complex shape.

Remarks

This function returns the current visible region for the given window, even if that window does not receive notification when the visible region is changed. Currently, the region returned will always be a set of rectangular areas.

Related Functions

- WinQueryUpdateRegion
- WinSetVisibleRegionNotify

WinQueryWindow

This function returns the handle of a window that has a specified relationship to a specified window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinQueryWindow  (HWND hwnd, LONG lCode)
```

Parameters

hwnd (HWND) – input
Handle of window to query.

lCode (LONG) – input
Type of window information.

Determines what window information is returned:

QW_NEXT	Next window in z-order (window below).
QW_PREV	Previous window in z-order (window above).
QW_TOP	Topmost child window.
QW_BOTTOM	Bottommost child window.
QW_OWNER	Owner of window.
QW_PARENT	Parent of window.
QW_NEXTTOP	Returns the next window of the owner window hierarchy subject to their z-ordering. The enumeration is evaluated in this order: 1. The hierarchy of windows owned by this window in their z-order. 2. The hierarchy of windows of the next z-ordered window having the same owner as this window. 3. The hierarchy of windows in their z-order having the same owner as the owner of this window. This step is repeated until the top of the owner tree for this window is reached. 4. The hierarchy of windows in their z-order of unowned windows.
QW_PREVTOP	Returns the previous main window, in the enumeration order defined by QW_NEXTTOP.

QW_FRAMEOWNER Returns the owner of *hwnd* normalized so that if shares the same parent as *hwnd*.

Returns

hwndRelated (HWND) – returns
Window handle.

Handle of window related to *hwnd*.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Remarks

If this function is used to enumerate windows of other threads, it cannot be ensured that all the windows are enumerated, because the z-ordering of the windows can change during the enumeration. WinGetNextWindow must be used for this purpose.

If this function is called with QW_OWNER or QW_PARENT, the return value is WinQueryDesktopWindow(hab, NULLHANDLE), and not HWND_DESKTOP, when the desktop window is reached.

If this function is called with QW_PARENT for an object window, the return value is the handle of the object window associated with the desktop window as returned by the WinQueryObjectWindow function.

Related Functions

- WinBeginEnumWindows
- WinEndEnumWindows
- WinEnumDlgItem
- WinGetNextWindow
- WinIsChild
- WinMultWindowFromIDs
- WinSetOwner
- WinSetParent

Example Code

This example shows how to get the frame window handle from the client window handle.

```
#define INCL_WINWINDOWMGR
#define INCL_WINACCELERATORS
#include <OS2.H>

HACCEL haccel;
HWND hwndFrame, hwndClient; /* window handles. */
HAB hab;                    /* anchor block.   */

hwndFrame = WinQueryWindow(hwndClient,
                           QW_PARENT); /* get handle of parent, */
                                       /* which is frame window. */
```

WinQueryWindowDC

This function returns the device context for a given window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HDC WinQueryWindowDC  (HWND hwnd)
```

Parameters

hwnd (HWND) – input
Window handle.

Returns

hdc (HDC) – returns
Device-context handle.

NULLHANDLE	Either WinOpenWindowDC has not been called for this window, or an error has occurred.
Other	Device context handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

A handle is returned only if a device context has been opened for the window with WinOpenWindowDC.

Related Functions

- WinEnableWindow
- WinIsThreadActive
- WinIsWindow
- WinIsWindowEnabled
- WinQueryDesktopWindow
- WinQueryObjectWindow
- WinQueryWindowProcess
- WinQueryWindowRect
- WinWindowFromDC
- WinWindowFromID
- WinWindowFromPoint

Example Code

This example shows how to check if WinOpenWindowDC has been called for this window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND  hwndClient;           /* window handle. */

if(WinQueryWindowDC(hwndClient))
{
     /*  ...    */
}
```

WinQueryWindowModel

This function queries the memory model associated with a window.

Syntax

```
#define INCL_WINTHUNKAPI /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryWindowModel  (HWND hwnd)
```

Parameters

hwnd (HWND) – input
Window handle.

Returns

ulModel (ULONG) – returns
Memory model associated with the window.

PM_MODEL_1X	The 16-bit memory model of the 80386 processor.
PM_MODEL_2X	The 32-bit memory model of the 80386 processor.

Remarks

This function enables an application to query the memory model associate with a particular window to find out whether or not conversion of application-defined data is required. This may be necessary, for example, when sending DDE data. An existing OS/2 Version 1.1 or 1.2 application does not know about pointer conversion, so its data has to be converted for use in a 32-bit application.

The memory model is determined by how the window procedure was registered. If an application calls WinRegisterClass from 32-bit code, any windows created with that class are called 32-bit windows. If the application calls WinSubclassWindow from 16-bit code on a 32-bit window, that window becomes a 16-bit window.

Related Functions

- WinQueryClassThunkProc
- WinQueryWindowThunkProc
- WinSetClassThunkProc
- WinSetWindowThunkProc

Example Code

This example shows how to check if WinOpenWindowDC has been called for this window.

```
#define INCL_WINHOOKS
#define INCL_WINTHUNKAPI
#include <OS2.H>
HWND  hwndClient;           /* window handle. */

if(WinQueryWindowModel(hwndClient) == PM_MODEL_2X)
{
     /*  The 32-bit memory model of the 80386 processor.    */
}
```

WinQueryWindowPos

This function queries the window size and position of a visible window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinQueryWindowPos  (HWND hwnd, PSWP pswp)
```

Parameters

hwnd (HWND) – input
Window handle.

pswp (PSWP) – output
SWP structure.

The fields are set such that a call to WinSetWindowPos with those values sets the window to its current size and position, with the exception of the *fl* bits which are set as follows:

- SWP_MOVE and SWP_SIZE are set to TRUE.
- SWP_ACTIVATE and SWP_DEACTIVATE, are set to the current state of the window.
- If the window is minimized, SWP_MINIMIZE, is set and SWP_MAXIMIZE, is zero.
- If the window is maximized, SWP_MAXIMIZE, is set and SWP_MINIMIZE, is zero.
- If the window is neither minimized nor maximized, both SWP_MINIMIZE, and SWP_MAXIMIZE, are zero.
- All other bits are set to zero.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Related Functions

- WinGetMinPosition
- WinQueryActiveWindow
- WinSaveWindowPos
- WinSetActiveWindow
- WinSetMultWindowPos
- WinSetWindowPos

Example Code

This example shows how to center a dialog box within the Screen using WinQueryWindowPos.

```
#define INCL_WINWINDOWMGR
#define INCL_WINSYS
#include <OS2.H>
HWND  hwnd;           /* window handle. */
SHORT ix, iy;
SHORT iwidth, idepth;
SWP   swp;

  /* Query width and height of Screen device                          */
  iwidth = WinQuerySysValue( HWND_DESKTOP, SV_CXSCREEN );
  idepth = WinQuerySysValue( HWND_DESKTOP, SV_CYSCREEN );

  /* Query width and height of dialog box                             */
  WinQueryWindowPos( hwnd, (PSWP)&swp );

  /* Center dialog box within the Screen                              */
  ix = (SHORT)(( iwidth  - swp.cx ) / 2);
  iy = (SHORT)(( idepth  - swp.cy ) / 2);
  WinSetWindowPos( hwnd, HWND_TOP, ix, iy, 0, 0, SWP_MOVE );
```

WinQueryWindowProcess

This function obtains the process identity and thread identity of the thread that created a window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinQueryWindowProcess  (HWND hwnd, PPID ppid, PTID ptid)
```

Parameters

hwnd (HWND) – input
Window handle.

ppid (PPID) – output
Process identity of the thread that created the window.

ptid (PTID) – output
Thread identity of the thread that created the window.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinEnableWindow
- WinIsThreadActive
- WinIsWindow
- WinIsWindowEnabled
- WinQueryDesktopWindow
- WinQueryObjectWindow
- WinQueryWindowDC
- WinQueryWindowRect
- WinWindowFromDC
- WinWindowFromID
- WinWindowFromPoint

Example Code

This example shows how to query a window's process and use that information to add a switch entry window.

```
#define INCL_WINWINDOWMGR
#define INCL_WINSYS
#include <OS2.H>

HWND     hwndFrame;           /* window handle. */
SWCNTRL  swctl;
PID      pid;
TID      tid;
HSWITCH  hsw;
char     szTitle[] = "app.exe";

  WinQueryWindowProcess( hwndFrame, &pid, &tid);
  swctl.hwnd   = hwndFrame;
  swctl.idProcess = pid;
  strcpy( swctl.szSwtitle, szTitle);
  hsw = WinAddSwitchEntry( &swctl);
```

WinQueryWindowPtr

This function retrieves a pointer value from the memory of the reserved window word.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
PVOID WinQueryWindowPtr  (HWND  hwnd, LONG  index)
```

Parameters

hwnd (HWND) – input
Window handle which has the pointer to retrieve.

index (LONG) – input
Zero-based index of the pointer value to retrieve.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* –4), where *cbWindowData* is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage.

The value QWP_PFNWP can be used for the address of the window's window procedure.

Returns

pRet (PVOID) – returns
Pointer value.

NULL	Error occurred.
Other	Pointer value.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Remarks

The *index* parameter is valid only if all of the bytes referenced are within the reserved memory.

Related Functions

- WinQueryWindowULong
- WinQueryWindowUShort
- WinSetWindowBits
- WinSetWindowPtr
- WinSetWindowULong
- WinSetWindowUShort

Example Code

This function retrieves a pointer value from the memory of the reserved window word.

```
MyWindowProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
MYINSTANCEDATA *InstanceData;  /* application defined structure */

    switch (msg) {
    case WM_CREATE:
       DosAllocMem(&InstanceData, sizeof(MYINSTANCEDATA), fALLOC);
       /* WindowProcedure initializes instance data for this window */
       .
       .
       /* set pointer to instance in window words */
       WinSetWindowPtr(hwnd, 0, InstanceData);
       break;

    case WM_USER + 1: /* application defined message */
       /* Window procedure retrieves instance data to   */
       /* process this message                          */

       InstanceData = WinQueryWindowPtr(hwnd, 0);
       .
       .

       break;
       .
       .
```

WinQueryWindowRect

This function returns a window rectangle.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinQueryWindowRect (HWND hwnd, PRECTL prclDest)
```

Parameters

hwnd (HWND) – input
Window handle whose rectangle is retrieved.

prclDest (PRECTL) – output
Window rectangle.

Window rectangle of *hwnd*, in window coordinates.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT may also be used, if supported by the language.

Returns

rc (BOOL) – returns
Rectangle-returned indicator.

TRUE	Rectangle successfully returned
FALSE	Rectangle not successfully returned.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The rectangle is in window coordinates relative to itself, so that the bottom left corner is at the position (0,0).

If the size of a frame window has been changed to zero by WinSetWindowPos or WinSetMultWindowPos, the original size is returned because the window is hidden, not sized, in this instance.

Related Functions

- WinEnableWindow
- WinIsThreadActive
- WinIsWindow

- WinIsWindowEnabled
- WinQueryDesktopWindow
- WinQueryObjectWindow
- WinQueryWindowDC
- WinQueryWindowProcess
- WinWindowFromDC
- WinWindowFromID
- WinWindowFromPoint

Example Code

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.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HAB   hab;
HWND  hwnd;
RECTL rcl;

 WinQueryWindowRect(hwnd, &rcl);
 WinInvalidateRect(hwnd,        /* window to invalidate    */
                   &rcl,    /* invalid rectangle       */
                   FALSE);      /* do not include children */
```

WinQueryWindowText

This function copies window text into a buffer.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinQueryWindowText (HWND hwnd, LONG lLength, PCH pchBuffer)
```

Parameters

hwnd (HWND) – input
Window handle.

If *hwnd* is a frame-window handle, the title-bar window text is copied.

lLength (LONG) – input
Length of *pchBuffer*.

It must be greater than 0.

pchBuffer (PCH) – output
Window text.

Returns

lRetLen (LONG) – returns
Length of returned text including the null terminator.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

If the window text is longer than (*lLength*–1) only the first (*lLength*–1) characters of window text are copied.

If the window is the frame window, the title bar window text is copied.

This function sends a WM_QUERYWINDOWPARAMS message to *hwnd*.

If this function references the window of another process, *pchBuffer* must be in memory that is shared by both processes, otherwise a memory fault can occur.

Note: Extreme caution must be used if this function is used to access an object window (that is, a window that is descended from HWND_OBJECT). This is because it is

very possible that the thread that owns the object window might not be processing it's message queue, and if this is the case, a system halt could occur.

Related Functions

- WinQueryDlgItemShort
- WinQueryDlgItemText
- WinQueryDlgItemTextLength
- WinQueryWindowTextLength
- WinSetDlgItemShort
- WinSetDlgItemText
- WinSetWindowText

Related Messages

- WM_QUERYWINDOWPARAMS

Example Code

This example shows how to query window text.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
#define FID_CLIENT 255

HWND hwndFrame;
HWND hwndClient;
char szTitle[32];

 /* This function creates a new window of    */
 /* class Generic and returns hwnd.          */

hwndClient = WinCreateWindow(hwndFrame,
                             "Generic",
                            (PSZ)"My Window", /* no window text.    */
                            0UL,              /* no window style.   */
                            0,0,0,0,          /* position and size. */
                            (HWND)NULL,       /* no owner.          */
                            HWND_TOP,      /* on top of siblings */
                            FID_CLIENT,    /* client window id.  */
                            NULL,             /* control data.      */
                            NULL);            /* pres. params.      */

WinQueryWindowText(hwndFrame, sizeof(szTitle), szTitle);
```

WinQueryWindowTextLength

This call returns the length of the window text, excluding any null termination character.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinQueryWindowTextLength  (HWND hwnd)
```

Parameters

hwnd (HWND) – input
Window handle.

Returns

lRetLen (LONG) – returns
Length of the window text.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function sends a WM_QUERYWINDOWPARAMS message to *hwnd*.

Related Functions

- WinQueryDlgItemShort
- WinQueryDlgItemText
- WinQueryDlgItemTextLength
- WinQueryWindowText
- WinSetDlgItemShort
- WinSetDlgItemText
- WinSetWindowText

Related Messages

- WM_QUERYWINDOWPARAMS

Example Code

This example shows how to get the tile-bar window text.

```
#define INCL_WINWINDOWMGR
#define INCL_DOSMEMMGR
#include <OS2.H>

HWND   hwndFrame;
PSZ    szTitle;
ULONG  cbBytes;

cbBytes =  WinQueryWindowTextLength(hwndFrame);
DosAllocMem((PPVOID)szTitle,
            (ULONG)cbBytes,
            PAG_READ | PAG_WRITE | PAG_COMMIT);
WinQueryWindowText(hwndFrame, sizeof(szTitle), szTitle);
```

WinQueryWindowThunkProc

This function queries the pointer-conversion procedure associated with a window.

Syntax

```
#define INCL_WINTHUNKAPI /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
PFN WinQueryWindowThunkProc  (HWND hwnd)
```

Parameters

hwnd (HWND) – input
Window handle.

Returns

thunkpr (PFN) – returns
Pointer-conversion procedure identifier.

NULL	No pointer-conversion procedure is associated with this window.
Other	Identifier of the pointer-conversion procedure associated with this window.

Related Functions

- WinQueryClassThunkProc
- WinQueryWindowModel
- WinSetClassThunkProc
- WinSetWindowThunkProc

Example Code

This example shows how to get pointer conversion procedure associated with the frame window.

```
#define INCL_WINTHUNKAPI
#include <OS2.H>

HWND hwndFrame;
PFN  pthnkproc;

pthnkproc = WinQueryWindowThunkProc(hwndFrame);
```

WinQueryWindowULong

This function obtains the unsigned long integer value, at a specified offset, from the memory of a reserved window word, of a given window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryWindowULong  (HWND hwnd, LONG index)
```

Parameters

hwnd (HWND) – input
Handle of window to be queried.

index (LONG) – input
Zero-based index into the window words of the value to be queried.

The units of **index** are bytes. Valid values are zero through (*cbWindowData* –4), where *cbWindowData* is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage. Any of the QWL_* values, are also valid.

Note: QWS_* values cannot be used.

QWL_HMQ	Handle of message queue of window. Note that the leading 16 bits of this value are zero.
QWL_STYLE	Window style.
QWL_HWNDFOCUSSAVE	Window handle of the child windows of this window that last possessed the focus when this frame window was last deactivated.
QWL_USER	A ULONG value for applications to use is present at offset QWL_USER in windows of the following preregistered window classes: WC_FRAME (includes dialog windows) WC_COMBOBOX WC_BUTTON WC_MENU WC_STATIC WC_ENTRYFIELD WC_LISTBOX WC_SCROLLBAR WC_TITTLEBAR WC_MLE WC_SPINBUTTON

WC_CONTAINER
WC_SLIDER
WC_VALUESET
WC_NOTEBOOK

This value can be used to place application-specific data in controls.

QWL_DEFBUTTON — The default push button for a dialog.

The default push button is the one that sends its WM_COMMAND message when the enter key is pressed.

QWL_PENDATA — Reserved for use by operating system extensions. It allows an operating system extension to store data on a per window basis.

Other — Zero-based index.

Returns

ulValue (ULONG) – returns
Value contained in the window word.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001) — An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003) — The value of a parameter was not within the defined valid range for that parameter.

Remarks

The window handle that is passed to this function can be the handle of a window with the same, or different, message queue as the caller, thereby allowing the caller to obtain data from windows belonging to other threads.

The specified *index* is valid only if all of the bytes referenced are within the reserved memory.

Related Functions

- WinQueryWindowPtr
- WinQueryWindowUShort
- WinSetWindowBits
- WinSetWindowPtr
- WinSetWindowULong
- WinSetWindowUShort

Example Code

This example shows how to get the handle of the message queue of a window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwnd;
HMQ  hmq;

hmq = (HMQ)WinQueryWindowULong(hwnd, QWL_HMQ);
```

WinQueryWindowUShort

This function obtains the unsigned short integer value at a specified offset from the reserved window word's memory of a given window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinQueryWindowUShort (HWND hwnd, LONG index)
```

Parameters

hwnd (HWND) – input
Handle of window to be queried.

index (LONG) – input
Zero-based index into the window words of the value to be queried.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* –2), where *cbWindowData* is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage. Any of the QWS_* values, are valid.

Note: QWL_* values cannot be used.

QWS_ID	Window identity. The value of the *id* parameter of the WinCreateWindow function.	
QWS_FLAGS	These indicators apply only to frame or dialog windows, and contain combinations of the following indicators:	
	FF_ACTIVE	Frame window is displayed in the active state.
	FF_DIALOGBOX	Frame window is being used as a dialog box.
	FF_DLGDISMISSED	Dialog has been dismissed by the WinDismissDlg function.
	FF_FLASHHILITE	Window is currently flashed. This indicator toggles with each flash.
	FF_FLASHWINDOW	Frame window is flashing.
	FF_OWNERDISABLED	Window's owner is disabled. This indicator is only set if the window and its owner are siblings.

	FF_OWNERHIDDEN	Frame window is hidden as a result of its owner being hidden or minimized. This indicator is set only if the window and its owner are siblings.
	FF_SELECTED	Frame window is selected.
QWS_RESULT	Dialog-result parameter, as established by the WinDismissDlg function.	
QWS_XRESTORE	The x-coordinate of the position to which the window is restored. See also the QWS_CYRESTORE value.	
QWS_YRESTORE	The y-coordinate of the position to which the window is restored. See also the QWS_CYRESTORE value.	
QWS_CXRESTORE	The width to which the window is restored. See also the QWS_CYRESTORE value.	
QWS_CYRESTORE	The height to which the window is restored. These values are only valid while the window is maximized or minimized (that is, while either the WS_MINIMIZED or WS_MAXIMIZED window style indicators are set). Changing these values with the WinSetWindowUShort call alters the restore size and position.	
QWS_XMINIMIZE	The x-coordinate of the position to which the window is minimized. If this value is -1, the window has not been minimized. See also the QWS_YMINIMIZE value.	
QWS_YMINIMIZE	The y-coordinate of the position to which the window is minimized. When the window is minimized for the first time an arbitrary position is chosen. Changing these values with the WinSetWindowUShort call alters the position of the minimized window, but only when the window is not in a minimized state.	
Other	Zero-based index.	

Returns

ulValue (ULONG) – returns
Value contained in the indicated window word.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Remarks

The window handle that is passed to this function can be the handle of a window with the same, or different, message queue as the caller, thereby allowing the caller to obtain data from windows belonging to other threads.

Related Functions

- WinQueryWindowPtr
- WinQueryWindowULong
- WinSetWindowBits
- WinSetWindowPtr
- WinSetWindowULong
- WinSetWindowUShort

Related Messages

- WM_COMMAND

Example Code

In this example, the WinQueryWindowUShort call is used to query the window words to see if a window has been minimized.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND   hwnd;
USHORT usResult;

usResult = WinQueryWindowUShort(hwnd, QWS_XMINIMIZE);
if (-1 == (LONG)usResult)
{
/*    window has not been minimized. */
}
```

WinRealizePalette

This function indicates that drawing is about to take place after a palette has been selected.

Syntax

```
#define INCL_WIN /* Or use INCL_PM, */
#include <os2.h>
LONG WinRealizePalette  (HWND hwnd, HPS hps, PULONG pcclr)
```

Parameters

hwnd (HWND) – input
Window handle where drawing is taking place.

hps (HPS) – input
Presentation-space handle.

pcclr (PULONG) – output
Number of physical palette entries changed.

A value of zero indicates that the palette was successfully realized without changing any entries in the display hardware physical table. A non-zero value gives the number of hardware table entries that were changed and indicates that a WM_REALIZEPALETTE message has been posted to all other applications.

Returns

lChanged (LONG) – returns
Number of colors remapped.

PAL_ERROR	Error occurred
Otherwise	Number of colors that are remapped. This includes both animating and non-animating indexes that have matches in the physical palette. This information can be used to determine whether the window needs repainting. Note that this information may already be out of date if there are other palette-using applications running.

Possible returns from WinGetLastError

PMERR_NO_PALETTE_SELECTED (0x2110)	An attempt to realize a palette failed because no palette was previously selected into the Presentation Space.

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INV_HDC (0x207C)	An invalid device-context handle or (micro presentation space) presentation-space handle was specified.
PMERR_HDC_BUSY (0x2033)	An internal device context busy error was detected. The device context was locked by one thread during an attempt to access it from another thread.
PMERR_INV_IN_AREA (0x2085)	An attempt was made to issue a function invalid inside an area bracket. This can be detected while the actual drawing mode is **draw** or **draw-and-retain** or during segment drawing or correlation functions.

Remarks

This function is typically used after GpiSelectPalette or in response to a WM_REALIZEPALETTE message. It causes the system to ensure that the palette is appropriately realized for all drawing operations.

When the window has the input focus, the palette will be realized absolutely. Otherwise, the realization is on a best-can-do basis. If the palette is larger than the currently associated device can support, as many entries as possible are realized, starting from the lowest index.

If the presentation space is currently associated with a device context of type OD_MEMORY (see DevOpenDC), then this function performs no function other than returning without error.

This function must not be called while processing a WM_SETFOCUS message, because a window's activation state is not known until processing of this message is complete.

Note that the palette cannot be physically changed on all devices. The effect of this call is therefore device dependent.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion

- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Related Messages

- WM_SETFOCUS
- WM_REALIZEPALETTE

Example Code

In this example, the WinRealizePalette call is issued in response to a WM_REALIZEPALETTE. This ensures that the palette is appropriately realized for all drawing operations.

```
#define INCL_WIN
#include <OS2.H>
HWND hwnd;
ULONG cclr;
USHORT msg;
HPS hps;

switch(msg)
{
 case WM_REALIZEPALETTE:

 WinRealizePalette(hwnd,hps,&cclr);
}
```

WinRegisterClass

This function registers a window class.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinRegisterClass  (HAB hab, PSZ pszClassName, PFNWP pfnWndProc,
                        ULONG flStyle, ULONG cbWindowData)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pszClassName (PSZ) – input
Window-class name.

An application-specified class name.

pfnWndProc (PFNWP) – input
Window-procedure identifier.

Can be NULL if the application does not provide its own window procedure.

flStyle (ULONG) – input
Default-window style.

This can be any of the standard class styles (CS_*) (see "General Window Styles" in the *Presentation Manager Programming Reference Volume II*) in addition to any class-specific styles that are defined. These styles can be augmented when a window of this class is created.

A public window class is created if the CS_PUBLIC style is specified, otherwise a private class is created. The CS_PUBLIC style must only be specified for the shell process.

Public classes are available for creating windows from any process. Private classes are only available to the registering process.

cbWindowData (ULONG) – input
Reserved storage.

This is the number of bytes of storage reserved per window created of this class for application use.

Returns

rc (BOOL) – returns
Window-class-registration indicator.

TRUE	Window class successfully registered
FALSE	Window class not successfully registered.

Possible returns from WinGetLastError

PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.
PMERR_INVALID_INTEGER_ATOM (0x1016)	The specified atom is not a valid integer atom.
PMERR_INVALID_HATOMTBL (0x1013)	An invalid atom-table handle was specified.
PMERR_INVALID_ATOM_NAME (0x1015)	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND (0x1017)	The specified atom name is not in the atom table.

Remarks

When an application registers a private class with the window procedure in a dynamic link library, it is the application's responsibility to resolve the window-procedure address before issuing this function.

A private class must not be registered with the same name as a public class in the same process.

However, if a private class is registered with the same name as one that already exists, the parameters replace the old class parameters, and the return value is TRUE. The window procedure of an existing window can be changed using WinSubclassWindow or WinSetWindowPtr. The style of an existing window can be changed with the WinSetWindowULong or WinSetWindowUShort functions. The number of bytes of storage allocated for application use cannot be changed once the window is created.

Private classes are deleted when the process that registers them terminates.

Related Functions

- WinCalcFrameRect
- WinCreateFrameControls
- WinCreateStdWindow
- WinCreateWindow
- WinDefWindowProc
- WinDestroyWindow
- WinQueryClassInfo
- WinQueryClassName
- WinSubclassWindow

Example Code

This example calls WinRegisterClass to register a class or returns FALSE if an error occurs.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HAB hab;
CHAR szClassName[] = "Generic"; /* window class name      */
PFNWP pGenericWndProc;

 if (!WinRegisterClass(hab,   /* anchor-block handle       */
        szClassName,          /* class name                */
        pGenericWndProc,       /* window procedure          */
        0L,                   /* window style              */
        0))                   /* amount of reserved memory */
    return (FALSE);
```

WinRegisterObjectClass

The WinRegisterObjectClass function registers a workplace object class.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinRegisterObjectClass  (PSZ pClassName, PSZ pModname)
```

Parameters

pClassName (PSZ) – input
A pointer to object class being registered.

A pointer to a zero-terminated string which contains the name of the object class being registered in the workplace.

pModname (PSZ) – input
A pointer to DLL name.

A pointer to a zero-terminated string which contains the name of the DLL which holds the object definition.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The DLL must be one created using the IBM System Object Model. Object classes will automatically be added to the system when installing a DLL which contains an object definition. Generally, it is not required for the object DLL to be present at the time WinRegisterObjectClass is called. However, if the object class overrides wpclsQueryInstanceType or wpclsQueryInstanceFilter, the DLL must be present at the time of the class registration.

Related Functions

- WinCreateObject
- WinDeregisterObjectClass
- WinReplaceObjectClass

WinRegisterUserDatatype

This function registers a data type and defines its structure.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinRegisterUserDatatype  (HAB hab, LONG datatype, LONG count,
                               PLONG types)
```

Parameters

hab (HAB) – input
Anchor-block handle.

datatype (LONG) – input
Data type code to be defined.

This must not be less than DTYP_USER, and must not have been defined previously.

count (LONG) – input
Number of elements.

Must not be less than one.

types (PLONG) – input
Data type codes of structure components.

Valid data types are the system-defined data types and their pointer equivalents, application-defined data types and their pointer equivalents, and control data types. Note that not all of the data types that occur in the CPI can be specified on this function.

A control data type is followed by one or more entries in the *types* array that are interpreted in a special way. Control data types allow arrays, offsets, and lengths to be defined.

Simple Data Types:

DTYP_ATOM	See ATOM.
DTYP_BIT16	See USHORT.
DTYP_BIT32	See ULONG.
DTYP_BIT8	See UCHAR.
DTYP_BOOL	See BOOL.
DTYP_COUNT2	See USHORT.
DTYP_COUNT2B	See USHORT.
DTYP_COUNT2CH	See USHORT.
DTYP_COUNT4B	See ULONG.
DTYP_CPID	See USHORT.
DTYP_ERRORID	See ERRORID.

DTYP_IDENTITY	See USHORT.
DTYP_IDENTITY4	See ULONG.
DTYP_INDEX2	See USHORT.
DTYP_IPT	See IPT.
DTYP_LENGTH2	See USHORT.
DTYP_LENGTH4	See ULONG.
DTYP_LONG	See LONG.
DTYP_OFFSET2B	See USHORT.
DTYP_PID	See PID.
DTYP_PIX	See PIX.
DTYP_PROGCATEGORY	See PROGCATEGORY.
DTYP_PROPERTY2	See USHORT.
DTYP_PROPERTY4	See LONG.
DTYP_RESID	See HMODULE.
DTYP_SEGOFF	See SEGOFF.
DTYP_SHORT	See SHORT.
DTYP_TID	See TID.
DTYP_TIME	See LONG.
DTYP_UCHAR	See UCHAR.
DTYP_ULONG	See ULONG.
DTYP_USHORT	See USHORT.
DTYP_WIDTH4	See LONG.
DTYP_WNDPROC	See PFNWP.

Handle Data Types:

DTYP_HAB	See HAB.
DTYP_HACCEL	See HACCEL.
DTYP_HAPP	See HAPP.
DTYP_HATOMTBL	See HATOMTBL.
DTYP_HBITMAP	See HBITMAP.
DTYP_HDC	See HDC.
DTYP_HENUM	See HENUM.
DTYP_HINI	See HINI.
DTYP_HLIB	See HLIB.
DTYP_HMF	See HMF.
DTYP_HMQ	See HMQ.
DTYP_HPOINTER	See HPOINTER.
DTYP_HPROGRAM	See HPROGRAM.
DTYP_HPS	See HPS.
DTYP_HRGN	See HRGN.
DTYP_HSEM	See HSEM.
DTYP_HSPL	See HSPL.
DTYP_HSWITCH	See HSWITCH.
DTYP_HWND	See HWND.

Character/String/Buffer Data Types:

DTYP_BYTE	See BYTE.
DTYP_CHAR	See CHAR.

DTYP_STRL	See PSZ.
DTYP_STR16	See STR16.
DTYP_STR32	See STR32.
DTYP_STR64	See STR64.
DTYP_STR8	See STR8.

Structure Data Types:

DTYP_ACCEL	See ACCEL.
DTYP_ACCELTABLE	See ACCELTABLE.
DTYP_ARCPARAMS	See ARCPARAMS.
DTYP_AREABUNDLE	See AREABUNDLE.
DTYP_BITMAPINFO	See BITMAPINFO.
DTYP_BITMAPINFOHEADER	See BITMAPINFOHEADER.
DTYP_BTNCDATA	See BTNCDATA.
DTYP_CATCHBUF	See CATCHBUF.
DTYP_CHARBUNDLE	See CHARBUNDLE.
DTYP_CLASSINFO	See CLASSINFO.
DTYP_CREATESTRUCT	See CREATESTRUCT.
DTYP_CURSORINFO	See CURSORINFO.
DTYP_DEVOPENSTRUC	See DEVOPENSTRUC.
DTYP_DLGTEMPLATE	See DLGTEMPLATE.
DTYP_DLGTITEM	See DLGTITEM.
DTYP_ENTRYFDATA	See ENTRYFDATA.
DTYP_FATTRS	See FATTRS.
DTYP_FFDESCS	See FFDESCS.
DTYP_FIXED	See FIXED.
DTYP_FONTMETRICS	See FONTMETRICS.
DTYP_FRAMECDATA	See FRAMECDATA.
DTYP_GRADIENTL	See GRADIENTL.
DTYP_HCINFO	See HCINFO.
DTYP_IMAGEBUNDLE	See IMAGEBUNDLE.
DTYP_KERNINGPAIRS	See KERNINGPAIRS.
DTYP_LINEBUNDLE	See LINEBUNDLE.
DTYP_MARGSTRUCT	See MLEMARGSTRUCT.
DTYP_MARKERBUNDLE	See MARKERBUNDLE.
DTYP_MATRIXLF	See MATRIXLF.
DTYP_MLECTLDATA	See MLECTLDATA.
DTYP_OVERFLOW	See MLEOVERFLOW.
DTYP_OWNERITEM	See OWNERITEM.
DTYP_POINTERINFO	See POINTERINFO.
DTYP_POINTL	See POINTL.
DTYP_PROGRAMENTRY	See PROGRAMENTRY.
DTYP_PROGTYPE	See PROGTYPE.
DTYP_QMSG	See QMSG.
DTYP_RECTL	See RECTL.
DTYP_RGB	See RGB.
DTYP_RGNRECT	See RGNRECT.
DTYP_SBCDATA	See SBCDATA.

DTYP_SIZEF	See SIZEF.
DTYP_SIZEL	See SIZEL.
DTYP_SWBLOCK	See SWBLOCK.
DTYP_SWCNTRL	See SWCNTRL.
DTYP_SWENTRY	See SWENTRY.
DTYP_SWP	See SWP.
DTYP_TRACKINFO	See TRACKINFO.
DTYP_USERBUTTON	See USERBUTTON.
DTYP_WNDPARAMS	See WNDPARAMS.
DTYP_WPOINT	See WPOINT.
DTYP_WRECT	See WRECT.
DTYP_XYWINSIZE	See XYWINSIZE.

Pointer Data Type:

DTYP_P*

Pointer to an item of data type DTYP_*, where DTYP_* is one of the data types in the preceding lists. The value of a pointer data type is the value of the corresponding non-pointer data type prefixed with a minus to make it negative.

Minimum Application Data Type:

DTYP_USER

Minimum value for application-defined non-pointer data types.

Control Data Type:

DTYP_CTL_ARRAY

This starts a sequence of three array elements that define an array; the array resides in the structure being defined, and may have a fixed number of elements or a variable number of elements.

The following describes the possible elements:

n DTYP_CTL_ARRAY

n+1 data type of array data.

n+2 minus the number of elements in the array (for an array of fixed size), or the index of the element in *types* corresponding to the structure component which contains the number of elements in the array being defined. This component must have a suitable numeric data type. The array-size element must recede element "n" in *types*. The index is zero-based.

DTYP_CTL_LENGTH

This starts a sequence of four array elements that define a structure component containing the length of part or all of the structure. The length component resides at this point in the structure.

The following describes the possible elements:

n DYP_CTL_LENGTH

n+1 data type of structure component that contains the length (must be a suitable numeric data type).

n+2 the index of the element in *types* corresponding to the first structure component that is included in the length; a value of –1 denotes the start of the structure. This index is zero-based.

The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements.

N+3 the index of the element in *types* corresponding to the last structure component that is included in the length; it must not be less than the value contained in element n+2. A value of –1 denotes the end of the structure. The index is zero-based.

The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements.

If the value is –1, the length includes all offset data residing at the end of the structure.

DTYP_CTL_OFFSET

This starts a sequence of four array elements that define data addressed by an offset. The offset resides at this point in the structure, and contains the offset in bytes of the data from the start of the *outermost* structure in which this component resides. The data addressed by the offset must occupy storage following the fixed part of the structure. The data may be scalar data or array data.

The following describes the possible elements:

n DTYP_CTL_OFFSET

n+1 data type of the structure component that contains the offset (must be a suitable unsigned numeric data type).

n+2 data type of offset data.

n+3 minus the number of elements in the array (for an array of fixed size), or the index of the element in *types* corresponding to the structure component that contains the number of elements in the array being defined; this component must have a suitable numeric data type; the array-size element may occur earlier or later in the structure. This index is zero-based.

A value of -1 indicates that the data is not an array.

DTYP_CTL_PARRAY

This starts a sequence of three array elements that define a pointer to an array. The pointer resides at this point in the structure, and the array resides elsewhere. The array can have a fixed or variable number of elements.

The following describes the possible elements:

n DTYP_CTL_PARRAY

n+1 data type of array data.

n+2 minus the number of elements in the array (for an array of fixed size) or the index of the element in *types* corresponding to the structure component that contains the number of elements in the array being defined. This component must have a suitable numeric data type. The array-size element may occur earlier or later in the structure. The index is zero-based.

Returns

rc (BOOL) – returns
Success indicator.

TRUE Successful completion
FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_DATATYPE_TOO_SMALL (0x1048) The datatype specified was too small.

PMERR_DATATYPE_NOT_UNIQUE (0x1046) An attempt to register a datatype failed because it is not unique.

PMERR_ARRAY_TOO_SMALL (0x103E) The array specified was too small.

PMERR_DATATYPE_ENTRY_NOT_NUM (0x1043) The datatype entry specified was not numerical.

PMERR_DATATYPE_ENTRY_NOT_OFF (0x1044) The datatype entry specified was not an offset.

PMERR_DATATYPE_ENTRY_BAD_INDEX (0x103F) An invalid datatype entry index was specified.

PMERR_DATATYPE_ENTRY_CTL_MISS (0x1041) The datatype entry control was missing.

PMERR_DATATYPE_ENTRY_CTL_BAD (0x1040) An invalid datatype entry control was specified.

Remarks

This function has no effect unless the RegisterUserHook hook has been set.

The value to be used should be obtained by calling WinAddAtom with the handle of the system atom manager, and subtracting DTYP_ATOM_OFFSET from the result.

WinAddAtom is guaranteed to return values in the range 0xC000 to 0xFFFF.

When a data type is defined using this function, a definition for the corresponding pointer data type is automatically established.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example calls WinRegisterUserDatatype to register a class or returns FALSE if an error occurs.

```
#define INCL_WINMESSAGEMGR
#define INCL_WINTYPES
#include <OS2.H>
#define DTYP_MINE DTYP_USER + 1

HAB hab;
LONG  asTypes[3] = {DTYP_CHAR,
                    DTYP_STRL,
                    DTYP_STR32};

WinRegisterUserDataType(hab,
                        DTYP_MINE,
                        3,
                        asTypes);
```

WinRegisterUserMsg

This function registers a user message and defines its parameters.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinRegisterUserMsg (HAB hab, ULONG msgid, LONG datatype1,
                         LONG dir1, LONG datatype2, LONG dir2,
                         LONG datatyper)
```

Parameters

hab (HAB) – input
Anchor-block handle.

msgid (ULONG) – input
Message identifier.

This must not be less than WM_USER, and must not have been defined previously.

datatype1 (LONG) – input
Data type of message parameter 1.

Valid data types are listed below. For data types that are shorter than 4 bytes, the data must be placed in the least-significant bytes, with the most significant bytes nullified (unsigned data types) or signed-extended (signed data types).

DTYP_BIT16	See BIT16 data type.
DTYP_BIT32	See BIT32 data type.
DTYP_BIT8	See BIT8 data type.
DTYP_BOOL	See BOOL data type.
DTYP_LONG	See LONG data type.
DTYP_SHORT	See SHORT data type.
DTYP_UCHAR	See UCHAR data type.
DTYP_ULONG	See ULONG data type.
DTYP_USHORT	See USHORT data type.
DTYP_P*	A pointer to a system data type. Note that not all of the system data types that exist in the CPI are valid.
< -DTYP_USER	A pointer to a user data type. The user data type must have already been defined via WinRegisterUserDatatype.

dir1 (LONG) – input
Direction of message parameter 1.

If the message parameter is a pointer, the direction values listed below apply to the *contents* of the storage location pointed at, as well as to the message parameter itself.

RUM_IN	Input parameter (inspected by the recipient of the message, but not altered)
RUM_OUT	Output parameter (altered by the recipient of the message, without inspecting its value first)
RUM_INOUT	Input/output parameter (inspected by the recipient of the message, and then altered).

datatype2 (LONG) – input
Data type of message parameter 2.

See the description of *datatype1*.

dir2 (LONG) – input
Direction of message parameter 2.

See the description of *dir1*.

datatyper (LONG) – input
Data type of message reply.

See the description of *datatype1*. The message reply is always an output parameter.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_MSGID_TOO_SMALL (0x104F)	The message identifier specified is too small.
PMERR_DATATYPE_INVALID (0x1045)	An invalid datatype was specified.
PMERR_DATATYPE_TOO_LONG (0x1047)	The datatype specified was too long.

Remarks

This function has no effect unless the RegisterUserHook hook, which is invoked by this function, has been set.

It is an error to attempt to register the same message identifier more than once within a single OS/2 process.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example uses the WinRegisterUserMsg call to register a user-defined message and define its parameters.

```
#define INCL_WINMESSAGEMGR
#define INCL_WINTYPES
#include <OS2.H>
#define WM_MY_MESSAGE WM_USER + 11

HAB hab;

WinRegisterUserMessage(hab,
                       WM_MY_MESSAGE,
                       DTYP_BIT16,  /* param1 is a USHORT */
                       RUM_INOUT,   /* param1 is input/output */
                       DTYP_BIT16,  /* param2 is a USHORT */
                       RUM_INOUT,   /* param2 is input/output */
                       DTYP_BIT16); /* reply is a USHORT */
```

WinReleaseHook

This function releases an application hook from a hook chain.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinReleaseHook (HAB hab, HMQ hmq, LONG lHook, PFN pAddress,
                     HMODULE Module)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hmq (HMQ) – input
Handle of message queue from which the hook is to be released.

HMQ_CURRENT	The hook is released from the message queue associated with the current thread (calling thread).
NULLHANDLE	The hook is released from the system hook chain.

lHook (LONG) – input
Type of hook chain.

This parameter can have one of the following HK_* values:

HK_CHECKMSGFILTER	See CheckMsgFilterHook.
HK_CODEPAGECHANGE	See CodePageChangedHook.
HK_DESTROYWINDOW	See DestroyWindowHook.
HK_HELP	See HelpHook.
HK_INPUT	See InputHook.
HK_JOURNALPLAYBACK	See JournalPlaybackHook.
HK_JOURNALRECORD	See JournalRecordHook.
HK_LOADER	See LoaderHook.
HK_MSGCONTROL	See MsgControlHook.
HK_MSGFILTER	See MsgFilterHook.
HK_SENDMSG	See SendMsgHook.

pAddress (PFN) – input
Address of the hook routine.

Module (HMODULE) – input
Module handle.

NULLHANDLE	The hook procedure is in the application's .EXE file.
Module	This is the module that contains the application procedure, as returned by the DosLoadModule or DosQueryModuleHandle call.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HMQ (0x1002)	An invalid message-queue handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Related Functions

- WinCallMsgFilter
- WinSetHook

Example Code

This example uses the WinReleaseHook call to release a hook that records user-input messages from the application queue.

```
#define INCL_WINHOOKS
#include <OS2.H>

void RecordHook(HAB hab, PQMSG pqmsg);

samp()
{
HAB hab;

WinSetHook(hab,
           HMQ_CURRENT,
           HK_JOURNALRECORD,
           (PFN)RecordHook,
           (HMODULE)0);     /* hook is into application queue. */

WinReleaseHook(hab,
               HMQ_CURRENT,
               HK_JOURNALRECORD,
               (PFN)RecordHook,
               (HMODULE)0); /* hook is into application queue, */

}

/*  This hook records user-input messages.  */
void RecordHook(HAB hab, PQMSG pqmsg)
{
   /*  ...   */
}
```

WinReleasePS

This function releases a cache presentation space obtained using the WinGetPS, the WinGetScreenPS, or the WinGetClipPS call.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinReleasePS  (HPS hps)
```

Parameters

hps (HPS) – input
Handle of the cache presentation space to release, as returned by the WinGetPS, the WinGetScreenPS, or the WinGetClipPS function.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

Only cache presentation spaces can be released using this function, after which the presentation space is returned to the cache to be used again. The presentation-space handle should not be used following this call.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect

- WinQueryUpdateRegion
- WinRealizePalette
- WinShowWindow
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Example Code

This example shows how a thread can access a presentation space, draw to it, and release it.

```
#define INCL_DOSSEMAPHORES
#define INCL_GPIPRIMITIVES
#define INCL_WINWINDOWMGR
#include <OS2.H>
HPS hps;


      hps = WinGetPS( hwndClient );

 /*   Draw client area              */
       .                                                              */
       .                                                              */
       .

 /* Release the presentation space  */

      WinReleasePS( hps );
```

WinRemovePresParam

This function removes a presentation parameter associated with the window *hwnd*.

Syntax

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinRemovePresParam (HWND hwnd, ULONG idAttrType)
```

Parameters

hwnd (HWND) – input
Window handle.

idAttrType (ULONG) – input
Attribute type identity.

The type of the presentation parameter attribute that is to be removed. The following is the list of available presentation parameter attributes:

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

PP_DISABLEDBACKGROUNDCOLORINDEX	Disabled background color index attribute.
PP_BORDERCOLOR	Border color (in RGB) attribute.
PP_BORDERCOLORINDEX	Border color index attribute.
PP_FONTNAMESIZE	Font name and size attribute.
PP_ACTIVECOLOR	Active color value of data type RGB.
PP_ACTIVECOLORINDEX	Active color index value of data type LONG.
PP_INACTIVECOLOR	Inactive color value of data type RGB.
PP_INACTIVECOLORINDEX	Inactive color index value of data type LONG.
PP_ACTIVETEXTFGNDCOLOR	Active text foreground color value of data type RGB.
PP_ACTIVETEXTFGNDCOLORINDEX	Active text foreground color index value of data type LONG.
PP_ACTIVETEXTBGNDCOLOR	Active text background color value of data type RGB.
PP_ACTIVETEXTBGNDCOLORINDEX	Active text background color index value of data type LONG.
PP_INACTIVETEXTFGNDCOLOR	Inactive text foreground color value of data type RGB.
PP_INACTIVETEXTFGNDCOLORINDEX	Inactive text foreground color index value of data type LONG.
PP_INACTIVETEXTBGNDCOLOR	Inactive text background color value of data type RGB.
PP_INACTIVETEXTBGNDCOLORINDEX	Inactive text background color index value of data type LONG.
PP_SHADOW	Changes the color used for drop shadows on certain controls.
PP_USER	This is a user-defined presentation parameter.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

See also WinSetPresParam and WinQueryPresParam.

Related Functions

- WinDrawBitmap
- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinFillRect
- WinGetSysBitmap
- WinInvertRect
- WinQueryPresParam
- WinScrollWindow
- WinSetPresParam

Example Code

This example removes the disable-foreground attribute after querying to ensure that the referenced window has this attribute defined.

```
#define INCL_WINSYS            /* System values          */
#include <os2.h>

HWND  hwnd;                      /* window handle           */
ULONG AttrFound;                 /* attributes found        */
ULONG AttrValue[32];    /* attribute value          */
ULONG cbRetLen;                  /* length of returned value */

WinRemovePresParam(hwnd, PP_DISABLEDFOREGROUNDCOLORINDEX);
```

WinRemoveSwitchEntry

This function removes a specified entry from the Window List.

Syntax

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
ULONG WinRemoveSwitchEntry  (HSWITCH hswitchSwitch)
```

Parameters

hswitchSwitch (HSWITCH) – input
Switch-list (Window List) entry handle.

Returns

usRetCode (ULONG) – returns
Success indicator.

0	Successful completion
Other	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_SWITCH_HANDLE (0x1202)	An invalid Window List entry handle was specified.
PMERR_INVALID_WINDOW (0x1206)	The window specified with a Window List call is not a valid frame window.

Remarks

An application that uses the operating system effectively should, at least, add its main window to the Window List when it starts, and remove it from the Window List when it stops.

Window List entries for non-OS/2 applications cannot be removed using this function. These entries are removed automatically by the system when the session they occupy terminates.

Note: This function and the WinCreateSwitchEntry and WinAddSwitchEntry functions are not required if the main window is created with the frame style FCF_TASKLIST or FCF_STANDARD, as these styles automatically update the Window List when the main window is created, destroyed, or its title changes.

Related Functions

- WinAddSwitchEntry
- WinChangeSwitchEntry
- WinCreateSwitchEntry
- WinQuerySessionTitle
- WinQuerySwitchEntry
- WinQuerySwitchHandle
- WinQuerySwitchList
- WinQueryTaskSizePos
- WinQueryTaskTitle
- WinSwitchToProgram

Example Code

This example calls WinQuerySwitchHandle to get the Task List handle of a frame window, and then calls WinRemoveSwitchEntry to remove it.

```
#define INCL_WINSWITCHLIST
#include <OS2.H>
HAB hab;
HWND hwndFrame;
HSWITCH hswitch;
SWCNTRL swctl;

hswitch = WinQuerySwitchHandle(hwndFrame, 0);
WinRemoveSwitchEntry(hswitch);
```

WinReplaceObjectClass

The WinReplaceObjectClass function replaces a registered class with another registered class. If *fReplace* is FALSE, *pOldClassName* will revert back to its original definition.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinReplaceObjectClass  (PSZ pOldClassName, PSZ pNewClassName,
                             BOOL fReplace)
```

Parameters

pOldClassName (PSZ) – input
Pointer to class name.

A pointer to a zero-terminated string which contains the name of the object class being replaced by *pNewClassName* in the workplace.

pNewClassName (PSZ) – input
Pointer to new class name.

A pointer to a zero-terminated string which contains the name of the object class replacing the *pOldClassName* class.

fReplace (BOOL) – input
Function replacement flag.

TRUE Replace the function of class *pOldClassName* with the function of the class *pNewClassName*.

FALSE Undo the replacement of the *pOldClassName* with *pNewClassName* by restoring the *pOldClassName* back to its original functionality.

Returns

rc (BOOL) – returns
Success indicator.

TRUE Successful completion
FALSE Error occurred.

Remarks

The class specified by *pNewClassName* must be a descendant of the class specified by *pOldClassName*, otherwise an error will be returned. Replacing an object is useful if it is desired to modify the behavior of objects which are instances of the class *pOldClassName* and which are not aware of the class *pNewClassName*.

Related Functions

- WinCreateObject
- WinDeregisterObjectClass
- WinRegisterObjectClass

WinRequestMutexSem

WinRequestMutexSem requests ownership of a mutex semaphore or waits for a Presentation Manager message.

Syntax

```
#define INCL_WINMESSAGEMGR
#include <os2.h>
APIRET WinRequestMutexSem  (HMTX hmtx, ULONG ulTimeout)
```

Parameters

hmtx (HMTX) – input
The handle of the mutex semaphore to request.

ulTimeout (ULONG) – input
Time-out in milliseconds.

This is the maximum amount of time the user wants to allow the thread to be blocked.

This parameter can also have the following values:

Returns

ulrc (APIRET) – returns
Return Code.

WinRequestMutexSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
95	ERROR_INTERRUPT
103	ERROR_TOO_MANY_SEM_REQUESTS
105	ERROR_SEM_OWNER_DIED
640	ERROR_TIMEOUT

Remarks

WinRequestMutexSem is similar to DosRequestMutexSem and requests ownership of a mutex semaphore or waits for a window message sent by the WinSendMsg function from another thread to be received.

This function can be called by any thread in the process that created the semaphore. Threads in other processes can also call this function, but they must first gain access to the semaphore by issuing DosOpenMutexSem.

Since the processing of a window message may take longer than the value specified by the *ulTimeout* parameter, this function may not return within the time specified by that value.

Related Functions

- WinSendMsg
- WinPostMsg

Example Code

This example requests ownership of a mutex semaphore. Assume that the handle of the semaphore has been placed into *hmtx* already.

ulTimeout is the number of milliseconds that the calling thread will wait for ownership of the mutex semaphore. If the specified mutex semaphore is not released during this time interval, the calling thread does not receive ownership of it.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#define INCL_WINMESSAGEMGR

#include <os2.h>
#include <stdio.h>

#ifndef ERROR_TIMEOUT
    #define  ERROR_TIMEOUT    640
    #define  ERROR_INTERRUPT   95
#endif

HMTX    hmtx;         /* Mutex semaphore handle */
ULONG   ulTimeout;    /* Number of milliseconds to wait */
ULONG   rc;           /* Return code */

ulTimeout = 60000;  /* Wait for a maximum of 1 minute */

rc = WinRequestMutexSem(hmtx, ulTimeout);

if (rc == ERROR_TIMEOUT)
  {
    printf("WinRequestMutexSem call timed out");
    return;
  }

if (rc == ERROR_INTERRUPT)
  {
    printf("WinRequestMutexSem call was interrupted");
    return;
  }
```

```
if (rc != 0)
  {
    printf("WinRequestMutexSem error: return code = %ld", rc);
    return;
  }
```

WinRestartSOMDD

This function starts the DSOM daemon.

Syntax

```
#define INCL_WPCLASS
#include <os2.h>
APIRET WinRestartSOMDD  (BOOL fAction)
```

Parameters

fAction (BOOL) – input
Flag indicating action to perform on the DSOM daemon.

TRUE Start the DSOM daemon.
FALSE Stop the DSOM daemon.

Returns

rc (APIRET) – returns
Return code.

PMERR_OK (0x0000)	Successfully invoked.
PMERR_WPDSERVER_IS_ACTIVE (0x1056)	The Workplace Shell DSOM Server is active and running. It cannot stop the DSOM daemon until the Workplace Shell DSOM Server has stopped.
PMERR_SOMDD_IS_ACTIVE (0x1058)	The DSOM daemon has been started by the Workplace Shell process and is active and running; therefore, it cannot be started again.
PMERR_SOMDD_NOT_STARTED (0x1059)	The DSOM daemon failed to start. It might be active in another process.

Remarks

This function starts the DSOM daemon in the Workplace Shell process as a background process invisible to the users. The status of the daemon can be determined at any time by calling WinIsSOMDDReady.

Note: This function requires that the PM Shell is up and running.

Related Functions

- WinIsSOMDDReady
- WinIsWPDServerReady
- WinRestartWPDServer

Example Code

This example starts the DSOM daemon within the Workplace Shell process.

```
#define  INCL_WPCLASS
#include <os2.h>

enum   {ON, OFF};
APIRET apiRtnCd;

apiRtnCd = WinRestartSOMDD(ON);
if (apiRtnCd == PMERR_SOMDD_NOT_STARTED)
   somPrintf ("DSOM Daemon failed to start; might be active already");
```

WinRestartWPDServer

This function starts the Workplace Shell DSOM Server.

Syntax

```
#define INCL_WPCLASS
#include <os2.h>
APIRET WinRestartWPDServer (BOOL fAction)
```

Parameters

fAction (BOOL) – input
Flag indicating the action to perform on the Workplace Shell DSOM Server.

TRUE	Start Workplace Shell DSOM Server.
FALSE	Stop Workplace Shell DSOM Server.

Returns

rc (APIRET) – returns
Return code.

PMERR_OK (0x0000)	Successfully invoked.
PMERR_WPDSERVER_IS_ACTIVE (0x1056)	The Workplace Shell DSOM Server is active and cannot be started again.
PMERR_WPDSERVER_NOT_STARTED (0x1057)	The Workplace Shell DSOM Server could not be started, possibly related to a corrupted copy of WPDSRVP.DLL.
Other	Any other non-zero value could indicate that the WPDSRVP.DLL is corrupted or not found.

Remarks

This function requires that the DSOM daemon is started first and must be running successfully in order for the Workplace Shell DSOM Server to become ready. The status of the Workplace Shell DSOM Server can be obtained at any time using WinIsWPDServerReady. Once the DSOM Server is ready, a client Workplace Shell DSOM Server application can be executed.

Note: This function requires that the PM Shell is up and running.

Related Functions

- WinIsSOMDDReady
- WinIsWPDServerReady
- WinRestartSOMDD

Example Code

This example starts the DSOM daemon within the Workplace Shell process and then starts the Workplace Shell DSOM Server.

```
#define  INCL_WPCLASS
#include <os2.h>

enum    {ON, OFF};
APIRET  apiRtnCd;

apiRtnCd=WinRestartSOMDD(ON);
if (apiRtnCd == PMERR_OK)
{
   apiRtnCd = WinRestartWPDServer (ON );
   if ( apiRtnCd )
   {
      PERRORINFO perriErrorInfo;

      if (apiRtnCd == PMERR_WPDSERVER_IS_ACTIVE)
      {
         somPrint ("Workplace Shell DSOM Server is running already");
         exit(2);
      }
      if ( apiRtnCd == PMERR_WPDSERVER_NOT_STARTED )
      {
         somPrint ("Failed to start Workplace Shell DSOM Server");
         exit(2);
      }

      /* Obtain error block.   */
      /* WinGetErrorInfo is for DosLoadModule related errors only. */
      perriErrorInfo = WinGetErrorInfo( hab );
      somPrint ("Loading WPDSRVP.DLL failed: %ld", perriErrorInfo-idError);
      WinFreeErrorInfo(perriErrorInfo);
      exit(99);
   }
   SomPrintf ("All's well!");
}
else
{
   somPrintf("Failed starting DSOM daemon");
   exit(1);
}
```

WinRestoreWindowPos

The WinRestoreWindowPos function will restore the size and position of the window specified by *hwnd* to the state it was in when WinStoreWindowPos was last called with the same *pAppName* and *pKeyName*.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinRestoreWindowPos  (PSZ pAppName, PSZ pKeyName, HWND hwnd)
```

Parameters

pAppName (PSZ) – input
Pointer to application name.

A pointer to a zero-terminated string which contains the application name.

pKeyName (PSZ) – input
Pointer to key name.

A pointer to a zero-terminated string which contains the key name.

hwnd (HWND) – input
Window handle of the window to restore.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function will also restore presentation parameters which were saved by a previous call to WinStoreWindowPos.

Related Functions

- WinStoreWindowPos

WinSaveObject

This function is specific to version 3, or higher, of the OS/2 operating system.

This function saves an object.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinSaveObject  (HOBJECT hObject, BOOL fAsync)
```

Parameters

hObject (HOBJECT) – input
Handle of the object to be saved.

fAsync (BOOL) – input
Asynchronous flag.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The object is saved asynchronously on a different thread.

Using HOBJECT for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

Related Functions

- WinCopyObject
- WinCreateObject.
- WinDestroyObject
- WinMoveObject
- WinQueryObjectWindow

WinSaveWindowPos

This function associates an array of SWP structures with the process of repositioning a frame window.

Syntax

```
#define INCL_WINFRAMEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSaveWindowPos  (HSAVEWP hsvwp, PSWP pswp, ULONG cswp)
```

Parameters

hsvwp (HSAVEWP) – input
Identifier of the frame window repositioning process.

This handle is provided in the second parameter of the WM_ADJUSTFRAMEPOS message.

pswp (PSWP) – input
Array of SWP structures.

cswp (ULONG) – input
Count of SWP structures.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function is used only during the processing of the WM_ADJUSTFRAMEPOS message.

Related Functions

- WinGetMinPosition
- WinQueryActiveWindow
- WinQueryWindowPos
- WinSetActiveWindow
- WinSetMultWindowPos
- WinSetWindowPos

Related Messages

- WM_ADJUSTFRAMEPOS

Example Code

This example shows how the repositioning of a window is recorded in SWP structures with the WinSaveWindowPos call.

```
#define INCL_WINFRAMEMGR
#include <OS2.H>
#define COUNT 10
HSAVEWP hsvwp;
SWP aswp[COUNT];
ULONG  msg;

switch(msg){

case WM_ADJUSTFRAMEPOS:
WinSaveWindowPos(hsvwp,aswp,COUNT);
}
```

WinScrollWindow

This function scrolls the contents of a window rectangle.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
LONG WinScrollWindow (HWND hwnd, LONG lDx, LONG lDy, PRECTL prclScroll,
                      PRECTL prclClip, HRGN hrgnUpdateRgn,
                      PRECTL prclUpdate, ULONG flOptions)
```

Parameters

hwnd (HWND) – input
Window handle.

lDx (LONG) – input
Amount of horizontal scroll to the right (in device units).

lDy (LONG) – input
Amount of vertical scroll upward (in device units).

prclScroll (PRECTL) – input
Scroll rectangle.

If this is NULL, the entire window is scrolled.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT may also be used, if supported by the language.

prclClip (PRECTL) – input
Clip rectangle.

If not NULL, this defines a clip rectangle that clips the destination of the scroll.

hrgnUpdateRgn (HRGN) – input
Update region.

If not NULLHANDLE, this contains the region uncovered by the scroll when returned.

prclUpdate (PRECTL) – in/out
Update rectangle.

If not NULL, this contains the bounding rectangle of the invalid bits uncovered by the scroll when returned.

flOptions (ULONG) – input
Scroll options.

SW_SCROLLCHILDREN	Unless this is set, child windows are not scrolled. If this is set, and *prclScroll* is NULL, all the child windows are scrolled by *lDx* and *lDy* units. If *prclScroll* is not NULL, only those child windows that intersect *prclScroll* are scrolled.
SW_INVALIDATERGN	The invalid region created as a result of the scroll is added to the update regions of those windows affected. This may result in sending WM_PAINT messages to CS_SYNCPAINT windows before the call returns.

Returns

lComplexity (LONG) – returns
Complexity of resulting region/error indicator.

RGN_NULL	NULL rectangle invalid
RGN_RECT	Simple rectangle invalid
RGN_COMPLEX	Complex rectangle invalid
RGN_ERROR	Error.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.
PMERR_HRGN_BUSY (0x2034)	An internal region busy error was detected. The region was locked by one thread during an attempt to access it from another thread.

Remarks

This function scrolls the contents of a rectangle defined by *prclScroll* in the window *hwnd*, by *lDx* units horizontally and *lDy* units vertically. All coordinates must be in device units.

Clipping takes place on the final image of the scrolling. Even if the scroll rectangle lies outside the clip rectangle, these bits are scrolled, if their destination lies within the intersection of the clip rectangle and the destination rectangle.

This function returns an RGN_* value, indicating the type of invalid region created by the scroll as returned by GpiCombineRegion. RGN_ERROR is returned if *hwnd* is invalid.

Note: If *hwnd* has style WS_CLIPCHILDREN, portions of any child window within the scroll area are scrolled. In this instance, this function must be called with *flOptions* SW_SCROLLCHILDREN.

This is the only function that can be used by a thread to move bits within its own window, because of the critical section nature of window update regions.

Scrolling is fastest without SW_SCROLLCHILDREN and SW_INVALIDATERGN. When scrolling needs to be repeated quickly, do not include the SW_INVALIDATERGN flag and repaint the invalid area if the invalid region is rectangular, otherwise invalidate and update.

If the scrolling is infrequent, include the SW_INVALIDATERGN flag. This function invalidates and updates synchronous-paint windows automatically before returning.

The cursor and the track rectangle are scrolled when they intersect with the scrolled region. Any part of the window's initial update region that intersects the scrolled region is also offset.

Related Functions

- WinDrawBitmap
- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinFillRect
- WinGetSysBitmap
- WinInvertRect
- WinQueryPresParam
- WinRemovePresParam
- WinSetPresParam

Related Messages

- WM_PAINT

Example Code

This example shows a very small part of the processing that must be done for a WM_VSCROLL message, which will be sent when a vertical scroll bar has a significant event to notify its owner.

```
#define INCL_WINSCROLLBARS
#define INCL_WINWINDOWMGR
#include <OS2.H>
#define COUNT 10

HWND   hwndClient;
MPARAM mp2;
ULONG  msg;

switch(msg)
{

case WM_VSCROLL:

 switch (SHORT2FROMMP(mp2))

   case SB_LINEUP:     /*  Sent if the operator      */
                       /*  clicks on the up arrow    */
                       /*  of the scroll bar, or     */
                       /*  presses the VK_UP key. */

        WinScrollWindow(hwndClient,
                        0,
                        (LONG)20,  /* vertical scroll */
                        (PRECTL)NULL,
                        (PRECTL)NULL,
                        (HRGN)NULLHANDLE,
                        (PRECTL)NULL,
                        0);
                break;
                        .
                        .
                        .

break;
}
```

WinSendDlgItemMsg

This function sends a message to the dialog item defined by *idItem* in the dialog window specified by *hwndDlg*.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
MRESULT WinSendDlgItemMsg (HWND hwndDlg, ULONG idItem, ULONG msg,
                           MPARAM mp1, MPARAM mp2)
```

Parameters

hwndDlg (HWND) – input
Parent-window handle.

idItem (ULONG) – input
Identity of the child window.

It must be greater or equal to 0 and less or equal to 0xFFFF.

msg (ULONG) – input
Message identity.

mp1 (MPARAM) – input
Message parameter 1.

mp2 (MPARAM) – input
Message parameter 2.

Returns

mresReply (MRESULT) – returns
Message-return data.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function is equivalent to the WinSendMsg function, in which the receiving window procedure is specified by means of the item identity of the child window and parent-window handle.

It does not return until the message has been processed by the dialog item, whose return value is returned in *mresReply*.

The call is equivalent to:

```
WinSendMsg (WinWindowFromID(hwndDlg, idItem), msgid, param1, param2, reply);
```

This function is valid for any window with children; however, it is typically used for dialog items in a dialog window.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example processes an application-defined message (IDM_SHOW) and sets a check mark next to the selected item.

```
#define INCL_WIN
#define INCL_WINDIALOGS
#include <OS2.H>
#define IDM_SHOW 902

HWND hwndFrame;
ULONG  msg;
MPARAM mp1;

/* Inside client procedure. */

switch(msg)
{
 case WM_COMMAND:
 /* The user has chosen a menu item.  Process the selection */
 /* accordingly.                                            */

  switch ( SHORT1FROMMP( mp1 ) )
   {

    case IDM_SHOW:
            WinSendDlgItemMsg(hwndFrame,  (ULONG) FID_MENU,
                    (ULONG) MM_SETITEMATTR,
                   MPFROM2SHORT(IDM_SHOW, TRUE),
                   MPFROM2SHORT(MIA_CHECKED,MIA_CHECKED));
    break;

   }
 break;
}
```

WinSendMsg

This function sends a message with identity *ulMsgid* to *hwnd*, passing *mpParam1* and *mpParam2* as the parameters to the window.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
MRESULT WinSendMsg  (HWND hwnd, ULONG ulMsgid, MPARAM mpParam1,
                     MPARAM mpParam2)
```

Parameters

hwnd (HWND) – input
Window handle.

ulMsgid (ULONG) – input
Message identity.

mpParam1 (MPARAM) – input
Parameter 1.

mpParam2 (MPARAM) – input
Parameter 2.

Returns

mresReply (MRESULT) – returns
Message-return data.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_WINDOW_NOT_LOCKED (0x1007)	The window specified in WinSendMsg was not locked.

Remarks

mresReply is the value returned by the window procedure that is invoked. For standard window classes, the values of *mresReply* are documented with the message definitions.

This function does not complete until the message has been processed by the window procedure whose return value is returned in *mresReply*.

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, the operating system switches to the appropriate thread that enters the necessary window procedure recursively. The message is not placed in the queue of the destination thread.

If a message is sent from one process to another and the message contains a pointer, the receiving process may not have read/write access to the memory referenced by the pointer. If the receiving process is expected to update that memory, this must be done using shared memory. For more information about Dynamic Data Exchange (DDE) and shared memory, see "Dynamic Data Exchange" section of the *Presentation Manager Programming Guide - Advanced Topics*.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example gets the window handle of the system menu and calls WinSendMsg to send a message to disable the Close menu item.

```
#define INCL_WINMENUS
#define INCL_WINMESSAGEMGR
#define INCL_WINFRAMEMGR
#include <OS2.H>
HWND hwndDlg;
HWND hwndSysMenu;

 hwndSysMenu = WinWindowFromID(hwndDlg, FID_SYSMENU);
 WinSendMsg(hwndSysMenu, MM_SETITEMATTR,
     MPFROM2SHORT(SC_CLOSE, TRUE),
     MPFROM2SHORT(MIA_DISABLED, MIA_DISABLED));
```

WinSetAccelTable

This function sets the window-accelerator, or queue-accelerator table.

Syntax

```
#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetAccelTable  (HAB hab, HACCEL haccelAccel, HWND hwndFrame)
```

Parameters

hab (HAB) – input
Anchor-block handle.

haccelAccel (HACCEL) – input
Accelerator-table handle.

NULLHANDLE	Remove any accelerator table in effect for the window or the queue
Other	Accelerator-table handle.

hwndFrame (HWND) – input
Frame-window handle.

NULLHANDLE	Set the queue-accelerator table
Other	Set the window-accelerator table.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_HACCEL (0x101A)	An invalid accelerator-table handle was specified.

Related Functions

- WinCopyAccelTable
- WinCreateAccelTable
- WinDestroyAccelTable
- WinLoadAccelTable
- WinQueryAccelTable
- WinTranslateAccel

Example Code

This example uses the WinSetAccelTable call to remove any accelerator table in effect for the window.

```
#define INCL_WIN
#include <OS2.H>
HWND hwndFrame, hwndClient;
HAB hab;
HACCEL haccel;

hwndFrame = WinQueryWindow(hwndClient,
                           QW_PARENT); /* get handle of parent, */
                                       /* which is frame window. */
WinSetAccelTable(hab,
                 (HACCEL)0, /* remove any accelerator table in */
                            /* effect.                        */
                  hwndFrame);
```

WinSetActiveWindow

This function makes the frame window the active window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinSetActiveWindow  (HWND hwndDeskTop, HWND hwnd)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

hwnd (HWND) – input
Window handle.

hwnd is either the frame window or its child. If it is a child, the parent frame window will become the active window.

Returns

rc (BOOL) – returns
Active-window-set indicator.

TRUE	Active window is set
FALSE	Active window is not set.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function is equivalent to the WinFocusChange function. in which the *flFocusChange* parameter is set.

Related Functions

- WinGetMinPosition
- WinQueryActiveWindow
- WinQueryWindowPos
- WinSaveWindowPos
- WinSetMultWindowPos
- WinSetWindowPos

Example Code

This example uses the WinSetActiveWindow call to make the main window the active window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwnd;

WinSetActiveWindow(HWND_DESKTOP,hwnd);
```

WinSetCapture

This function captures all pointing device messages.

Syntax

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetCapture  (HWND hwndDesktop, HWND hwnd)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle, or HWND_DESKTOP.

hwnd (HWND) – input
Handle of the window that is to receive all pointing device messages.

hwnd can take the special value HWND_THREADCAPTURE to capture the pointing device to the current thread rather than to a particular window. HWND_THREADCAPTURE is unique among window handles.

If *hwnd* is NULLHANDLE, pointing device capture is released.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred. If the pointing device has already been captured by another thread or window, the call fails. This is to prevent applications from removing the capture from other windows or threads.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function assigns the pointing device capture to *hwnd*.

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

When this function (*hwndDesktop*, NULLHANDLE) is called to release the pointing device capture, a WM_MOUSEMOVE message is posted regardless of whether the pointing device

pointer has actually moved. This ensures that the window below the pointing device, at that time, is able to change features, such as the shape of the pointing device pointer.

If this function (*hwndDesktop*, HWND_THREADCAPTURE) is called, the pointing device is captured to the current thread. Pointing device QMSGs processed in this manner have NULLHANDLE window handles, and the pointing device coordinates are relative to the screen.

This function returns an unlocked window handle.

It must only be called while processing pointing device or keyboard input. A message box or dialog box must not be created while the pointing device is captured.

Related Functions

- WinQueryCapture

Related Messages

- WM_MOUSEMOVE

Example Code

This example uses the WinSetCapture call to capture the mouse until the button is released. The user has selected a specific object with mouse button 2.

```
#define INCL_WININPUT
#include <OS2.H>
HWND hwnd;
USHORT msg;
WinSetCapture(hwnd,HWND_DESKTOP);
  switch (msg) {
    case WM_BUTTON2DOWN:

    /*********************************************************************/
    /* An object has been picked. Set the mouse capture until            */
    /* a 'button up' message is detected.                                */
    /*********************************************************************/
      if (hwnd != WinQueryFocus(HWND_DESKTOP)){
          WinSetFocus(HWND_DESKTOP, hwnd);
      }
      WinSetCapture(HWND_DESKTOP, hwnd);
      break;
   }
```

WinSetClassMsgInterest

This function sets the message interest of a window class.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetClassMsgInterest (HAB hab, PSZ pszClassName,
                             ULONG ulMsgClass, LONG lControl)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pszClassName (PSZ) – input
Window-class name.

ulMsgClass (ULONG) – input
Message class to have interest level set.

msgid	A single message identity (for example, WM_SHOW).
SMIM_ALL	All messages (except for WM_QUIT if *lControl* is SMI_AUTODISPATCH or SMI_NOINTEREST).

lControl (LONG) – input
Interest identifier for the message class.

SMI_INTEREST	Interested in the message, or messages
SMI_NOINTEREST	Not interested in the message, or messages
SMI_AUTODISPATCH	Interested in the message or messages, but they are to be automatically dispatched to the window procedure.

Returns

rc (BOOL) – returns
Interest-changed indicator.

TRUE	Interest successfully changed
FALSE	Interest not successfully changed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function has no effect unless the MsgControlHook hook, which is invoked by this function, has been set. The interest for WM_QUIT cannot be set to SMI_AUTODISPATCH using SMIM_ALL, because WM_QUIT is the normal means of terminating an application. It can be set specifically, if required.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example uses the WinSetClassMsgInterest call to set the message interest of window class WC_MENU. It allows one to process the messages of this window class in the MsgControlHook procedure.

```
#define INCL_WINMESSAGEMGR
#define INCL_WINHOOKS
#define INCL_WINMENUS    /* for WC_MENU parameter definition. */
#include <OS2.H>
main()
{
   /* Hook Procedure Prototype */

BOOL MsgControlHook(HAB hab,LONG  idContext,    /* this hook can */
               HWND hwnd, PSZ pszClassname,     /* be given any */
               ULONG  ulMsgclass,               /* name.        */
               LONG  idControl, PBOOL fSuccess);
HWND hwnd;
HAB hab;
BOOL fSuccess;


/* This function passes the hook procedure address to the system. */

WinSetHook(hab,
          (HMQ)0,
          MCHK_CLASSMSGINTEREST,
          (PFN)MsgControlHook,.
          (HMODULE)0); /* hook is into application queue. */

 /*
 This function sets the message interest of a window class.
 */
WinSetClassMsgInterest(hab,
                       WC_MENU,  /* menu window class.         */
                       SMIM_ALL, /* set interest level for all */
                                 /* messages.                  */
                       SMI_AUTODISPATCH); /* interested in the */
                                 /* messages, but they are to  */
                                 /* be automatically dispatched */
                                 /* to the window procedure.    */
}
/*
 This hook allows the call which determine the flow of messages to be
 intercepted. It must be present for the WinSetClassMsgInterest
 call to have an effect.
*/

BOOL MsgControlHook(HAB hab,LONG  idContext,    /* this hook can */
               HWND hwnd, PSZ pszClassname,     /* be given any */
               ULONG  ulMsgclass,               /* name.        */
               LONG  idControl, PBOOL fSuccess)
{
/* ... */
}
```

WinSetClassThunkProc

This function associates a pointer-conversion procedure with a window class.

Syntax

```
#define INCL_WINTHUNKAPI /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetClassThunkProc (PSZ pszClassName, PFN pfnThunkProc)
```

Parameters

pszClassName (PSZ) – input
Window-class name.

pfnThunkProc (PFN) – input
Pointer-conversion procedure identifier.

NULL Any existing pointer-conversion procedure is dissociated from this class.

By default, a class has no pointer-conversion procedure associated with it.

Other The pointer-conversion procedure to be associated with this class.

Returns

rc (BOOL) – returns
Success indicator.

TRUE Successful completion
FALSE An error occurred.

Remarks

This function does not alter the pointer-conversion procedure associated with any existing window. It changes the pointer-conversion procedure that will be associated with a window created with a subsequent WinCreateWindow or WinCreateStdWindow function.

Related Functions

- WinQueryClassThunkProc
- WinQueryWindowModel
- WinQueryWindowThunkProc
- WinSetWindowThunkProc

Example Code

This example sets the pointer conversion procedure of the window class, given that we have an anchor-block handle.

```
#define INCL_WINWINDOWMGR
#define INCL_WINTHUNKAPI
#include <OS2.H>

LONG thunkpr(LONG *p);  /* prototype definition. */
main()
{
HAB hab;
PFN pfn;
char *classname;

WinQueryClassName(hab,
                  sizeof(classname),
                  classname);

WinSetClassThunkProc(classname,
                     (PFN)thunkpr);

}

LONG thunkpr(LONG *p)
{
/* 16-bit to 32-bit pointer conversion procedure. */
}
```

WinSetClipbrdData

This call puts data into the clipboard.

Syntax

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetClipbrdData (HAB hab, ULONG ulh, ULONG ulfmt,
                        ULONG flFmtInfo)
```

Parameters

hab (HAB) – input
Anchor-block handle.

ulh (ULONG) – input
General handle to the data object being set into the clipboard.

If NULLHANDLE, a WM_RENDERFMT message is sent to the clipboard-owner window to render the format when WinQueryClipbrdData is called with the specified format.

Once the data has been set into the clipboard, this handle can no longer be used by the application.

If CFI_POINTER is specified, this parameter contains a pointer to memory. The memory must have been allocated as unnamed and shareable, by DosAllocSharedMem with the OBJ_GIVEABLE attribute.

ulfmt (ULONG) – input
Clipboard format of the data object referenced by *ulh*.

The standard clipboard formats are shown in the following list. In addition to these predefined formats, any format value registered through the standard system atom manager displays this format in preference to privately-formatted data.

CF_TEXT	Text format. Each line ends with a carriage-return/line-feed combination. Tab characters separate fields within a line. A NULL character signals the end of the data.
CF_DSPTEXT	Text display format associated with private format.
CF_BITMAP	Bit map.
CF_DSPBITMAP	Bit-map display format associated with private format.
CF_METAFILE	Metafile.
CF_DSPMETAFILE	Metafile display format associated with private format.
CF_PALETTE	Palette.

flFmtInfo (ULONG) – input
Information.

Information about the type of data referenced by the *ulh* parameter.

Memory Model

One and only one of CFI_POINTER and CFI_HANDLE must be specified, unless CFI_OWNERDISPLAY is also specified.

CFI_POINTER — The *ulh* parameter is a flat pointer to the object.

When this memory model is specified, the system:

- saves the address (accessing it from the shell process), so that if the setting application terminates normally or abnormally, the data is still available.
- frees the memory from the setting process, so that the setting application may no longer use it.

CFI_POINTER must be specified if the *ulfmt* parameter is CF_TEXT or CF_DSPTEXT.

CFI_HANDLE — The *ulh* parameter is the handle to a metafile or bit map.

This must be specified if the *ulfmt* parameter is CF_BITMAP, CF_DSPBITMAP, CF_METAFILE or CF_DSPMETAFILE.

Usage Flags

CFI_OWNERFREE — Handle is not freed by WinEmptyClipbrd. The application must free the data if necessary.

CFI_OWNERDISPLAY — This flag indicates that the format is drawn by the clipboard owner in the clipboard viewer window by means of the WM_PAINTCLIPBOARD message. The *ulh* parameter should be NULL.

The values may be combined with bit-wise OR. Any number of the usage flags may be specified, but only one of the memory models.

Returns

rc (BOOL) – returns
Data-placed indicator.

Indicates whether data is placed into clipboard by this call:

TRUE — Data placed into clipboard.
FALSE — Data is not placed into clipboard, either an error occurred, or *ulh* is NULL.

Possible returns from WinGetLastError

PMERR_INVALID_FLAG (0x1019) — An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

PMERR_INVALID_INTEGER_ATOM (0x1016) — The specified atom is not a valid integer atom.

Remarks

Data of the specified format, already in the clipboard, is freed by this call.

An object passed to the clipboard becomes the property of the system, and is not deleted when the process that created it terminates.

Related Functions

- WinCloseClipbrd
- WinEmptyClipbrd
- WinEnumClipbrdFmts
- WinOpenClipbrd
- WinQueryClipbrdData
- WinQueryClipbrdFmtInfo
- WinQueryClipbrdOwner
- WinQueryClipbrdViewer
- WinSetClipbrdOwner
- WinSetClipbrdViewer

Related Messages

- WM_RENDERFMT
- WM_PAINTCLIPBOARD

Example Code

This example puts a bit map into the clipboard.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
HAB hab;                /* anchor-block handle. */
HBITMAP bmap;           /* bit-map handle.      */

WinOpenClipbrd(hab);
WinSetClipbrdData(hab,
                  (ULONG)bmap,
                  CF_BITMAP,
                  CFI_HANDLE); /* tells the system that the  */
                               /* bmap parameter is a handle */
                               /* to a bit map.              */
WinCloseClipbrd(hab);
```

WinSetClipbrdOwner

This function sets the current clipboard-owner window.

Syntax

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetClipbrdOwner (HAB hab, HWND hwnd)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hwnd (HWND) – input
Window handle of the new clipboard owner.

NULLHANDLE	Clipboard-owner window is released and no new clipboard-owner window is established.
Other	Window handle of the new clipboard owner.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The clipboard owner window receives the following clipboard-related messages at appropriate times:

WM_DESTROYCLIPBOARD
WM_HSCROLLCLIPBOARD
WM_PAINTCLIPBOARD
WM_RENDERALLFMTS
WM_RENDERFMT
WM_SIZECLIPBOARD
WM_VSCROLLCLIPBOARD.

Related Functions

- WinCloseClipbrd
- WinEmptyClipbrd
- WinEnumClipbrdFmts
- WinOpenClipbrd
- WinQueryClipbrdData
- WinQueryClipbrdFmtInfo
- WinQueryClipbrdOwner
- WinQueryClipbrdViewer
- WinSetClipbrdData
- WinSetClipbrdViewer

Related Messages

- WM_DESTROYCLIPBOARD
- WM_HSCROLLCLIPBOARD
- WM_PAINTCLIPBOARD
- WM_RENDERALLFMTS
- WM_RENDERFMT
- WM_SIZECLIPBOARD
- WM_VSCROLLCLIPBOARD

Example Code

This example places a bit map into the clipboard.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
HAB hab;                  /* anchor-block handle. */
HBITMAP bmap;             /* bit-map handle.      */
HWND hwnd;

WinOpenClipbrd(hab);
WinSetClipbrdOwner(hab,
                   hwnd); /* window handle of the clipboard */
                          /* owner.                         */
WinSetClipbrdData(hab,
                  (ULONG)bmap,
                  CF_BITMAP,
                  CFI_HANDLE); /* tells the system that the  */
                               /* bmap parameter is a handle */
                               /* to a bit map.              */
WinCloseClipbrd(hab);
```

WinSetClipbrdViewer

This function sets the current clipboard-viewer window to a specified window.

Syntax

```
#define INCL_WINCLIPBOARD /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetClipbrdViewer  (HAB hab, HWND hwndNewClipViewer)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hwndNewClipViewer (HWND) – input
Window handle of the new clipboard viewer.

NULLHANDLE	The clipboard-viewer window is released and no new clipboard-viewer window is established.
Other	Window handle of the new clipboard viewer.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Valid, new clipboard-viewer window established
FALSE	There is no new clipboard-viewer window established.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The clipboard-viewer window receives the WM_DRAWCLIPBOARD message when the contents of the clipboard change. This allows the viewer window to display an up-to-date version of the clipboard contents.

The clipboard must be open before this function is invoked.

Related Functions

- WinCloseClipbrd
- WinEmptyClipbrd
- WinEnumClipbrdFmts
- WinOpenClipbrd
- WinQueryClipbrdData
- WinQueryClipbrdFmtInfo
- WinQueryClipbrdOwner
- WinQueryClipbrdViewer
- WinSetClipbrdData
- WinSetClipbrdOwner

Related Messages

- WM_DRAWCLIPBOARD

Example Code

This example shows how a window views the clipboard contents.

```
#define INCL_WINCLIPBOARD
#include <OS2.H>
ULONG hclipbrdData;
HAB hab;                /* anchor-block handle. */
HBITMAP bmap;           /* bit-map handle.      */
HWND hwnd;

WinOpenClipbrd(hab);
WinSetClipbrdViewer(hab,
                    hwnd); /* window handle of the clipboard */
                           /* viewer.                        */
hclipbrdData = WinQueryClipbrdData(hab,
                                   CF_TEXT);
WinCloseClipbrd(hab);
```

WinSetCp

This function sets the code page for a queue.

Syntax

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetCp  (HMQ hmq, ULONG ulCodePage)
```

Parameters

hmq (HMQ) – input
Message-queue handle.

ulCodePage (ULONG) – input
Code page.

Either of the two ASCII code pages specified in CONFIG.SYS can be selected.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HMQ (0x1002)	An invalid message-queue handle was specified.
PMERR_RESOURCE_NOT_FOUND (0x100A)	The specified resource identity could not be found.

Remarks

The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861
Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865

Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft Windows.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	Code Page
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data.

Related Functions

- WinCpTranslateChar
- WinCpTranslateString
- WinQueryCp
- WinQueryCpList

Example Code

This example sets the code page for a message queue to 850 if it is not already set.

```
#define INCL_WINCOUNTRY
#include <OS2.H>
HMQ hmq;

if(WinQueryCp(hmq) != 850)
{
 WinSetCp(hmq, 850);
}
```

WinSetDesktopBkgnd

This function sets the desktop window state.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HBITMAP WinSetDesktopBkgnd  (HWND hwndDeskTop,
                             PDESKTOP pDeskTopState)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

pDeskTopState (PDESKTOP) – input
Desktop-state structure.

If the *fl* parameter has the SDT_LOADFILE flag set then *szFile* is used to load the bit map. If the SDT_NOBKGND flag is set, then the background is unaffected although the bit-map file may still be loaded and tiled, or scaled as requested.

Returns

hbm (HBITMAP) – returns
Desktop background bit-map handle loaded or set.

NULLHANDLE	Error occurred.
Other	Bit-map handle loaded, set, or both for desktop background. The bit-map handle returned may be different from that passed into call if any scaling or tiling is performed.

Possible returns from WinGetLastError

PMERR_INVALID_HPTR (0x101B)	An invalid pointer handle was specified.
PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function allows an application to present an image in the background of the desktop window. This application must be acting as the OS/2 PM shell in place of the IBM supplied shell. If the IBM supplied shell is executing it maintains control of the background of the desktop window, and WinSetDesktopBkgnd will have no effect on the desktop window

background, but will indicate a successful return code. The background of the desktop window is that portion of the desktop on which no other windows have been painted.

The system assumes ownership of the bit map which forms the desktop background. This implies that once the bit map is set to form the desktop background, it is no longer available to an application and therefore must not be associated with any application presentation space or any symbol set LCID. The system repaints the desktop background automatically to show any changes.

The most recent invocation of this function sets the state of the desktop background. Consequently, any application which sets the desktop background must be aware that changing the desktop background every time the application is activated, which implies the repainting of the whole desktop, could be distracting, if not disorienting, to the user. Therefore, such an application should determine if the correct desktop background is already showing by processing the WM_ACTIVATE message and if its *usactive* parameter is set to TRUE, determining the desktop background state by using the WinQueryDesktopBkgnd function and checking the bit-map handle of the current desktop background with the desired bit-map handle.

When setting a new desktop background, it is important to ensure that any previous desktop background bit map is destroyed, in order to prevent the system becoming cluttered with unused bit maps.

Related Functions

- WinQueryDesktopBkgnd

Example Code

This example sets the desktop background with a bit map if it is not already set.

```
#define INCL_WINDESKTOP
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND    hwndDeskTop;
HAB     hab;
DESKTOP DeskTopState;
HBITMAP hbm;
HBITMAP hbm_user;

WinQueryDesktopBkgnd(HWND_DESKTOP,
                     &DeskTopState);

if (hbm_user != DeskTopState.hbm)
{

 DeskTopState.fl = SDT_LOADFILE;
              /* the szFile is used to load the bit map because*/
              /* the fl parameter is set to SDT_LOADFILE.      */
 strcpy(DeskTopState.szFile,"fruit.bmp");
 DeskTopState.hbm = hbm_user;
 WinSetDesktopBkgnd(hwndDeskTop,
                    &DeskTopState);
}
```

WinSetDlgItemShort

This function converts an integer value into the text of a dialog item.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinSetDlgItemShort (HWND hwndDlg, ULONG idItem, USHORT usValue,
                         BOOL fSigned)
```

Parameters

hwndDlg (HWND) – input
Parent-window handle.

idItem (ULONG) – input
Identity of the child window whose text is to be changed.

It must be greater or equal to 0 and less or equal to 0xFFFF.

usValue (USHORT) – input
Integer value used to generate the dialog item text.

fSigned (BOOL) – input
Sign indicator.

TRUE	Signed integer value
FALSE	Unsigned integer value.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The text produced is an ASCII string.

This function is valid for any window with children; however, it is typically used for dialog items in a dialog window.

Related Functions

- WinQueryDlgItemShort
- WinQueryDlgItemText
- WinQueryDlgItemTextLength
- WinQueryWindowText
- WinQueryWindowTextLength
- WinSetDlgItemText
- WinSetWindowText

Example Code

This example gets the text from a Dialog Box entry field as an integer value.

```
#define INCL_WINDIALOGS
#define INCL_WINBUTTONS
#include <OS2.H>
#define ID_ENTRYFLD 900
#define EM_SETTEXTLIMIT 2
HAB hab;
HWND  hwnd;
ULONG  msg;

switch(msg)
 {
  case WM_INITDLG:

/* set entry field text limit. */
   WinSendDlgItemMsg(hwnd,
         /* identifier of the entry field window, which is */
         /* a child of the the window defined by hwnd.     */
                     (ULONG)ID_ENTRYFLD,
                     (ULONG)EM_SETTEXTLIMIT,  /* Limit length */
         /* MPFROM2SHORT macro is of the form (low 2 bytes, */
         /* high 2 bytes), the the number passed is simply 2. */
                    MPFROM2SHORT(2,0),
                    (MPARAM)0);

/* set entry field to 12. */
   WinSetDlgItemShort(hwnd, ID_ENTRYFLD, (SHORT)12,TRUE);
  }
```

WinSetDlgItemText

This function sets a text string in a dialog item.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinSetDlgItemText (HWND hwndDlg, ULONG idItem, PSZ pszText)
```

Parameters

hwndDlg (HWND) – input
Parent-window handle.

idItem (ULONG) – input
Identity of the child window whose text is to be set.

It must be greater or equal to 0 and less or equal to 0xFFFF.

pszText (PSZ) – input
Source string.

This is the text string that is to be set into the dialog item.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function is valid for any window with children. However, it is typically used for dialog items in a dialog window.

This function is equivalent to:

```
WinSetWindowText (WinWindowFromID (hwndDlg, idItem), pszText);
```

Related Functions

- WinQueryDlgItemShort
- WinQueryDlgItemText
- WinQueryDlgItemTextLength
- WinQueryWindowText
- WinQueryWindowTextLength
- WinSetDlgItemShort
- WinSetWindowText

Example Code

This example sets the text "CALENDAR" in a dialog box.

```
#define INCL_WINDIALOGS
#include <OS2.H>
#define ID_DLG_CALENDAR 900
HWND    hwndDlg;

    WinSetDlgItemText(hwndDlg,
                      ID_DLG_CALENDAR,
                      "CALENDAR");
```

WinSetFileIcon

WinSetFileIcon sets the icon for the file specified by *pFileName* to the icon specified by *picon*.

Syntax

```
#define INCL_WINPOINTERS
#include <os2.h>
BOOL WinSetFileIcon  (PSZ pFileName, PICONINFO picon)
```

Parameters

pFileName (PSZ) – input
Pointer to file name.

A pointer to a zero-terminated string which contains the name of the file whose icon will be set.

picon (PICONINFO) – input
A pointer to an ICONINFO structure containing an icon specification.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The specified icon is written to the file's .ICON extended attribute.

Related Functions

- WinLoadFileIcon
- WinFreeFileIcon

WinSetFocus

This function sets the focus window.

Syntax

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinSetFocus  (HWND hwndDeskTop, HWND hwndNewFocus)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

hwndNewFocus (HWND) – input
Window handle to receive the focus.

If *hwndNewFocus* identifies a desktop window, no window on the device associated with the *hwndDeskTop* receives the focus.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function is equivalent to the WinFocusChange call in which the *flFocusChange* parameter is set to 0.

If no window has the input focus, WM_CHAR messages are posted to the queue of the active window and are not thrown away.

When this function is called a WM_MOUSEMOVE message is posted regardless of whether the pointing device pointer has actually moved. This ensures that the window below the pointing device, at that time, is able to change features, such as the shape of the pointing device pointer.

This function requires the existence of a message queue.

Related Functions

- WinEnablePhysInput
- WinFocusChange
- WinGetKeyState
- WinGetPhysKeyState
- WinQueryFocus
- WinSetKeyboardStateTable

Related Messages

- WM_CHAR
- WM_MOUSEMOVE

Example Code

This example gives the client the focus if it does not already have it.

```
#define INCL_WININPUT
#include <OS2.H>
#define SYS_MENU 900
HWND hwndFrame;

if (WinQueryFocus(HWND_DESKTOP) !=        /* returns handle of  */
                                          /* window with focus. */
    WinWindowFromID(hwndFrame,FID_CLIENT))
{
    WinSetFocus(HWND_DESKTOP,
    WinWindowFromID(hwndFrame,FID_CLIENT));  /* handle of client */

}
```

WinSetHook

This function installs an application procedure into a specified hook chain.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetHook  (HAB hab, HMQ hmq, LONG lHookType, PFN pHookProc,
                  HMODULE Module)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hmq (HMQ) – input
Queue identity.

This parameter identifies the queue to which the hook chain belongs. If *hmq* is set to NULLHANDLE, the hook is installed in the system hook chain. If *hmq* is set to HMQ_CURRENT, the hook is installed in the message queue associated with the current thread (calling thread).

lHookType (LONG) – input
Hook-chain type.

HK_CHECKMSGFILTER	See CheckMsgFilterHook
HK_CODEPAGECHANGE	See CodePageChangedHook
HK_DESTROYWINDOW	See DestroyWindowHook
HK_FINDWORD	See FindWordHook
HK_FLUSHBUF	See FlushBufHook
HK_HELP	See HelpHook
HK_INPUT	See InputHook
HK_JOURNALPLAYBACK	See JournalPlaybackHook
HK_JOURNALRECORD	See JournalRecordHook
HK_LOADER	See LoaderHook
HK_LOCKUP	See LockupHook
HK_MSGCONTROL	See MsgControlHook
HK_MSGFILTER	See MsgFilterHook
HK_MSGINPUT	See MsgInputHook
HK_REGISTERUSERMSG	See RegisterUserHook
HK_SENDMSG	See SendMsgHook
HK_WINDOWDC	See WindowDCHook

pHookProc (PFN) – input
Address of the application hook procedure.

Module (HMODULE) – input
Resource identity.

Handle of the module that contains the application hook procedure, as returned by the DosLoadModule or DosQueryModuleHandle call. This parameter can be NULLHANDLE when a queue hook is being installed by an application into its own message queue.

When hooking a system hook this parameter must be a valid module handle.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HMQ (0x1002)	An invalid message-queue handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Remarks

Queue hooks are called before system hooks.

This function installs the hook at the *head* of either the system or queue chain. The most recently installed hook is called first.

Use the WinQueryWindowULong function to obtain the queue handle associated with a window handle.

Related Functions

- WinCallMsgFilter
- WinReleaseHook

Example Code

This example uses the WinSetHook call to intercept user-input messages from the application queue.

```
#define INCL_WINHOOKS
#include <OS2.H>
void RecordHook(HAB hab, PQMSG pqmsg);  /* prototype of hook */
                                        /* procedure.        */
samp()
{
HAB hab;
WinSetHook(hab,
          HMQ_CURRENT,
          HK_JOURNALRECORD,
          (PFN)RecordHook,
          (HMODULE)0); /* hook is into application queue. */

WinReleaseHook(hab,
              HMQ_CURRENT,
              HK_JOURNALRECORD,
              (PFN)RecordHook,
              (HMODULE)0); /* hook is into application queue, */

}
/*  This hook records user-input messages.  */
void RecordHook(HAB hab, PQMSG pqmsg)
{
   /*  ...   */
}
```

WinSetKeyboardStateTable

This function gets or sets the keyboard state.

Syntax

```
#define INCL_WININPUT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetKeyboardStateTable  (HWND hwndDeskTop,
                                PBYTE abKeyStateTable, BOOL fSet)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

abKeyStateTable (PBYTE) – in/out
Key state table.

This is a 256-byte table indexed by virtual key value.

For any virtual key, the 0x80 bit is set if the key is down, and zero if it is up. The 0x01 bit is set if the key is toggled, (pressed an odd number of times), otherwise it is zero.

fSet (BOOL) – input
Set indicator.

TRUE	The keyboard state is set from *abKeyStateTable*
FALSE	The keyboard state is copied to *abKeyStateTable*.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function does not change the physical state of the keyboard, but changes the value returned by WinGetKeyState, not WinGetPhysKeyState.

To set the state of a single key, first get the entire table, modify the individual key, and then set the table from the modified value.

Related Functions

- WinEnablePhysInput
- WinFocusChange
- WinGetKeyState
- WinGetPhysKeyState
- WinQueryFocus
- WinSetFocus

Example Code

This example changes the value returned by the WinGetKeyState for the NEWLINE key.

```
#define INCL_WININPUT
#include <OS2.H>
HWND hwndDeskTop;
BYTE KeyState[256]; /* This is a 256 byte table   */
                    /* indexed by virtual key     */
                    /* value.                     */
                    /* For any virtual key, the   */
                    /* 0x80 bit is set if the key */
                    /* is down, and zero if it is */
                    /* up.  The 0x01 bit is set   */
                    /* if the key is toggled,     */
                    /* (pressed an odd number     */
                    /* of times), otherwise it is */
                    /* zero.                      */

WinSetKeyboardStateTable(HWND_DESKTOP,
  /* the address of the second element is passed so that the */
  /* key number corresponds to the array index              */
                         &KeyState[1],
                         FALSE); /* get a copy of the keyboard */
                                 /* state.                     */
KeyState[VK_CAPSLOCK] |= 0x01;   /* set the CAPSLOCK key to    */
                                 /* on state          */
WinSetKeyboardStateTable(HWND_DESKTOP,
                         &KeyState[1],
                         TRUE); /* get a copy of the keyboard */
                                /* state.                     */
```

WinSetLboxItemText

This macro sets the text of the list box indexed item to buffer.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetLboxItemText  (HWND hwndLbox, LONG index, PSZ psz)
```

Parameters

hwndLbox (HWND) – input
List box handle.

index (LONG) – input
Index of the list box item.

psz (PSZ) – input
Pointer to a null terminated string.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This macro is defined as:

```
#define WinSetLboxItemText(hwndLbox, index, psz)  \
        ((BOOL)WinSendMsg(hwndLbox,               \
                          LM_SETITEMTEXT,         \
                          MPFROMLONG(index),      \
                          MPFROMP(psz)))
```

This macro requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- LM_SETITEMTEXT

Example Code

This example uses the WinSetLboxItemText call to set the months in a calendar list box.

```
#define INCL_WINDIALOGS
#include <OS2.H>

HWND    hwndLbox;
LONG    i;
typedef char MONTH[12];
MONTH months[12] = {"January", "February", "March",
                    "April", "May", "June", "July",
                    "August", "September", "October",
                    "November", "December" };

for (i=0; i<12; i++)
{
   WinSetLboxItemText(hwndLbox,
                      i,
                      months[i]);
} /* endfor */
```

WinSetMenuItemText

This macro sets the text for Menu indexed item to buffer.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetMenuItemText  (HWND hwndMenu, USHORT usId, PSZ pText)
```

Parameters

hwndMenu (HWND) – input
Menu window handle.

usId (USHORT) – input
Identity of the menu item.

pText (PSZ) – input
Text for the menu item.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This macro expands to:

```
#define WinSetMenuItemText (hwndMenu, usId, pszText)
  ((BOOL)WinSendMsg(hwndMenu,
                    MM_SETITEMTEXT,
                    MPFROMSHORT(id),
                    MPFROMP(pszText)))
```

This function requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- MM_SETITEMTEXT

Example Code

This example sets the options text in a menu.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
#define IDM_OPTIONS 900

HWND    hwndMenu;

WinQuerySetMenuItemText(hwndMenu,
                        IDM_OPTIONS,
                        "Options");
```

WinSetMsgInterest

This function sets a window's message interest.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetMsgInterest (HWND hwnd, ULONG ulMsgClass, LONG lControl)
```

Parameters

hwnd (HWND) – input
Window handle.

ulMsgClass (ULONG) – input
Message class to have interest level set.

msgid	A single message identity (for example, WM_SHOW)
SMIM_ALL	All messages (except for WM_QUIT if *lControl* is SMI_AUTODISPATCH or SMI_NOINTEREST).

lControl (LONG) – input
Interest-identifier for the message class.

SMI_RESET	Revert to interest specified for the window class.
SMI_INTEREST	Interested in the messages.
SMI_NOINTEREST	Not interested in the messages.
SMI_AUTODISPATCH	Interested in the message or messages, but they are to be automatically dispatched to the window procedure.

Returns

rc (BOOL) – returns
Interest-changed indicator.

TRUE	Interest successfully changed
FALSE	Interest not successfully changed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function has no effect unless the MsgControlHook hook, which is invoked by this function, has been set. The interest for WM_QUIT cannot be set to SMI_AUTODISPATCH using SMIM_ALL, because WM_QUIT is the normal means of terminating an application. It can be set specifically, if required.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example uses the WinSetMsgInterest call to set the message interest of a window to only WM_SHOW messages.

```
#define INCL_WINMESSAGEMGR
#define INCL_WINHOOKS
#include <OS2.H>

HWND hwnd;
HAB  hab;
BOOL fSuccess;
   /* Hook Procedure Prototype */

BOOL MsgCtlHook(HAB hab,LONG  idContext,    /* this hook can */
                HWND hwnd, PSZ pszClassname,     /* be given any */
                ULONG  ulMsgclass,               /* name.        */
                LONG  idControl, PBOOL fSuccess);

main()
{
/* This function passes the hook procedure address to the system. */

WinSetHook(hab,
          (HMQ)0,
          MCHK_CLASSMSGINTEREST,
          (PFN)MsgCtlHook,
          (HMODULE)0); /* hook is into application queue. */

 /* This function sets the message interest of a window class. */
     WinSetMsgInterest(hab,
                       WM_SHOW,
                       SMI_AUTODISPATCH); /* interested in the */
                                /* messages, but they are to     */
                                /* be automatically dispatched   */
                                /* to the window procedure.      */
}

BOOL MsgCtlHook(HAB hab, LONG idContext,       /* this hook can */
                HWND hwnd, PSZ pszClassname,    /* be given any  */
                ULONG ulMsgclass,               /* name.         */
                LONG idControl, PBOOL fSuccess)
{
/* ... */
}
```

WinSetMsgMode

This function indicates the mode for the generation and processing of messages for the private window class of an application.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinSetMsgMode  (HAB hab, PSZ pszClassName, LONG lControl)
```

Parameters

hab (HAB) – input
Anchor block handle.

pszClassName (PSZ) – input
Window class name.

lControl (LONG) – input
Message mode identifier.

SMD_DELAYED	The generation of messages may be delayed
SMD_IMMEDIATE	The generation of messages will not be delayed.

Returns

rc (BOOL) – returns
Message delay indicator.

TRUE	Message mode successfully set
FALSE	Message mode not successfully set.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function has no effect unless the MsgControlHook hook, which is invoked by this function, has been set.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg

- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example uses the WinSetMsgMode call to set the a delayed message processing mode for private window class "Generic".

```
#define INCL_WINWINDOWMGR
#define INCL_WINMESSAGEMGR
#include <OS2.H>
HWND hwnd;
HAB hab;
PFNWP GenericWndProc;
CHAR szClassName[] = "Generic"; /* window class name      */

 if (!WinRegisterClass(hab,   /* anchor-block handle       */
        szClassName,          /* class name                */
        GenericWndProc,       /* window procedure          */
        0L,                   /* window style              */
        0));                  /* amount of reserved memory */
    return (FALSE);

 WinSetMsgMode(hab,
              "Generic",
              SMD_DELAYED);
```

WinSetMultWindowPos

This function performs the WinSetWindowPos function for *cswp* windows, using *pswp*, an array of structures whose elements correspond to the input parameters of WinSetWindowPos.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetMultWindowPos  (HAB hab, PSWP pswp, ULONG cswp)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pswp (PSWP) – input
An array of set window position (SWP) structures.

The elements of each correspond to the input parameters of WinSetWindowPos.

cswp (ULONG) – input
Window count.

Returns

rc (BOOL) – returns
Positioning success indicator.

TRUE	Positioning succeeded
FALSE	Positioning failed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

All windows being positioned must have the same parent.

It is more efficient to use this function than to issue multiple WinSetWindowPos functions, as it causes less screen updating. If *hwnd* specifies a frame window, this function recalculates

the sizes and positions of the frame controls. If the new window rectangle for any frame control is to be empty, instead of resizing or repositioning that control, it is hidden by (SWP_HIDE) instead. This eliminates needless processing of windows that are not visible. The window rectangle of the control in question is left in its original state. For example, if WinSetWindowPos is issued to change the size of a standard frame window to an empty rectangle, and WinQueryWindowRect is issued against the client window, the rectangle returned is not an empty rectangle, but the original client rectangle before WinSetWindowPos was issued.

Related Functions

- WinGetMinPosition
- WinQueryActiveWindow
- WinQueryWindowPos
- WinSaveWindowPos
- WinSetActiveWindow
- WinSetWindowPos

Related Messages

- WM_ACTIVATE
- WM_ADJUSTWINDOWPOS
- WM_CALCVALIDRECTS
- WM_MOVE
- WM_SHOW
- WM_SIZE

Example Code

This example uses the WinSetMultWindowPos to cascade up to 16 main windows.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND ahwnd[16];  /* array of window handles. */
SWP aSwp[16];    /* array of SWP structures. */
HAB hab;
SWP swp;
LONG  xcoord,ycoord;
LONG  i=1;

 /* get recommended window position */

WinQueryTaskSizePos(hab,
                    0,
                    &swp);

xcoord = swp.x; ycoord = swp.y;


 /* initialize array of SWP structures where each is displaced */
 /* by (10,10).                                                */

for (i=0;i<16 ;i++ ) {
     aSwp[i]=swp;
     aSwp[i].x=xcoord;
     aSwp[i].y=ycoord;
     xcoord += 10;
     ycoord += 10;
} /* endfor */


 /* get a list of all the main windows into the ahwnd array. */

      i=0
henum=WinBeginEnumWindows
      (HWND_DESKTOP);
do
{
 ahwnd[i] = WinGetNextWindow
                    (henum);
}
while((ahwnd[i++]!=NULL) && i < 16);

WinEndEnumWindows(henum);

WinSetMultWindowPos(hab,aSwp,i-1);
```

WinSetObjectData

The WinSetObjectData function is called to set data on a workplace object.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinSetObjectData (HOBJECT object, PSZ pSetupString)
```

Parameters

object (HOBJECT) – input
Handle to a workplace object.

pSetupString (PSZ) – input
Pointer to object-specific parameters.

A pointer to a zero-terminated string which contains the object-specific parameters to the new object.

The *pSetupString* string is extracted when the wpSetup method is called. For a listing of setup strings, see the individual object classes.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

Remarks

The WinSetObjectData function will change settings on an object that was created with the WinCreateObject function.

Using HOBJECT for .INI files or files in which an application uses a rename/save/delete sequence is not supported.

Related Functions

- WinCreateObject
- WinDestroyObject

WinSetOwner

This function changes the owner window of a specified window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetOwner  (HWND hwnd, HWND hwndNewOwner)
```

Parameters

hwnd (HWND) – input
Window handle whose owner window is to be changed.

hwndNewOwner (HWND) – input
Handle of the new owner.

NULLHANDLE	The window becomes "disowned"
Other	Handle of the new owner window.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The old owner window is not locked by this function.

The WinQueryWindow function can be used to get the handle of the owner window.

Related Functions

- WinBeginEnumWindows
- WinEndEnumWindows
- WinEnumDlgItem
- WinGetNextWindow
- WinIsChild
- WinMultWindowFromIDs
- WinQueryWindow
- WinSetParent

Example Code

This example uses the WinSetOwner call to "disown" a window.

```
#define INCL_WINWINDOWMGR

#include <OS2.H>

HWND hwnd;              /* window handles. */
WinSetOwner(hwnd,(HWND)0);
```

WinSetParent

This function sets the parent for *hwnd* to *hwndNewParent*.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetParent (HWND hwnd, HWND hwndNewParent, BOOL fRedraw)
```

Parameters

hwnd (HWND) – input
Window handle.

hwndNewParent (HWND) – input
New parent window handle.

This cannot be a descendant of *hwnd*.

If this parameter is a desktop window handle or HWND_DESKTOP, *hwnd* becomes a main window.

If this parameter is not equal to HWND_OBJECT, it must be a descendant of the same desktop window as *hwnd*.

If this parameter is HWND_OBJECT or a window handle returned by the WinQueryObjectWindow function, *hwnd* becomes an object window.

fRedraw (BOOL) – input
Redraw indicator.

TRUE If *hwnd* is visible, any necessary redrawing of both the old parent and the new parent windows is performed.

FALSE No redrawing of the old and new parent windows is performed. This avoids an extra device update when subsequent calls cause the windows to be redrawn.

Returns

rc (BOOL) – returns
Parent-changed indicator.

TRUE Parent successfully changed
FALSE Parent not successfully changed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Related Functions

- WinBeginEnumWindows
- WinEndEnumWindows
- WinEnumDlgItem
- WinGetNextWindow
- WinIsChild
- WinMultWindowFromIDs
- WinQueryWindow
- WinSetOwner

Related Messages

- WM_PAINT

Example Code

This example uses the WinSetParent call to change a window to a main window.

```
#define INCL_WINWINDOWMGR

#include <OS2.H>

HWND hwnd;            /* window handles. */
WinSetParent(hwnd,
             HWND_DESKTOP,
             TRUE); /* do any necessary redrawing */
```

WinSetPointer

This call sets the desktop-pointer handle.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetPointer (HWND hwndDeskTop, HPOINTER hptrNewPointer)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

hptrNewPointer (HPOINTER) – input
New pointer handle.

NULL	Remove pointer from the screen.
Other	Pointer handle associated with *hwndDeskTop*. Handles for application-defined pointers are returned by the WinLoadPointer and WinCreatePointer calls.

Returns

rc (BOOL) – returns
Pointer-updated indicator.

TRUE	Pointer successfully updated
FALSE	Pointer not successfully updated.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_HPTR (0x101B)	An invalid pointer handle was specified.
PMERR_INV_CURSOR_BITMAP (0x205E)	An invalid pointer was referenced with WinSetPointer.

Remarks

This call is very efficient if *hptrNewPointer* is the same as the current pointer handle.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinQuerySysPointerData
- WinSetPointerPos
- WinSetSysPointerData
- WinShowPointer

Example Code

This example calls WinLoadPointer to load an application-defined pointer. When processing the WM_MOUSEMOVE message, the loaded pointer is displayed by calling WinSetPointer.

```
#define INCL_WININPUT
#define INCL_WINPOINTERS
#include <OS2.H>
#define IDP_CROSSHAIR 900

HPOINTER hptrCrossHair;
USHORT   msg;

switch(msg)
{
 case WM_CREATE:
     hptrCrossHair = WinLoadPointer(HWND_DESKTOP,
                       (ULONG)0,          /* load from .exe file    */
                       IDP_CROSSHAIR);  /* identifies the pointer */

 case WM_MOUSEMOVE:
     WinSetPointer(HWND_DESKTOP, hptrCrossHair);
}
```

WinSetPointerOwner

This function is specific to version 2.1, or higher, of the OS/2 operating system.

This function allows an appplication to declare that a pointer it created can now be used by ANY process in the system (not just the process that created the pointer).

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetPointerOwner (HPOINTER hptr, PID pid, BOOL fDestroy)
```

Parameters

hptr (HPOINTER) – input
Handle of pointer.

pid (PID) – input
Process identity.

fDestroy (BOOL) – input
Pointer destruction flag.

TRUE	Pointer can be destroyed by the current owner.
FALSE	Pointer cannot be destroyed by any owner.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Pointer owner successfully updated.
FALSE	Pointer owner not successfully updated.

Possible returns from WinGetLastError

PMERR_INVALID_HPTR (0x101B)	An invalid pointer handle was specified.

Remarks

Setting *pid* to 0 actually makes the pointer global (available on all processes).

Setting *pid* to 0, or setting *fDestroy* to FALSE should be done with extreme care. The resources allocated with this pointer will never be freed by the system, unless the pointer is set back to an active process, and *fDestroy* is set to TRUE.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinQuerySysPointerData
- WinSetPointer
- WinSetPointerPos
- WinSetSysPointerData
- WinShowPointer

Example Code

This example makes a pointer global, allowing its use by other processes.

```
WinSetPointerOwner(hMyPtr, 0L, FALSE);
```

On Exit of a DLL/Application using global pointers, the following sample code frees the resources allocated by the system.

```
WinSetPointerOwner(hMyPtr, CurPid, TRUE);
WinDestroyPointer(hMyPtr);
```

WinSetPointerPos

This function sets the pointer position.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetPointerPos  (HWND hwndDeskTop, LONG lx, LONG ly)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

lx (LONG) – input
x-position of pointer in screen coordinates.

ly (LONG) – input
y-position of pointer in screen coordinates.

Returns

rc (BOOL) – returns
Pointer position updated indicator.

TRUE	Pointer position successfully updated
FALSE	Pointer position not successfully updated.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer

- WinQuerySysPointerData
- WinSetPointer
- WinSetSysPointerData
- WinShowPointer

Example Code

This example calls WinSetPointer to set the pointer at 50, 50 in Screen coordinates.

```
#define INCL_WINPOINTERS
#include <OS2.H>

WinSetPointerPos(HWND_DESKTOP,
                 (LONG)50,
                 (LONG)50);
```

WinSetPresParam

This function sets a presentation parameter for a window.

Syntax

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetPresParam  (HWND hwnd, ULONG idAttrType,
                       ULONG cbAttrValueLen, PVOID pAttrValue)
```

Parameters

hwnd (HWND) – input
Window handle.

idAttrType (ULONG) – input
Attribute type identity.

This is either one of the system-defined presentation parameter attribute types or an application-defined type. The following list show the system-defined presentation parameter attributes:

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

PP_DISABLEDBACKGROUNDCOLORINDEX	Disabled background color index attribute.
PP_BORDERCOLOR	Border color (in RGB) attribute.
PP_BORDERCOLORINDEX	Border color index attribute.
PP_FONTNAMESIZE	Font name and size attribute.
PP_ACTIVECOLOR	Active color value of data type RGB.
PP_ACTIVECOLORINDEX	Active color index value of data type LONG.
PP_INACTIVECOLOR	Inactive color value of data type RGB.
PP_INACTIVECOLORINDEX	Inactive color index value of data type LONG.
PP_ACTIVETEXTFGNDCOLOR	Active text foreground color value of data type RGB.
PP_ACTIVETEXTFGNDCOLORINDEX	Active text foreground color index value of data type LONG.
PP_ACTIVETEXTBGNDCOLOR	Active text background color value of data type RGB.
PP_ACTIVETEXTBGNDCOLORINDEX	Active text background color index value of data type LONG.
PP_INACTIVETEXTFGNDCOLOR	Inactive text foreground color value of data type RGB.
PP_INACTIVETEXTFGNDCOLORINDEX	Inactive text foreground color index value of data type LONG.
PP_INACTIVETEXTBGNDCOLOR	Inactive text background color value of data type RGB.
PP_INACTIVETEXTBGNDCOLORINDEX	Inactive text background color index value of data type LONG.
PP_SHADOW	Changes the color used for drop shadows on certain controls.
PP_USER	This is a user-defined presentation parameter.

cbAttrValueLen (ULONG) – input
Byte count of the data passed in the *pAttrValue* parameter.

pAttrValue (PVOID) – input
Attribute value.

See the *ab* parameter of the PARAM data type for the values of system-defined attributes.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function associates the presentation parameter attribute identified by *idAttrType* with the window *hwnd*. If the attribute already exists for the window, its value is changed to the new value specified by *pAttrValue*. If the attribute does not exist, it is added to the window's presentation parameters, with the specified value. (See also WinQueryPresParam and WinRemovePresParam).

When a presentation parameter is set, a WM_PRESPARAMCHANGED message is sent to all windows owned by the window calling the WinSetPresParam function.

When switching from using color indexes to using RGB color values, you must call GpiCreateLogColorTable with the LCOLF_RGB parameter. Otherwise, this function will treat the RGB color passed as an index into a color table, rather than as an absolute color value.

Related Functions

- WinDrawBitmap
- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinFillRect
- WinGetSysBitmap
- WinInvertRect
- WinQueryPresParam
- WinRemovePresParam
- WinScrollWindow

Example Code

This example changes the border color to blue.

```
#define INCL_WINSYS
#define INCL_GPIBITMAPS  /* for RGB2 structure definition. */
#include <OS2.H>

HWND hwnd;
RGB2 rgb2;   /* Red, green, and blue color index. */
rgb2.bBlue = 200;
rgb2.bGreen = 10;
rgb2.bRed = 5;
rgb2.fcOptions = 0;

WinSetPresParam(hwnd,
                PP_BORDERCOLOR,
                (ULONG)sizeof(RGB2)
                (PVOID)&rgb2);
```

WinSetRect

This function sets rectangle coordinates.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetRect  (HAB hab, PRECTL prclrect, LONG lLeft, LONG lBottom,
                  LONG lRight, LONG lTop)
```

Parameters

hab (HAB) – input
Anchor-block handle.

prclrect (PRECTL) – in/out
Rectangle to be updated.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT may also be used, if supported by the language.

lLeft (LONG) – input
Left edge of rectangle.

lBottom (LONG) – input
Bottom edge of rectangle.

lRight (LONG) – input
Right edge of rectangle.

lTop (LONG) – input
Top edge of rectangle.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function is equivalent to assigning the left, top, right, and bottom arguments to the appropriate fields of RECTL.

Related Functions

- WinCopyRect
- WinEqualRect
- WinFillRect
- WinInflateRect
- WinIntersectRect
- WinIsRectEmpty
- WinOffsetRect
- WinPtInRect
- WinSetRectEmpty
- WinSubtractRect
- WinUnionRect

Example Code

This example calls WinQueryWindowRect to get the dimensions of the window, and then calls WinSetRect to downsize it.

```
#define INCL_WINRECTANGLES
#include <OS2.H>
HAB hab;
RECTL rcl;
HWND hwnd;

 WinQueryWindowRect(hwnd, &rcl);       /* get window dimensions */
 WinSetRect(hab,&rcl,
            rcl.xLeft - 10,
            rcl.yBottom -10,
            rcl.xRight - 10,
            rcl.yTop - 10);
```

WinSetRectEmpty

This function sets a rectangle empty.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetRectEmpty  (HAB hab, PRECTL prclrect)
```

Parameters

hab (HAB) – input
Anchor-block handle.

prclrect (PRECTL) – in/out
Rectangle to be set empty.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT may also be used, if supported by the language.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function is equivalent to a WinSetRect (*hab*, *prclrect*, 0, 0, 0, 0) call.

Related Functions

- WinCopyRect
- WinEqualRect
- WinFillRect
- WinInflateRect
- WinIntersectRect
- WinIsRectEmpty
- WinOffsetRect
- WinPtInRect
- WinSetRect
- WinSubtractRect
- WinUnionRect

Example Code

This example calls WinSetRectEmpty to empty the rectangle structure.

```
#define INCL_WINRECTANGLES
#include <OS2.H>
HAB hab;
RECTL rcl;

 WinSetRectEmpty(hab,&rcl);
```

WinSetSynchroMode

This function is intended for use in a distributed application.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinSetSynchroMode  (HAB hab, LONG lMode)
```

Parameters

hab (HAB) – input
Anchor-block handle.

lMode (LONG) – input
Synchronization mode.

SSM_SYNCHRONOUS	Synchronous mode
SSM_ASYNCHRONOUS	Asynchronous mode
SSM_MIXED	Mixed mode.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function allows an application whose message queue is distributed, to synchronize the processing of those messages. This is achieved by the use of the MsgControlHook hook which is invoked by this function.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg

- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinWaitMsg

Example Code

This function is intended for use in an application with a distributed queue.

```
#define INCL_WINMESSAGEMGR
#include <OS2.H>
HAB hab;
WinSetSynchroMode(hab,
                  SSM_SYNCHRONOUS);  /* synchronous mode. */
```

WinSetSysColors

This function sets system color values.

Syntax

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetSysColors (HWND hwndDeskTop, ULONG flOptions,
                      ULONG ulFormat, LONG lStart, ULONG ulTablen,
                      PLONG alTable)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

flOptions (ULONG) – input
Options.

LCOL_RESET	The system colors are all to be reset to default before processing the remainder of the data in this function.
LCOL_PURECOLOR	Color-dithering should not be used to create colors not available in the physical palette. If this option is set, only pure colors are used and no dithering is done.

ulFormat (ULONG) – input
Format of entries in the table, as follows.

LCOLF_INDRGB	Array of (index,RGB) values. Each pair of entries is 8-bytes long, comprising 4 bytes for the index, and 4 bytes for the color value. For system color indexes, see *lStart*.
LCOLF_CONSECRGB	Array of (RGB) values, corresponding to color indexes *lStart* upwards. Each entry is 4-bytes long.

lStart (LONG) – input
Starting system color index.

This parameter is applicable only if the *ulFormat* parameter is set to LCOLF_CONSECRGB.

The number of system colors (as defined below) is given by SYSCLR_CSYSCOLORS.

The following system color indexes are defined (each successive index is one larger than its predecessor):

SYSCLR_ENTRYFIELD	Entry field and list box background color.
SYSCLR_MENUDISABLEDTEXT	Entry field background color.
SYSCLR_MENUHILITE	Selected menu item text.
SYSCLR_MENUHILITEBGND	Selected menu item background.
SYSCLR_PAGEBACKGROUND	Notebook page background.
SYSCLR_FIELDBACKGROUND	Inactive scroll bar and default control background color.
SYSCLR_BUTTONLIGHT	Light push button (3D effect).
SYSCLR_BUTTONMIDDLE	Middle push button (3D effect).
SYSCLR_BUTTONDARK	Dark push button (3D effect).
SYSCLR_BUTTONDEFAULT	Push button.
SYSCLR_TITLEBOTTOM	Line drawn under title bar.
SYSCLR_SHADOW	Drop shadow for menus and dialogs.
SYSCLR_ICONTEXT	Text written under icons on the desktop.
SYSCLR_DIALOGBACKGROUND	Pop up dialog box background.
SYSCLR_HILITEFOREGROUND	Selection foreground.
SYSCLR_HILITEBACKGROUND	Selection background.
SYSCLR_INACTIVETITLETEXTBKGD	Background of inactive title text.
SYSCLR_ACTIVETITLETEXTBKGD	Background of active title text.
SYSCLR_INACTIVETITLETEXT	Inactive title text.
SYSCLR_ACTIVETITLETEXT	Active title text.
SYSCLR_OUTPUTTEXT	Output text.
SYSCLR_WINDOWSTATICTEXT	Static (nonselectable) text.
SYSCLR_SCROLLBAR	Active scroll bar background area.
SYSCLR_BACKGROUND	Desktop background.
SYSCLR_ACTIVETITLE	Active window title.
SYSCLR_INACTIVETITLE	Inactive window title.
SYSCLR_MENU	Menu background.
SYSCLR_WINDOW	Window background.
SYSCLR_WINDOWFRAME	Window frame (border line).
SYSCLR_MENUTEXT	Normal menu item text.
SYSCLR_WINDOWTEXT	Window text.
SYSCLR_TITLETEXT	Text in title bar, size box, scroll bar arrow box.
SYSCLR_ACTIVEBORDER	Border fill of active window.
SYSCLR_INACTIVEBORDER	Border fill of inactive window.
SYSCLR_APPWORKSPACE	Background of specific main windows.
SYSCLR_HELPBACKGROUND	Background of help panels.
SYSCLR_HELPTEXT	Help text.
SYSCLR_HELPHILITE	Highlighted help text.
SYSCLR_SHADOWHILITEBGND	Shadows of workplace object background highlight color.
SYSCLR_SHADOWHILITEFGND	Shadows of workplace object foreground highlight color.
SYSCLR_SHADOWTEXT	Shadows of workplace object text color.

ulTablen (ULONG) – input
Number of elements.

Number of elements supplied in *alTable*. This may be 0 if, for example, the color table is merely to be reset to the default. For LCOLF_INDRGB it must be an even number.

alTable (PLONG) – input
Table.

Start address of the application data area, containing the color-table definition data. The format depends on the value of *ulFormat*.

Each color value is a 4-byte integer, with a value of

```
(R * 65536) + (G * 256) + B
```

where:

R is red intensity value
G is green intensity value
B is blue intensity value.

There are 8 bits for each primary; the maximum intensity for each primary is 255.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Remarks

This function sends all main windows in the system a WM_SYSCOLORCHANGE message to indicate that the colors have changed. When this message is received, applications that depend on the system colors can query the new color values with the WinQuerySysColor function.

After the WM_SYSCOLORCHANGE messages are sent, all windows in the system are invalidated so that they are redrawn with the new system colors.

This function does *not* write any system color changes to the initialization file See "Format of Interchange Files" the *Presentation Manager Programming Reference Volume II*.

The following table gives the default RGB values for each color index:

System Color Index	Default Color	Default RGB Values
SYSCLR_ACTIVEBORDER	Pale yellow	255 255 128
SYSCLR_ACTIVETITLE	Teal	64 128 128
SYSCLR_ACTIVETITLETEXT	White	255 255 255
SYSCLR_ACTIVETITLETEXTBGND	Teal	64 128 128
SYSCLR_APPWORKSPACE	Off-white	255 255 224
SYSCLR_BACKGROUND	Light gray	204 204 204
SYSCLR_BUTTONDARK	Dark gray	128 128 128
SYSCLR_BUTTONDEFAULT	Black	0 0 0
SYSCLR_BUTTONLIGHT	White	255 255 255
SYSCLR_BUTTONMIDDLE	Light gray	204 204 204
SYSCLR_DIALOGBACKGROUND	Light gray	204 204 204
SYSCLR_ENTRYFIELD	Pale yellow	255 255 204
SYSCLR_FIELDBACKGROUND	Light gray	204 204 204
SYSCLR_HELPBACKGROUND	White	255 255 255
SYSCLR_HELPHILITE	Blue green	0 128 128
SYSCLR_HELPTEXT	Dark blue	0 0 128
SYSCLR_HILITEBACKGROUND	Dark gray	128 128 128
SYSCLR_HILITEFOREGROUND	White	255 255 255
SYSCLR_ICONTEXT	Black	0 0 0
SYSCLR_INACTIVEBORDER	Light gray	204 204 204
SYSCLR_INACTIVETITLE	Light gray	204 204 204
SYSCLR_INACTIVETITLETEXT	Dark gray	128 128 128
SYSCLR_INACTIVETITLETEXTBGND	Light gray	204 204 204
SYSCLR_MENU	Light gray	204 204 204
SYSCLR_MENUDISABLEDTEXT	Dark gray	128 128 128
SYSCLR_MENUHILITE	Black	0 0 0
SYSCLR_MENUHILITEBGND	Light grey	204 204 204
SYSCLR_MENUTEXT	Black	0 0 0
SYSCLR_OUTPUTTEXT	Black	0 0 0
SYSCLR_PAGEBACKGROUND	White	255 255 255

System Color Index	Default Color	Default RGB Values
SYSCLR_SCROLLBAR	Pale gray	192 192 192
SYSCLR_SHADOW	Dark gray	128 128 128
SYSCLR_SHADOWHILITEBGND	Dark gray	128 128 128
SYSCLR_SHADOWHILITEFGND	White	255 255 255
SYSCLR_SHADOWTEXT	Dark gray	128 128 128
SYSCLR_TITLEBOTTOM	Dark gray	128 128 128
SYSCLR_TITLETEXT	White	255 255 255
SYSCLR_WINDOW	White	255 255 255
SYSCLR_WINDOWFRAME	Dark gray	128 128 128
SYSCLR_WINDOWSTATICTEXT	Blue	0 0 128
SYSCLR_WINDOWTEXT	Black	0 0 0

Related Functions

- WinQuerySysColor

Related Messages

- WM_SYSCOLORCHANGE

Example Code

This example changes the desktop background to blue and the output text to green.

```
#define INCL_WINSYS
#define INCL_GPILOGCOLORTABLE
#include <OS2.H>

typedef struct {
LONG index;
LONG color;
               } ENTRY;
LONG R, G ,B;

ENTRY alTable[2];  /* array of two color/index entries. */

R = 5L; G = 5L; B = 200L;
alTable[0].color = (R * 65536L) + (G * 256L) + B;
R = 5; G = 200; B = 5;
alTable[1].color = (R * 65536L) + (G * 256L) + B;
alTable[0].index = SYSCLR_OUTPUTTEXT; /* output text. */
alTable[1].index = SYSCLR_BACKGROUND; /* desktop background. */

WinSetSysColors (HWND_DESKTOP,
                 LCOL_RESET,     /* reset system colors before   */
                                 /* processing remainder of this */
                                 /* call.                        */
                 LCOLF_INDRGB,   /* Array of (index,RGB) */
                                 /* values. Each pair of */
                                 /* entries is 8 bytes   */
                                 /* long, comprising 4   */
                                 /* bytes for the index, */
                                 /* and 4 bytes for the  */
                                 /* color value. For     */
                                 /* system color indexes,*/
                                 /* see lStart.          */
                 0L,             /* not applicable.      */
                (ULONG)4,
                (PLONG)&alTable[0].index);
```

WinSetSysModalWindow

This function makes a window become the system-modal window, or ends the system-modal state.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetSysModalWindow  (HWND hwndDesktop, HWND hwnd)
```

Parameters

hwndDesktop (HWND) – input
Desktop-window handle, or HWND_DESKTOP.

hwnd (HWND) – input
Handle of window to become system-modal window.

If NULLHANDLE, system-modal state is ended, and input processing returns to its normal state.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

Input processing can enter a "system modal" state. In this state, all pointing device and keyboard input is directed to a special window, known as the system-modal window, or to one of its child windows (or a window owned by one of them). An "owned" window is a window that refers to its owner window set by using either the *hwndOwner* parameter of the WinCreateWindow function or the *hwndNewOwner* parameter of the WinSetOwner function. All other main windows behave as though they are disabled and no interaction is possible with them.

Note: The disabled windows are not actually disabled, but made noninteractive. No messages are sent to these windows when the system-modal state is entered or left, and their WS_DISABLE style bits are not changed.

Where a system-modal window exists and another window is explicitly made the active window, the newly activated window becomes the system-modal window. This replaces the old one, which becomes a noninteractive window. When the system-modal window is destroyed, the system-modal state is ended, and input processing returns to its normal state.

This function should only be called while processing pointing device or keyboard input.

The new system-modal window is **not** locked during the processing of this function.

Related Functions

- WinQuerySysModalWindow

Example Code

This example uses the WinSetModalWindow to set a system modal window.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND hwndSysModal;

/*  Input processing can enter a "system modal" state. In */
/*  this state, all pointing  device and keyboard input   */
/*  is directed to a special window, known as the         */
/*  system-modal window. Typically, this will be a dialog */
/*  window requiring input.                               */

WinSetSysModalWindow(HWND_DESKTOP,hwndSysModal);
```

WinSetSysPointerData

This function is specific to version 2.1, or higher, of the OS/2 operating system. This function sets the given system pointer to the new icon specified by the ICONINFO structure.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetSysPointerData (HWND hwndDesktop, ULONG iptr,
                           PICONINFO pIconInfo)
```

Parameters

hwndDesktop (HWND) – input
Handle to the desktop window.

iptr (ULONG) – input
Index of the desired system pointer.

pIconInfo (PICONINFO) – in/out
Icon data.

New icon data for the requested system pointer or NULL to reset the system pointer to its default appearance.

Returns

rc (BOOL) – returns
Return value.

TRUE	Successful.
FALSE	An error occurred.

Remarks

This function sets the given system pointer to the new icon specified by the ICONINFO structure that is passed in. Provided that the icon is valid, the screen will be refreshed with the updated system pointer if necessary.

If NULL is passed for the *pIconInfo* parameter, the operating system reverts back to the default pointer shapes defined by the system. All alterations made using WinSetSysPointerData are persistent. They are preserved across IPLs of the system.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinQuerySysPointerData
- WinSetPointer
- WinSetPointerPos
- WinShowPointer

WinSetSysValue

This function sets a system value.

Syntax

```
#define INCL_WINSYS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetSysValue  (HWND hwndDeskTop, LONG iSysValue, LONG lValue)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	Set the system values for the desktop-window handle
Other	Set the system values for the specified desktop-window handle.

iSysValue (LONG) – input
System-value identity.

This must be a valid SV_* value (see WinQuerySysValue). The following values can be set:

SV_ALARM	TRUE enables the alarm sound generated by WinAlarm FALSE disables the alarm sound
SV_ALTMNEMONIC	
SV_ANIMATION	(*) TRUE when animation is set on. FALSE when animation is set off.
SV_CHORDTIME	
SV_CICONTEXTLINES	
SV_CURSORRATE	Cursor blink rate, in milliseconds
SV_CXCHORD	
SV_CXDBLCLK	Width of the mouse double-click sensitive area
SV_CXICONTEXTWIDTH	
SV_CXMOTIONSTART	The width in pels that a pointing device must be moved in the horizontal direction, while the button is depressed, before a WM_BUTTONxMOTIONSTR message is sent.
SV_CXSIZEBORDER	Width of the sizing border
SV_CYCHORD	

SV_CYDBLCLK	Height of the mouse double-click sensitive area
SV_CYMOTIONSTART	The height in pels that a pointing device must be moved in the vertical direction, while the button is depressed, before a WM_BUTTONxMOTIONSTR message is sent.
SV_CYSIZEBORDER	Height of the sizing border
SV_DBLCLKTIME	Mouse double-click time, in milliseconds
SV_ERRORDURATION	Duration for error alarms generated by WinAlarm
SV_ERRORFREQ	Frequency for error alarms generated by WinAlarm
SV_EXTRAKEYBEEP	
SV_FIRSTSCROLLRATE	Delay (in milliseconds) before autoscrolling starts, when using a scroll bar
SV_INSERTMODE	When TRUE, the system is in insert mode.
SV_MENUROLLDOWNDELAY	Delay in milliseconds before displaying a pulldown referred to from a submenu item, when the button is already down as the pointer moves onto the submenu item.
SV_MENUROLLUPDELAY	Delay in milliseconds before hiding a pulldown referred to from a submenu item, when the button is already down as the pointer moves off the submenu item.
SV_NOTEDURATION	Duration for note alarms generated by WinAlarm
SV_NOTEFREQ	Frequency for note alarms generated by WinAlarm
SV_NUMBEREDLISTS	
SV_PRINTSCREEN	
	TRUE when the Print Screen function is enabled; FALSE when the Print Screen function is disabled.
SV_SCROLLRATE	Delay (in milliseconds) between scroll operations, when using a scroll bar
SV_SETLIGHTS	When TRUE, the appropriate light is set when the keyboard state table is set.
SV_SWAPBUTTON	TRUE when the mouse buttons are set for left-handed use
SV_TASKLISTMOUSEACCESS	
SV_WARNINGDURATION	Duration of warning alarms generated by WinAlarm
SV_WARNINGFREQ	Frequency for warning alarms generated by WinAlarm

SV_BEGINDRAG	Mouse begin drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*).
SV_ENDDRAG	Mouse end drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*).
SV_SINGLESELECT	
SV_OPEN	Mouse open drag (low word=mouse message id (WM_*), high word=keyboard control code (KC_*).
SV_BEGINSELECT	Mouse begin swipe select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*))
SV_ENDSELECT	Mouse select or end swipe select (low word=mouse message id (WM_*), high word=keyboard control code (KC_*))
SV_CONTEXTMENU	Mouse request popup menu (low word=mouse message id (WM_*), high word=keyboard control code (KC_*))
SV_TEXTEDIT	Mouse begin direct name edit (low word=mouse message id (WM_*), high word=keyboard control code (KC_*))
SV_CONTEXTMENUKB	Keyboard request popup menu (low word=virtual key code (VK_*), high word=keyboard control code (KC_*))
SV_TEXTEDITKB	Keyboard begin direct name edit (low word=virtual key code (VK_*), high word=keyboard control code (KC_*))
SV_CONTEXTHELP	
SV_BEGINDRAGKB	
SV_ENDDRAGKB	
SV_SELECTKB	
SV_OPENKB	
SV_CONTEXTHELPKB	
SV_BEGINSELECTKB	
SV_ENDSELECTKB	
SV_LOCKSTARTINPUT	
SV_MONOICONS	When TRUE preference is given to black and white icons when selecting which icon resource definition to use on the screen. Black and white icons may have more clarity than color icons on LCD and Plasma display screens.

SV_KBDALTERED	Hardware ID of the newly attached keyboard. **Note:** The OS/2 National Language Support is only loaded once per system IPL. The OS/2 NLS translation is based partially on the type of keyboard device attached to the system. There are two main keyboard device types: PC AT styled and Enhanced styled. Hot Plugging between these two types of devices may result in typing anomalies due to a mismatch in the NLS device tables loaded and that of the attached device. It is strongly recommended that keyboard hot plugging be limited to the device type that the system was IPL'd with. In addition, OS/2 support will default to the 101/102 key Enhanced keyboard if no keyboard or a NetServer Mode password was in use during system IPL. (See Category 4, IOCtls 77h and 7Ah for more information on keyboard devices and types.)

lValue (LONG) – input
The system value.

Dimensions are in pels and times are in milliseconds.

Returns

rc (BOOL) – returns
Value-set indicator.

TRUE	System value set
FALSE	An error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Related Functions

- WinQuerySysValue

Example Code

This example uses the WinSetSysValue call change the sizing border dimensions.

```
#define INCL_WINSYS
#include <OS2.H>
LONG vlXBorder, vlYBorder;

vlXBorder = WinSetSysValue(HWND_DESKTOP,
                           SV_CXSIZEBORDER,
                           20L);
vlYBorder = WinSetSysValue(HWND_DESKTOP,
                           SV_CYSIZEBORDER,
                           20L);
```

WinSetVisibleRegionNotify

This function allows an application to request that a given window receive notifications every time that its visible region gets altered.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetVisibleRegionNotify  (HWND hwnd, BOOL fEnable)
```

Parameters

hwnd (HWND) – input
The window handle.

fEnable (BOOL) – input
Enable flag.

TRUE The specified window wants to receive notifications every time its visible region is modified.

FALSE The specified window no longer wants to receive any notification messages concerning its visible region.

Returns

fSuccess (BOOL) – returns
Success indicator.

TRUE Successful completion
FALSE Error occurred.

Remarks

This call causes a notification message to be sent to the specified window every time its visible area is modified on the screen. In addition, the window is notified when an application issues a WinLockWindowUpdate call that affects it, so that any asynchronous drawing activity can be suspended.

Applications that blit to the screen memory directly can use the notification messages WM_VRNENABLED and WM_VRNDISABLED to determine when to suspend blitting and also to determine when the clipping areas for the visible region have been modified.

Use WinQueryVisibleRegion to obtain the current visible region when a window is notified that its visible region has changed.

Related Functions

- WinLockWindowUpdate
- WinQueryVisibleRegion

WinSetWindowBits

This function sets a number of bits into the memory of the reserved window words.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetWindowBits (HWND hwnd, LONG index, ULONG flData,
                       ULONG flMask)
```

Parameters

hwnd (HWND) – input
Window handle.

index (LONG) – input
Zero-based index of the value to be set.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* –4), where *cbWindowData* is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage. Any of the following QWL_* values can be specified:

QWL_HMQ	Handle of message queue of window. Note that the leading 16 bits of this value are zero.
QWL_STYLE	Window style.
QWL_HWNDFOCUSSAVE	Window handle of the child windows of this window that last possessed the focus when this frame window was last deactivated.
QWL_USER	A ULONG value for applications to use is present at offset QWL_USER in windows of the following preregistered window classes: WC_FRAME (includes dialog windows) WC_COMBOBOX WC_BUTTON WC_MENU WC_STATIC WC_ENTRYFIELD WC_LISTBOX WC_SCROLLBAR WC_TITTLEBAR WC_MLE WC_SPINBUTTON WC_CONTAINER

WC_SLIDER
WC_VALUESET
WC_NOTEBOOK

This value can be used to place application-specific data in controls.

QWL_DEFBUTTON — The default push button for a dialog.

The default push button is the one that sends its WM_COMMAND message when the enter key is pressed.

QWL_PENDATA — Reserved for use by operating system extensions. It allows an operating system extension to store data on a per window basis.

Other — Zero-based index.

flData (ULONG) – input
Bit data to store in the window words.

This is done under the control of the *flMask* parameter.

flMask (ULONG) – input
Bits to be written indicator.

A "1" bit indicates that the corresponding bit of the *flData* parameter is to be stored into the window word. A "0" bit indicates that the corresponding bit of the *flData* parameter is to be ignored in the storing operation; the value of that bit position in the window word is unaltered.

Returns

rc (BOOL) – returns
Success indicator.

TRUE Successful completion
FALSE Error occurred.

Remarks

The bits are set in a single operation.

Related Functions

- WinQueryWindowPtr
- WinQueryWindowULong
- WinQueryWindowUShort
- WinSetWindowPtr
- WinSetWindowULong
- WinSetWindowUShort

Example Code

This example uses the WinSetWindowBits call to change the attributes of a list box so that only one item can be be selected. This is done by turning off the multiple-select bit.

```
#define INCL_WINSYS
#include <OS2.H>
HWND  hwndMessageLB;
WinSetWindowBits(hwndMessageLB,
                 QWL_STYLE,        /* change style bit. */
                 0L,               /* set to 0.         */
                 LS_MULTIPLESEL);  /* multiple select bit.  */
```

WinSetWindowPos

This function allows the general positioning of a window.

Note: Messages may be received from other processes or threads during the processing of this function.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetWindowPos (HWND hwnd, HWND hwndInsertBehind, LONG x,
                      LONG y, LONG cx, LONG cy, ULONG fl)
```

Parameters

hwnd (HWND) – input
Window handle.

hwndInsertBehind (HWND) – input
Relative window-placement order.

Ignored if SWP_ZORDER is not selected. Values that can be specified are:

HWND_TOP	Place *hwnd* on top of all siblings
HWND_BOTTOM	Place *hwnd* behind all siblings
Other	Identifies the sibling window behind which *hwnd* is to be placed.

x (LONG) – input
Window position, x-coordinate.

This is the x-coordinate of *hwnd*. It is in window coordinates relative to the bottom left corner of its parent, but is ignored if SWP_MOVE is not selected.

y (LONG) – input
Window position, y-coordinate.

This is the y-coordinate of *hwnd*. It is in window coordinates relative to the bottom left corner of its parent, but is ignored if SWP_MOVE is not selected.

cx (LONG) – input
Window size.

This specifies the width of *hwnd* in device units, but is ignored if SWP_SIZE is not selected.

cy (LONG) – input
Window size.

This specifies the depth of *hwnd* in device units, but is ignored if SWP_SIZE is not selected.

fl (ULONG) – input
Window-positioning options.

One or more of these options can be specified:

SWP_SIZE	Change the window size.
SWP_MOVE	Change the window x,y position.
SWP_ZORDER	Change the relative window placement.
SWP_SHOW	Show the window.
SWP_HIDE	Hide the window.
SWP_NOREDRAW	Changes are not redrawn.
SWP_NOADJUST	Do not send a WM_ADJUSTWINDOWPOS message before moving or sizing.
SWP_ACTIVATE	Activate the *hwnd* window if it is a frame window. This indicator has no effect on other windows. The frame window is made the topmost window, unless SWP_ZORDER is specified also in which instance the *hwndInsertBehind* window is used.
SWP_DEACTIVATE	Deactivate the *hwnd* window if it is a frame window. This indicator has no effect on other windows. The frame window is made the bottommost window, unless SWP_ZORDER is specified, in which instance the *hwndInsertBehind* window is used.
SWP_MINIMIZE	Minimize the window. This indicator has no effect if the window is in a minimized state, and is also mutually exclusive with SWP_MAXIMIZE and SWP_RESTORE.
SWP_MAXIMIZE	Maximize the window. This indicator has no effect if the window is in a maximized state, and is also mutually exclusive with SWP_MINIMIZE and SWP_RESTORE.
SWP_RESTORE	Restore the window. This indicator has no effect if the window is in its normal state, and is also mutually exclusive with SWP_MINIMIZE and SWP_MAXIMIZE. The position and size of the window in its normal state is remembered in its window words when it is first maximized or minimized, although these values can be altered by use of the WinSetWindowUShort function. The window is restored to the position and size remembered in its window words, unless the SWP_MOVE or SWP_SIZE indicators are set. These indicators cause the position and size values specified in this function to be used.

Returns

rc (BOOL) – returns
Repositioning indicator.

TRUE	Window successfully repositioned
FALSE	Window not successfully repositioned.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

If a window created with the CS_SAVEBITS style is reduced, the screen image saved is used to redraw the area uncovered when the window size changes, if those bits are still valid.

If the CS_SIZEREDRAW style is present, the entire window area is assumed invalid if sized. Otherwise, WM_CALCVALIDRECTS is sent to the window to inform the window manager which bits it may be possible to preserve.

Messages sent from WinSetWindowPos and WinSetMultWindowPos have specific orderings within the window-positioning process. The process begins with redundancy checks and precalculations on every window for each requested operation. For example, if SWP_SHOW is present but the window is already visible, SWP_SHOW is turned off. If SWP_SIZE is present, and the new size is equal to the old size, SWP_SIZE is turned off.

If the operations create new results, the information is calculated and stored (for instance, when sizing or moving, the new window rectangle is stored for later use). It is at this point that the WM_ADJUSTWINDOWPOS message is sent to any window that is sizing or moving. It is also at this point that the WM_CALCVALIDRECTS message is sent to any window that is sizing and does not have the CS_SIZEREDRAW window style.

When all the new window states are calculated, the window-management process begins. Window areas that can be preserved are moved from the old to the new positions, window areas that are invalidated by these operations are calculated and distributed as update regions. When this is finished, and before any synchronous-paint windows are repainted, the WM_SIZE message is sent to any windows that have changed size. Next, all the synchronous-paint windows that can be are repainted, and the process is complete.

If a synchronous-paint parent window has a size-sensitive area displayed that includes synchronous-paint child windows, the parent needs to reposition those windows when it receives the WM_SIZE message. Their invalid regions are added to the parent's invalid region, resulting in one update after the parent's WM_SIZE message, rather than many independent (and later, duplicated) updates.

Note: Some windows will not be positioned precisely to the parameters of this function, but according to the behavior of their window procedure. For example, frame windows without a style creation flag of FCF_NOBYTEALIGN will not position to any specific screen coordinate. Similarly, frame windows with zero size and position are created by the WinCreateStdWindow function and therefore these values are treated as a special case by the frame window procedure.

Messages sent by this function are:

WM_ACTIVATE	Sent if a different window becomes the active window. See also WinSetActiveWindow.
WM_ADJUSTWINDOWPOS	Not sent if SWP_NOADJUST is specified. The message contains an SWP structure that has been filled in by this function with the proposed move/size data. The window can adjust this new position by changing the contents of the SWP structure. If *hwnd* specifies a frame window, this function recalculates the sizes and positions of the frame controls. If the new window rectangle for any frame control is empty, instead of resizing or repositioning that control, it is hidden if SWP_HIDE is specified. This eliminates needless processing of windows that are not visible. The window rectangle of the control in question is left in its original state. For example, if WinSetWindowPos is issued to change the size of a standard-frame window to an empty rectangle, and WinQueryWindowRect is issued against the client window, the rectangle returned is not an empty rectangle, but the original client rectangle before WinSetWindowPos was issued.
WM_CALCVALIDRECTS	Sent to determine the area of a window that may be possible to preserve as the window is sized.
WM_SIZE	Sent if the size of the window has changed, after the change has been made.
WM_MOVE	Sent when a window with CS_MOVENOTIFY class style moves its absolute position.

Related Functions

- WinGetMinPosition
- WinQueryActiveWindow
- WinQueryWindowPos
- WinSaveWindowPos
- WinSetActiveWindow
- WinSetMultWindowPos

Related Messages

- WM_ACTIVATE
- WM_ADJUSTWINDOWPOS
- WM_CALCVALIDRECTS
- WM_ERASEBACKGROUND
- WM_MOVE
- WM_SIZE

Example Code

This example uses the recommended size, position and status from the WinQueryTaskSize call to position the first window of a newly-started application (typically the main window).

```
#define INCL_WINSWITCHLIST
#define INCL_WINFRAMEMGR
#include <OS2.H>

HAB  hab;
SWP  winpos;
HWND hwndFrame;

WinQueryTaskSizePos(hab, 0, &winpos);

 WinSetWindowPos(hwndFrame, HWND_TOP,
     winpos.x,                                        /* x pos */
     winpos.y,                                        /* y pos */
     winpos.cx,                                       /* x size */
     winpos.cy,                                       /* y size */
     SWP_ACTIVATE | SWP_MOVE | SWP_SIZE | SWP_SHOW); /* flags*/
```

WinSetWindowPtr

This function sets a pointer value into the memory of the reserved window words.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetWindowPtr  (HWND hwnd, LONG lb, PVOID pp)
```

Parameters

hwnd (HWND) – input
Window handle.

lb (LONG) – input
Zero-based index into the window words.

The units of b are bytes. Valid values are zero through (*cbWindowData* –4), where *cbWindowData* is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage.

The value QWP_PFNWP can be used as the index for the address of the window procedure for the window.

pp (PVOID) – input
Pointer value to store in the window words.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Related Functions

- WinQueryWindowPtr
- WinQueryWindowULong
- WinQueryWindowUShort
- WinSetWindowBits
- WinSetWindowULong
- WinSetWindowUShort

Example Code

This function retrieves a pointer value from the memory of the reserved window word.

```
MyWindowProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
MYINSTANCEDATA *InstanceData;  /* application defined structure */

    switch (msg) {
    case WM_CREATE:
       DosAllocMem(&InstanceData, sizeof(MYINSTANCEDATA), fALLOC);
       /* WindowProcedure initializes instance data for this window */
       .
       .
       /* set pointer to instance in window words */
       WinSetWindowPtr(hwnd, 0, InstanceData);
       break;

    case WM_USER + 1: /* application defined message */
       /* Window procedure retrieves instance data to   */
       /* process this message                          */

       InstanceData = WinQueryWindowPtr(hwnd, 0);
       .
       .

       break;
       .
       .
```

WinSetWindowText

This function sets the window text for a specified window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetWindowText (HWND hwnd, PSZ pszString)
```

Parameters

hwnd (HWND) – input
Window handle.

pszString (PSZ) – input
Window text.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Text updated
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function sends a WM_SETWINDOWPARAMS message to the window identified by *hwnd*.

If this function references the window of another process, *pszString* must be in memory that is shared by both processes; otherwise, a memory error may occur.

If *hwnd* has a style of WS_FRAME, the title-bar window text is set.

Some window classes interpret the *pszString* in a special way. The tilde character (~) indicates that the following character is a mnemonic; for details, see "Button Control Window Processing" and "Menu Control Window Processing" in the *Presentation Manager Programming Reference Volume II*.

Related Functions

- WinQueryDlgItemShort
- WinQueryDlgItemText
- WinQueryDlgItemTextLength
- WinQueryWindowText
- WinQueryWindowTextLength
- WinSetDlgItemShort
- WinSetDlgItemText

Related Messages

- WM_SETWINDOWPARAMS

Example Code

This example calls WinQuerySessionTitle to retrieve the application's title, and then sets the title bar of the frame window to that title with WinSetWindowText.

```
#define INCL_WINMESSAGEMGR
#define INCL_WINWINDOWMGR
#include <OS2.H>
HAB hab;
HWND hwndFrame, hwndClient;
CHAR szTitle[MAXNAMEL + 1];

WinQuerySessionTitle(hab,
                     0, szTitle,
                     sizeof(szTitle));

hwndFrame = WinQueryWindow(hwndClient,
                           QW_PARENT); /* get handle of parent, */
                                       /* which is frame window. */
WinSetWindowText(hwndFrame, szTitle);
```

WinSetWindowThunkProc

This function associates a pointer-conversion procedure with a window.

Syntax

```
#define INCL_WINTHUNKAPI /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetWindowThunkProc  (HWND hwnd, PFN pthunkpr)
```

Parameters

hwnd (HWND) – input
Window handle.

pthunkpr (PFN) – input
Pointer-conversion procedure identifier.

NULL	Any existing pointer-conversion procedure is dissociated from this window.
Other	The pointer-conversion procedure to be associated with this window.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	An error occurred.

Related Functions

- WinQueryClassThunkProc
- WinQueryWindowModel
- WinQueryWindowThunkProc
- WinSetClassThunkProc

Example Code

In this example, any thunking procedure is dissociated from the window.

```
#define INCL_WINTHUNKAPI
#include <OS2.H>

HWND hwnd;

WinSetWindowThunkProc(hwnd, NULL);
```

WinSetWindowULong

This function sets an unsigned, long integer value into the memory of the reserved window words.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetWindowULong  (HWND hwnd, LONG index, ULONG ul)
```

Parameters

hwnd (HWND) – input
Window handle.

index (LONG) – input
Zero-based index of the value to be set.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* –4), where *cbWindowData* is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage. Any of the QWL_* values are also valid.

Note: QWS_* values cannot be used.

QWL_HMQ	Handle of message queue of window. Note that the leading 16 bits of this value are zero.
QWL_STYLE	Window style.
QWL_HHEAP	Heap handle used by child windows of this window.
QWL_HWNDFOCUSSAVE	Window handle of the child windows of this window that last possessed the focus when this frame window was last deactivated.
QWL_USER	A ULONG value for applications to use is present at offset QWL_USER in windows of the following preregistered window classes: WC_FRAME (includes dialog windows) WC_LISTBOX WC_BUTTON WC_STATIC WC_ENTRYFIELD WC_SCROLLBAR WC_MENU This value can be used to place application-specific data in controls.

QWL_DEFBUTTON	The default push button for a dialog. The default push button is the one that sends its WM_COMMAND message when the enter key is pressed.
QWL_PENDATA	Reserved for use by operating system extensions. It allows an operating system extension to store data on a per window basis.
Other	Zero-based index.

ul (ULONG) – input
Unsigned, long integer value to store in the window words.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE (0x1003)	The value of a parameter was not within the defined valid range for that parameter.

Remarks

The specified *index* is valid only if all of the bytes referenced are within the reserved memory.

Related Functions

- WinQueryWindowPtr
- WinQueryWindowULong
- WinQueryWindowUShort
- WinSetWindowBits
- WinSetWindowPtr
- WinSetWindowUShort

Example Code

This example transfers a pointer from the application-defined data area of a dialog window to the application-defined data area (window word) of a main window. The pointer is then retrieved.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND hwndClient;
ULONG  msg;
MPARAM pParm, mp1, mp2;

/* inside dialog procedure. */
switch( msg )
{
  case WM_INITDLG:

      pParm = (MPARAM)mp2;  /* This points to the data  */
                            /* area and is passed by    */
                            /* the WinLoadDlg,          */
                            /* WinCreateDlg, and        */
                            /* WinDlgBox calls in their */
                            /* pCreateParams            */
                            /* parameter.               */

WinSetWindowULong(hwndClient,     /* place pointer in window */
                  QWL_USER,       /* word area.              */
                  (ULONG) pParm);


  case WM_COMMAND:
        switch ( SHORT1FROMMP( mp1 ) )
        {
         case DID_OK:

                                   /* retrieve  pointer from  */
                                   /* window word area.       */
          pParm = (MPARAM)WinQueryWindowULong(hwndClient,
                                              QWL_USER);
        }
}
```

WinSetWindowUShort

This function sets an unsigned, short integer value into the memory of the reserved window words.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSetWindowUShort (HWND hwnd, LONG index, USHORT us)
```

Parameters

hwnd (HWND) – input
Window handle.

index (LONG) – input
Zero-based index of the value to be set.

The units of *index* are bytes. Valid values are zero through (*cbWindowData* –2), where *cbWindowData* is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage.

QWL_* values cannot be used. Any of the following QWS__* values can be specified:

QWL_HMQ	Handle of message queue of window. Note that the leading 16 bits of this value are zero.
QWL_STYLE	Window style.
QWL_HWNDFOCUSSAVE	Window handle of the child windows of this window that last possessed the focus when this frame window was last deactivated.
QWL_USER	A ULONG value for applications to use is present at offset QWL_USER in windows of the following preregistered window classes: WC_FRAME (includes dialog windows) WC_COMBOBOX WC_BUTTON WC_MENU WC_STATIC WC_ENTRYFIELD WC_LISTBOX WC_SCROLLBAR WC_TITTLEBAR WC_MLE WC_SPINBUTTON

WC_CONTAINER
WC_SLIDER
WC_VALUESET
WC_NOTEBOOK

This value can be used to place application-specific data in controls.

QWL_DEFBUTTON — The default push button for a dialog.

The default push button is the one that sends its WM_COMMAND message when the enter key is pressed.

QWL_PENDATA — Reserved for use by operating system extensions. It allows an operating system extension to store data on a per window basis.

Other — Zero-based index.

us (USHORT) – input
Unsigned, short integer value to store in the window words.

Returns

rc (BOOL) – returns
Success indicator.

TRUE Successful completion
FALSE Error occurred.

Related Functions

- WinQueryWindowPtr
- WinQueryWindowULong
- WinQueryWindowUShort
- WinSetWindowBits
- WinSetWindowPtr
- WinSetWindowULong

Example Code

This example changes the height to which a window is restored to 100 by changing the value of a system defined window word.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwnd;

WinSetWindowUShort(hwnd,
                   QWS_CYRESTORE,  /* The height to which     */
                                   /* the window is restored. */
                   (USHORT)100);
```

WinShowCursor

This function shows or hides the cursor that is associated with a specified window.

Syntax

```
#define INCL_WINCURSORS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinShowCursor (HWND hwnd, BOOL fShow)
```

Parameters

hwnd (HWND) – input
Handle of window to which the cursor belongs.

fShow (BOOL) – input
Show indicator.

TRUE	Make cursor visible
FALSE	Make cursor invisible.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or an attempt was made to show the cursor when it was already visible.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function must be called by the same thread that created the cursor that is affected.

A cursor show-level count is maintained. It is incremented by a hide operation and decremented by a show operation. The cursor is actually visible if the cursor show-level count is zero, otherwise it is invisible. When decrementing, the cursor show-level count is fixed at zero so as not to show the cursor too many times, but it is possible to hide the cursor a number of times in succession.

Related Functions

- WinCreateCursor
- WinDestroyCursor
- WinQueryCursorInfo

Example Code

This example shows the cursor if it is successfully created.

```
#define INCL_WINCURSORS
#include <OS2.H>

HWND  hwnd; /* handle of window that has pointer captured */
RECTL rcl;

WinQueryWindowRect(hwnd, &rcl);

      if (WinCreateCursor(hwnd,      /* This must be the handle */
                                     /* of a window for which   */
                                     /* the application can     */
                                     /* receive input.          */
                          100,       /* x,y position of cursor. */
                          100,
                          5,         /* cursor width.           */
                          5,         /* cursor height.          */
                          CURSOR_FLASH,
                          &rcl)) /* region where the cursor */
                                     /* is visible.             */
      WinShowCursor(hwnd,
                    TRUE);           /* make cursor visible. */
```

WinShowPointer

This function adjusts the pointer display level to show or hide a pointer.

Syntax

```
#define INCL_WINPOINTERS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinShowPointer (HWND hwndDeskTop, BOOL fShow)
```

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop-window handle
Other	The specified desktop-window handle.

fShow (BOOL) – input
Level-update indicator.

TRUE	Decrement pointer display level by one. (The pointer level is not decremented to a negative value.)
FALSE	Increment pointer display level by one.

Returns

rc (BOOL) – returns
Display-level-updated indicator.

TRUE	Pointer display level not successfully updated.
FALSE	Pointer display level successfully updated

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The pointer display level determines whether the pointer is shown. If it is zero, the pointer is visible, but if it is greater than zero, the pointer is not visible. The initial setting of the pointer display level is dependent on the capabilities of the device. If a pointing device exists, the initial setting of the pointer display level is zero, otherwise it is one. The existing pointer display level can be obtained by using the WinQuerySysValue function with *iSysValue* set to SV_POINTERLEVEL.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinQuerySysPointerData
- WinSetPointer
- WinSetPointerPos
- WinSetSysPointerData

Example Code

This example obtains the pointer handle from the desktop window handle and hides the pointer.

```
#define INCL_WINPOINTERS
#define INCL_WINDESKTOP
#include <OS2.H>
HPOINTER hpointer;
HWND hwnd;

hpointer = WinQueryPointer(HWND_DESKTOP);

WinShowPointer(hwnd,FALSE);
```

WinShowTrackRect

This function hides or shows the tracking rectangle.

Syntax

```
#define INCL_WINTRACKRECT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinShowTrackRect  (HWND hwnd, BOOL fShow)
```

Parameters

hwnd (HWND) – input
Window handle.

Passed to the WinTrackRect function.

fShow (BOOL) – input
Show indicator.

TRUE	Show the tracking rectangle
FALSE	Hide the tracking rectangle.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function maintains a show count. When a hide request is made, this count is decremented; when a show request is made, the count is incremented. When the count makes a transition from 0 to -1, the rectangle is hidden; when the count makes a transition from -1 to 0, the rectangle is shown.

When a rectangle is tracking, the application must call this function to hide the rectangle if there is a possibility of corrupting the tracking rectangle while drawing. The rectangle is shown afterwards. Because the *rclTrack* structure is updated continuously, the application can examine the coordinates of the current tracking rectangle to determine whether temporary hiding is necessary.

The only case where an application needs to use this function is during asynchronous drawing. If an application is drawing on one thread, and issuing WinTrackRect on another, unwanted areas of tracking rectangle may be left behind. The drawing thread is therefore responsible for calling this function whenever tracking is in progress. The application must provide for communication between the two threads to ensure that if one thread is tracking, the drawing thread issues this function. This can be done with a *semaphore*.

Related Functions

- WinTrackRect

WinShowWindow

This function sets the visibility state of a window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinShowWindow (HWND hwnd, BOOL fNewVisibility)
```

Parameters

hwnd (HWND) – input
Window handle.

fNewVisibility (BOOL) – input
New visibility state.

TRUE	Set window state visible
FALSE	Set window state invisible.

Returns

rc (BOOL) – returns
Visibility changed indicator.

TRUE	Window visibility successfully changed
FALSE	Window visibility not successfully changed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

A window possesses a visibility state indicated by the WS_VISIBLE style bit. When the WS_VISIBLE style bit is set, the window is shown and subsequent drawing into the window is presented, unless that window is obscured by some other window, or at least one of the windows upward in the parent chain from *hwnd* does not have the WS_VISIBLE style.

When the WS_VISIBLE style bit is not set, the window is not shown ("hidden") and subsequent drawing into the window is not presented, even if that window is not obscured by another window.

If the value of the WS_VISIBLE style bit has been changed, the WM_SHOW message is sent to the window of *hwnd* before the function returns.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinUpdateWindow
- WinValidateRect
- WinValidateRegion

Related Messages

- WM_SHOW

Example Code

This example uses the WinShowWindow call to make a modeless dialog window visible.

```
#define INCL_WINWINDOWMGR
#define INCL_WINDIALOGS
#include <OS2.H>
#define DLG_MODELESS 900
          /* dialog procedure declaration. */
MRESULT EXPENTRY DlgProc( HWND hwndDlg, ULONG msg, MPARAM mp1, MPARAM mp2 );
HWND hwnd;

  hwnd = WinLoadDlg( HWND_DESKTOP,
                     HWND_OBJECT,
                     (PFNWP)DlgProc,
                     (HMODULE)NULL,
                     DLG_MODELESS,
                     NULL);

/*
DlgProc( HWND hwndDlg, ULONG msg, MPARAM mp1,
                                  MPARAM mp2 )
{
CASE USER_DEFINED:

*/
      WinShowWindow( hwnd,
                     TRUE );     /* show window. */
      WinSetFocus( HWND_DESKTOP, hwnd );
}
```

WinShutdownSystem

The WinShutdownSystem function will close down the system.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinShutdownSystem  (HAB hab, HMQ hmq)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hmq (HMQ) – input
Message-queue handle.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The WinShutdownSystem function will close all running applications and will then call DosShutdown.

Presentation Manager applications will receive a WM_SAVEAPPLICATION message prior to a WM_QUIT message.

When the system is restarted, all applications that were running when WinShutdownSystem was last called will be restarted.

Related Functions

- WinCancelShutdown

WinStartApp

This function starts an application.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HAPP WinStartApp (HWND hwndNotify, PPROGDETAILS pDetails, PSZ pParams,
                  PVOID pReserved, ULONG ulOptions)
```

Parameters

hwndNotify (HWND) – input
Notification-window handle.

A WM_APPTERMINATENOTIFY message is posted to this window, when the started application terminates.

NULLHANDLE	Do not post the notification message
Other	Post the notification message to this window.

pDetails (PPROGDETAILS) – input
Program list structure.

pParams (PSZ) – input
Input parameters for the application to be started.

This specifies the command line parameters to be passed to this application when it starts.

NULL	There are no parameters to be passed to the application
Other	The parameters to be passed to the application.

pReserved (PVOID) – input
Start data.

Reserved, must be NULL.

ulOptions (ULONG) – input
Option indicators.

If more than one option is selected, the values can be ORed together.

0	No options selected.
SAF_INSTALLEDCMDLINE	The command line parameters installed in the program starter list are used; the *pParams* parameter is ignored.

SAF_STARTCHILDAPP — The specified application is started as a child session of the session from which WinStartApp is issued. The calling application may terminate the called application with a WinTerminateApp function.

Returns

happ (HAPP) – returns
Application handle.

NULL Application not started
Other Application handle.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS (0x1208) — An application parameter value is invalid for its converted PM type. For example: a 4-byte value outside the range -32 768 to +32 767 cannot be converted to a SHORT, and a negative number cannot be converted to a ULONG or USHORT.

PMERR_INVALID_APPL (0x1530) — Attempted to start an application whose type is not recognized by OS/2.

PMERR_INVALID_WINDOW (0x1206) — The window specified with a Window List call is not a valid frame window.

PMERR_STARTED_IN_BACKGROUND (0x1532) — The application started a new session in the background.

PMERR_DOS_ERROR (0x1200) — A DOS call returned an error.

Remarks

Starts the application identified in PROGDETAILS.

If the application is successfully started, the return value is a handle to the application. If SAF_STARTCHILDAPP is specified, this can be used to stop the application (see the WinTerminateApp function).

When the program specified by the application handle terminates, the window specified by the *hwndNotify* parameter (if the window still exists and is valid) has a WM_APPTERMINATENOTIFY message posted to it to notify it of the application termination.

This function requires the existence of a message queue.

Related Functions

- WinTerminateApp

Related Messages

- WM_APPTERMINATENOTIFY

Example Code

This example calls WinStartApp in a typical termination sequence.

```
#define INCL_DOSSESMGR
#include <os2.h>
HWND         hwndNotify;
PPROGDETAILS pDetails;
HAPP         happ;

 pDetails->Length          = sizeof(PROGDETAILS);
 pDetails->progt.progc     = PROG_WINDOWABLEVIO;
 pDetails->progt.fbVisible = SHE_VISIBLE;
 pDetails->pszTitle        = "TEXT";
 pDetails->pszExecutable   = "TEXT.EXE";
 pDetails->pszParameters   = NULL;
 pDetails->pszStartupDir   = "";
 pDetails->pszICON         = "T.ICO";
 pDetails->pszEnvironment  = "WORKPLACE\0\0";
 pDetails->swpInitial.fl   = SWP_ACTIVATE; /* window positioning */
 pDetails->swpInitial.cy   = 0;    /* width of window */
 pDetails->swpInitial.cx   = 0;    /* height of window */
 pDetails->swpInitial.y    = 0;    /* lower edge of window */
 pDetails->swpInitial.x    = 0;    /* left edge of window */
 pDetails->swpInitial.hwndInsertBehind = HWND_TOP;
 pDetails->swpInitial.hwnd             = hwndNotify;
 pDetails->swpInitial.ulReserved1      = 0;
 pDetails->swpInitial.ulReserved2      = 0;

 happ = WinStartApp( hwndNotify,pDetails,NULL,NULL,SAF_STARTCHILDAPP);
        .
        .
        .
 WinTerminateApp(happ);
```

WinStartTimer

This function starts a timer.

Syntax

```
#define INCL_WINTIMER /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinStartTimer  (HAB hab, HWND hwnd, ULONG idTimer,
                      ULONG dtTimeout)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hwnd (HWND) – input
Window handle that is part of the timer identification.

NULLHANDLE	The *idTimer* parameter is ignored, and this function returns a unique, nonzero, identity which represents that timer. The timer message is posted in the queue associated with the current thread, with the *hwnd* parameter of the QMSG structure set to NULLHANDLE.
Other	Window handle.

idTimer (ULONG) – input
Timer identifier.

The value of an application-timer identifier must be below TID_USERMAX to avoid clashes with timers used by the system.

A timer identification, TID_SCROLL, is created by a scroll bar control. An application does not normally see the associated WM_TIMER, but passes it to the scroll-bar control.

A timer identification, TID_CURSOR, is created when the cursor is flashing. An application must ensure that the associated WM_TIMER is passed on to the default window procedure.

dtTimeout (ULONG) – input
Delay time in milliseconds.

For OS/2 Version 3, the value of this parameter must be in the range of 0-4 294 967 295.

For OS/2 2.1 and earlier, the value of this parameter must be in the range of 0-65 535.

Returns

idTimer (ULONG) – returns
Timer identity.

A return value of 0 indicates that an error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function creates a timer identified by *hwnd* and *idTimer*, set to time out every *dtTimeout* milliseconds. When a timer times out, a WM_TIMER message is posted.

A *dtTimeout* value of zero causes the timer to timeout as fast as possible; generally, this is about 1/18 second.

A second call to this function, for a timer that already exists, resets that timer.

Related Functions

- WinGetCurrentTime
- WinQueryMsgTime
- WinStopTimer

Related Messages

- WM_TIMER

Example Code

This example uses the WinStartTimer call to add up elapsed seconds.

```
#define INCL_WINTIMER
#include <OS2.H>
HAB hab;              /* anchor-block handle. */
ULONG seconds;
ULONG  msg;
WinStartTimer(hab,
              (HWND)0,
                    0, /* ignored because previous parameter */
                       /* is null.                           */
                    1000UL);

switch(msg)
{
  case WM_TIMER:
    seconds += 1;
  break;
}
```

WinStopTimer

This function stops a timer.

Syntax

```
#define INCL_WINTIMER /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinStopTimer  (HAB hab, HWND hwnd, ULONG ulTimer)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hwnd (HWND) – input
Window handle.

ulTimer (ULONG) – input
Timer identifier.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred, or timer did not exist.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

When this function is called, no further messages are received from the stopped timer, even if it has timed out since the last call to WinGetMsg.

Related Functions

- WinGetCurrentTime
- WinQueryMsgTime
- WinStartTimer

Example Code

This example uses the WinStopTimer call to stop a clock after one minute.

```
#define INCL_WINTIMER
#include <OS2.H>
HAB hab;              /* anchor-block handle. */
ULONG  ulTimerId;
HWND hwnd;
ulTimerId = WinStartTimer(hab,
             (HWND)0,
                   0, /* ignored because previous parameter */
                      /* is null.                           */
                   1000UL);

BOOL WndProc(. . . . ) {
static ULONG seconds;

  switch(msg)
  {
   case WM_TIMER:
     if (seconds) {
       seconds ++ ;
       if (seconds == 60) WinStopTimer(hab, hwnd, ulTimerId);
     }
     break;
   case WM_CREATE:
     seconds = 0;
        .
        .
        .

}
```

WinStoreWindowPos

The WinStoreWindowPos function will save the current size and position of the window specified by *hwnd*.

Syntax

```
#define INCL_WINWORKPLACE
#include <os2.h>
BOOL WinStoreWindowPos  (PSZ pAppName, PSZ pKeyName, HWND hwnd)
```

Parameters

pAppName (PSZ) – input
Pointer to application name.

A pointer to a zero-terminated string which contains the application name.

pKeyName (PSZ) – input
Pointer to key name.

A pointer to a zero-terminated string which contains the key name.

hwnd (HWND) – input
Window handle for the window to be stored.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function will also save the presentation parameters.

Related Functions

- WinRestoreWindowPos

WinSubclassWindow

This function subclasses the indicated window by replacing its window procedure with another window procedure, specified by *pNewWindowProc*.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
PFNWP WinSubclassWindow  (HWND hwnd, PFNWP pNewWindowProc)
```

Parameters

hwnd (HWND) – input
Handle of window that is being subclassed.

pNewWindowProc (PFNWP) – input
New window procedure.

Window procedure used to subclass *hwnd*.

Returns

pOldWindowProc (PFNWP) – returns
Old window procedure.

Previous window procedure belonging to *hwnd*.

If this function fails, 0L is returned.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

To subclass a window effectively, the new window procedure calls the old window procedure rather than WinDefWindowProc, for those messages it does not process itself.

To reverse the effect of subclassing, call this function again using the old window procedure address.

Note: It is not possible to subclass a window created by another process.

Related Functions

- WinCalcFrameRect
- WinCreateFrameControls
- WinCreateStdWindow

- WinCreateWindow
- WinDefWindowProc
- WinDestroyWindow
- WinQueryClassInfo
- WinQueryClassName
- WinRegisterClass

Example Code

This example uses the WinSubclassWindow call to subclass the frame window procedure, so that frame-sizing restrictions can be implemented.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HAB hab;
PFNWP FrameWndProc, OldpFrame;
HWND hwndFrame;


OldpFrame = WinSubclassWindow(hwndFrame,
                                    (PFNWP)FrameWndProc);


MRESULT EXPENTRY FrameWndProc(hwnd, msg, mp1, mp2)
{
         .
         .
         .
   switch(msg) {

     case . . . :
         .
         .
         .

     default:
        OldpFrame(hwnd, msg, mp1, mp2);
  }

}
```

WinSubstituteStrings

This function performs a substitution process on a text string, replacing specific marker characters with text supplied by the application.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
LONG WinSubstituteStrings (HWND hwnd, PSZ pszSrc, LONG lDestMax,
                           PSZ pszDest)
```

Parameters

hwnd (HWND) – input
Handle of window that processes the call.

pszSrc (PSZ) – input
Source string.

This is the text string that is to have substitution performed.

lDestMax (LONG) – input
Maximum number of characters returnable.

This is the maximum number of characters that can be returned in *pszDest*. It must be greater than 0.

pszDest (PSZ) – output
Resultant string.

This is the text string produced by the substitution process.

The string is truncated if it would otherwise contain more than *lDestMax* characters. When truncation occurs, the last character of the truncated string is always the null-termination character.

Returns

lDestRet (LONG) – returns
Actual number of characters returned.

This is the actual number returned in *pszDest*, excluding the null-termination character. The maximum value is (*lDestMax*–1). It is zero if an error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

When a string of the form "%n" (where n is in the range 0 through 9) occurs in the source string, a WM_SUBSTITUTESTRING message is sent to the specified window. This message returns a text string to use as a substitution for "%n" in the destination string, which is otherwise an exact copy of the source string.

If "%%" occurs in the source, "%" is copied to the destination, but no other substitution occurs. If "%x" occurs in the source, where x is not a digit or "%," the source is copied unchanged to the destination. The source and destination strings must not overlap in memory.

This function is particularly useful for displaying variable information in dialogs, menus, and other user-interface calls. Variable information can include such things as file names, which cannot be statically declared within resource files.

This function is called by the system while creating child windows in a dialog box. It allows the child windows to perform textual substitutions in their window text.

Related Functions

- WinCompareStrings
- WinLoadString
- WinNextChar
- WinPrevChar
- WinUpper
- WinUpperChar

Related Messages

- WM_SUBSTITUTESTRING

Example Code

This example shows how the substitution process works when the WinSubstituteStrings call is made.

```
#define INCL_WINDIALOGS
#include <OS2.H>
static MRESULT ClientWindowProc( HWND hwnd, ULONG  msg, MPARAM mp1, MPARAM mp2 );
test()
{
HWND hwnd;
char source[] = " this is the source string: %1 ";
char result[22];
MPARAM mp1;
ULONG  msg;

 /*
 This function performs a substitution process on a text string,
 replacing specific marker characters with text supplied by the
 application.
 */

          WinSubstituteStrings(hwnd,
                               source,
                               sizeof(source),
                               result);

 /* WM_SUBSTITUTESTRING message is sent to the window defined by */
 /* hwnd. */

}
static MRESULT ClientWindowProc( HWND hwnd, ULONG  msg, MPARAM mp1, MPARAM mp2 )
{
switch(msg)
{
 case WM_SUBSTITUTESTRING:

    switch( (ULONG)mp1)
    {
      case 1:
        return(MRFROMP("A"));
      break;
    }

 break;
}
}
```

WinSubtractRect

This function subtracts one rectangle from another.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinSubtractRect (HAB hab, PRECTL prclDest, PRECTL prclSrc1,
                      PRECTL prclSrc2)
```

Parameters

hab (HAB) – input
Anchor-block handle.

prclDest (PRECTL) – output
Result.

The result of the subtraction of *prclSrc2* from *prclSrc1*.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT may also be used, if supported by the language.

prclSrc1 (PRECTL) – input
First source rectangle.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT may also be used, if supported by the language.

prclSrc2 (PRECTL) – input
Second source rectangle.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT may also be used, if supported by the language.

Returns

rc (BOOL) – returns
Not-empty indicator.

TRUE	Rectangle is not empty
FALSE	Rectangle is empty or an error occurred.

Remarks

Subtracts *prclSrc2* from *prclSrc1*.

prclSrc1, *prclSrc2*, and *prclDest* must be distinct RECTL structures.

Subtracting one rectangle from another does not always result in a rectangular area. When this occurs, this function returns *prclSrc1* in *prclDest*. For this reason, this function provides only an approximation of subtraction. However, the area described by *prclDest* is always greater than, or equal to, the true result of the subtraction.

GpiCombineRegion can be used to calculate the true result of the subtraction of two rectangular areas. The WinSubtractRect function is much faster.

Related Functions

- WinCopyRect
- WinEqualRect
- WinFillRect
- WinInflateRect
- WinIntersectRect
- WinIsRectEmpty
- WinOffsetRect
- WinPtInRect
- WinSetRect
- WinSetRectEmpty
- WinUnionRect

Example Code

This example uses the WinSubstractRect call to subtract two rectangles.

```
#define INCL_WINRECTANGLES
#include <OS2.H>
HAB hab;
RECTL resultrcl;   /* result. */
RECTL rclminuend={25, /* x coordinate of left-hand edge of */
                     /* rectangle. */
                  25, /* y coordinate of bottom edge of    */
                     /* rectangle. */
                  425,/* x coordinate of right-hand edge of */
                     /* rectangle. */
                  425};/* y coordinate of top edge of rectangle. */

RECTL rclsubtrahend={15, /* x coordinate of left-hand edge of */
                     /* rectangle. */
                  15, /* y coordinate of bottom edge of    */
                     /* rectangle. */
                  125,/* x coordinate of right-hand edge of */
                     /* rectangle. */
                  125};/* y coordinate of top edge of rectangle.*/


WinSubtractRect(hab,
                &resultrcl,
                &rclminuend,
                &rclsubtrahend);
```

WinSwitchToProgram

This function makes a specific program the active program.

Syntax

```
#define INCL_WINSWITCHLIST /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinSwitchToProgram  (HSWITCH hswitchSwHandle)
```

Parameters

hswitchSwHandle (HSWITCH) – input
Window List entry handle of program to be activated.

Returns

rc (ULONG) – returns
Return code.

0	Successful completion.
INV_SWITCH_LIST_ENTRY_HANDLE	Invalid Window List entry handle of the program to be activated.
NOT_PERMITTED_TO_CAUSE_SWITCH	Requesting program is not the current foreground process.

Possible returns from WinGetLastError

PMERR_INVALID_SWITCH_HANDLE (0x1202)	An invalid Window List entry handle was specified.

Remarks

Use of this function causes another window (and its related windows) of a PM session to appear on the front of the screen, or a switch to another session in the case of a non-PM program. In either case, the keyboard (and mouse for the non-PM case) input is directed to the new program.

A program can only be made the foreground process by the application which is the current foreground process. This function is ignored if the issuer is not the current foreground process.

A foreground process is defined as being any process within the active non-PM session, or the window with the input focus for a PM session.

Related Functions

- WinAddSwitchEntry
- WinChangeSwitchEntry
- WinCreateSwitchEntry
- WinQuerySessionTitle
- WinQuerySwitchEntry
- WinQuerySwitchHandle
- WinQuerySwitchList
- WinQueryTaskSizePos
- WinQueryTaskTitle
- WinRemoveSwitchEntry

Example Code

This example calls WinSwitchToProgram to make a window the foreground process.

```
#define INCL_WINSWITCHLIST
#include <OS2.H>

HAB     hab;
HWND    hwndFrame;
HSWITCH hswitch;

hswitch = WinQuerySwitchHandle(hwndFrame, 0);

WinSwitchToProgram(hswitch);   /* will switch to window defined */
                               /* by hwndFrame.                 */
```

WinTerminate

This function terminates an application thread's use of the Presentation Manager and releases all of its associated resources.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
BOOL WinTerminate (HAB hab)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

rc (BOOL) – returns
Termination indicator.

TRUE	Application usage of Presentation Manager successfully terminated
FALSE	Application usage of Presentation Manager not successfully terminated, or WinInitialize has not been issued on this thread.

Remarks

It is good practice to issue this function before terminating an application thread. Before issuing this function, the application must destroy all windows and message queues that have been created by the thread, and return any cached presentation spaces to the cache. If it does not do so, the results, and the return value from this and subsequent calls are indeterminate.

Related Functions

- WinCancelShutdown
- WinCreateMsgQueue
- WinInitialize

Example Code

This example calls WinTerminate in a typical termination sequence.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HAB hab;
HWND hwndFrame;
QMSG qmsg;
HMQ hmq;

      while( WinGetMsg( hab, &qmsg, NULL, 0, 0 ) )
         WinDispatchMsg( hab,              /* PM anchor block handle        */
                         &qmsg );          /* pointer to message            */


   /* Destroy the standard windows if they were created.                     */

   if ( hwndFrame != NULL )
      WinDestroyWindow( hwndFrame );       /* frame window handle           */

   /* Destroy the message queue and release the anchor block.                */

   if ( hmq != NULL )
      WinDestroyMsgQueue( hmq );

   if ( hab != NULL )
      WinTerminate( hab );
```

WinTerminateApp

This function terminates an application previously started with the WinStartApp function.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinTerminateApp  (HAPP  happ)
```

Parameters

happ (HAPP) – input
Anchor-block handle.

Identifies the application to terminate.

Returns

rc (BOOL) – returns
Termination indicator.

TRUE	Application successfully terminated
NULL	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HAPP (0x1533)	The application handle passed to WinTerminateApp does not correspond to a valid session.
PMERR_CANNOT_STOP (0x1534)	The session cannot be stopped.

Remarks

The application to terminate must have been started using the WinStartApp function with the SAF_STARTCHILDAPP option specified.

If the specified application does not stop, this function returns TRUE. To ensure that the application has terminated, the application calling WinTerminateApp must wait for the appropriate message to be posted to the window specified in the WinStartApp function.

The WinTerminateApp function must be called from the same process as the WinStartApp function.

This function requires the existence of a message queue.

Related Functions

- WinStartApp

Example Code

This example calls WinTerminate in a typical termination sequence.

```
#define INCL_DOSSESMGR
#include <os2.h>

HWND        hwndNotify;
PPROGDETAILS pDetails;
HAPP        happ;

 pDetails->Length          = sizeof(PROGDETAILS);
 pDetails->progt.progc     = PROG_WINDOWABLEVIO;
 pDetails->progt.fbVisible = SHE_VISIBLE;
 pDetails->pszTitle        = "TEXT";
 pDetails->pszExecutable   = "TEXT.EXE";
 pDetails->pszParameters   = NULL;
 pDetails->pszStartupDir   = "";
 pDetails->pszICON         = "T.ICO";
 pDetails->pszEnvironment  = "WORKPLACE\0\0";
 pDetails->swpInitial.fl   = SWP_ACTIVATE; /* window positioning */
 pDetails->swpInitial.cy   = 0;    /* width of window */
 pDetails->swpInitial.cx   = 0;    /* height of window */
 pDetails->swpInitial.y    = 0;    /* lower edge of window */
 pDetails->swpInitial.x    = 0;    /* left edge of window */
 pDetails->swpInitial.hwndInsertBehind = HWND_TOP;
 pDetails->swpInitial.hwnd             = hwndNotify;
 pDetails->swpInitial.ulReserved1      = 0;
 pDetails->swpInitial.ulReserved2      = 0;

 happ = WinStartApp( hwndNotify,pDetails,NULL,NULL,SAF_STARTCHILDAPP);
       .
       .
       .
 WinTerminateApp(happ);
```

WinTrackRect

This function draws a tracking rectangle.

Syntax

```
#define INCL_WINTRACKRECT /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinTrackRect (HWND hwnd, HPS hps, PTRACKINFO ptiTrackinfo)
```

Parameters

hwnd (HWND) – input
Window handle where tracking is to take place.

It is assumed that the style of this window is not WS_CLIPCHILDREN.

HWND_DESKTOP	Track over the entire screen
Other	Track over specified window only.

hps (HPS) – input
Presentation-space handle.

Used for drawing the clipping rectangle:

NULLHANDLE	The *hwnd* parameter is used to calculate a presentation space for tracking. It is assumed that tracking takes place within *hwnd* and that the style of this window is not WS_CLIPCHILDREN. Thus, when the drag rectangle appears, it is not clipped by any children within the window. If the window style is WS_CLIPCHILDREN and the application causes the drag rectangle to be clipped, it must explicitly pass an appropriate presentation space.
Other	Specified presentation-space handle.

ptiTrackinfo (PTRACKINFO) – in/out
Track information.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Tracking successful.
FALSE	Tracking canceled, or the pointing device was already captured when this function was called.

Only one tracking rectangle can be in use at one time.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

The WinTrackRect call provides general-purpose pointing-device tracking. It draws a rectangle and enables the user to position the entire rectangle, or size a specific side or corner, as required. The resulting rectangle is then returned to the application, which can use this new information for size and position data. The window manager interface for moving and sizing windows by means of the wide sizing borders uses this function, for example.

This function enables the caller to control such limiting values as:

- A maximum and minimum tracking size
- Absolute tracking-position limits
- The tracking rectangle side widths
- A restriction of tracking rectangle movements to a predefined positional grid.

It automatically calls WinLockWindowUpdate to prevent output in the window *hwnd* and its descendants while tracking. When tracking has been completed, output is enabled before this function returns. It also determines which button of the pointing device is depressed at the start of the operation, and only completes the tracking operation when the same button is released.

If the *fs* parameter of the TRACKINFO structure specified by the TF_SETPOINTERPOS value is included, the pointing device pointer is positioned at the center of the tracking rectangle. Otherwise, the pointing device pointer is not moved from its current position and *delta* is established between the pointing device position and the part of the tracking rectangle that it moves (the *delta* is kept constant).

While moving or sizing with the keyboard interface, the pointing device pointer is repositioned with the tracking rectangle's new size or position.

While tracking, these keys are active:

Enter	Accepts the new position or size.
Left cursor	Moves the pointing device pointer and tracking rectangle left. If the pointing device pointer is on the upper or lower edge of the tracking rectangle, the pointer is moved to the top-left or bottom-left corner respectively.
Up cursor	Moves the pointing device pointer and tracking rectangle up. If the pointing device pointer is on the left or right edge of the tracking rectangle, the pointer is moved to the top-left or top-right corner respectively.

Right cursor	Moves the pointing device pointer and tracking rectangle right. If the pointing device pointer is on the upper or lower edge of the tracking rectangle, the pointer is moved to the top-right or bottom-right corner respectively.
Down cursor	Moves the pointing device pointer and tracking rectangle down. If the pointing device pointer is on the left or right edge of the tracking rectangle, the pointer is moved to the bottom-left or bottom-right corner respectively.
Esc	Cancels the current tracking operation. The value of the tracking rectangle is undefined on exit.

The pointing device and the keyboard interface can be intermixed. The caller need not include the TF_SETPOINTERPOS value to use the keyboard interface, as this value simply initializes the position of the pointing device pointer.

If TF_GRID is specified in the TRACKINFO structure, the interior of the tracking rectangle is restricted to multiples of the values of the *cxGrid* and *cyGrid* parameters. The default values for these are the system font character width and half the system font character height, respectively.

Tracking movements using the keyboard arrow keys depend on whether or not TF_GRID is specified in the TRACKINFO structure, If not specified, the increments are the values of *cxKeyboard* and *cyKeyboard*. If specified the increments are the largest multiples of *cxGrid* and *cyGrid* that do not exceed *cxKeyboard* and *cyKeyboard*, respectively. If *cxGrid* exceeds *cxKeyboard*, or *cyGrid* exceeds *cyKeyboard*, the keyboard arrow keys do not cause tracking.

The tracking rectangle is usually logically "on top" of objects it tracks, so that the user can see the old size and position while tracking the new. Thus, it is possible for a window "below" the tracking rectangle to be updated while part of the tracking rectangle is "above" it.

Because the tracking rectangle is drawn in exclusive-OR mode, no window can draw below the tracking rectangle (and thereby obliterate it) without first notifying the tracking code, because unwanted areas of the tracking rectangle can be left behind. If the window doing the drawing is clipped out from the window in which the tracking is occurring, this problem does not arise.

To prevent a window that is currently processing a WM_PAINT message drawing over the tracking rectangle, the tracking rectangle is considered as a system-wide resource, only one of which can be in use at any time. If there is a risk of the currently-updating window drawing on the tracking rectangle, the tracking rectangle is removed while that window and its child windows update, and it is then replaced. This is done during the WinBeginPaint and WinEndPaint functions. If the tracking rectangle overlaps, it is removed in the WinBeginPaint function. In the WinEndPaint function, all the child windows are updated by means of the WinUpdateWindow function before the tracking rectangle is redrawn.

WinTrackRect has a modal loop within it. The loop has a HK_MSGFILTER hook and a MSGF_TRACK hook code.

Note: The rectangle tracked by this function stays within the specified tracking bounds and dimensions. If the rectangle passed is out of these bounds, or it is too large or too small, it is modified to a rectangle that meets these limits.

Related Functions

- WinShowTrackRect

Related Messages

- WM_PAINT

Example Code

This example shows how WinTrackRect can be used to allow a user size a rectangle on the screen.

```
#define INCL_WINTRACKRECT

#include <os2.h>

BOOL MyTrackRoutine(HAB hab, HPS hps, PRECTL rcl)
{
  TRACKINFO track;

  track.cxBorder = 4;
  track.cyBorder = 4;  /* 4 pel wide lines used for rectangle */
  track.cxGrid = 1;
  track.cyGrid = 1;    /* smooth tracking with mouse */
  track.cxKeyboard = 8;
  track.cyKeyboard = 8; /* faster tracking using cursor keys */

  WinCopyRect(hab, &track.rclTrack, rcl);   /* starting point */

  WinSetRect(hab, &track.rclBoundary, 0, 0, 640, 480); /* bounding rectangle */

  track.ptlMinTrackSize.x = 10;
  track.ptlMinTrackSize.y = 10;  /* set smallest allowed size of rectangle */
  track.ptlMaxTrackSize.x = 200;
  track.ptlMaxTrackSize.y = 200; /* set largest allowed size of rectangle */

  track.fs = TF_MOVE;

  if (WinTrackRect(HWND_DESKTOP, hps, &track) )
  {
    /* if successful copy final position back */
    WinCopyRect(hab, rcl, &track.rclTrack);
    return(TRUE);
  }
  else
  {
    return(FALSE);
  }
}
```

WinTranslateAccel

This function translates a WM_CHAR message.

Syntax

```
#define INCL_WINACCELERATORS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinTranslateAccel  (HAB hab, HWND hwnd, HACCEL haccelAccel,
                         PQMSG pQmsg)
```

Parameters

hab (HAB) – input
Anchor-block handle.

hwnd (HWND) – input
Destination window.

haccelAccel (HACCEL) – input
Accelerator-table handle.

pQmsg (PQMSG) – in/out
Message to be translated.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_HACCEL (0x101A)	An invalid accelerator-table handle was specified.

Remarks

This function translates *pQmsg* if it is a WM_CHAR message in the accelerator table *haccelAccel*. The message is translated into a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message, with *hwnd* identifying the destination window. Normally, this parameter is a frame-window handle. This function does not highlight menu items.

If *haccelAccel* equals NULL, the current accelerator table is assumed.

WinTranslateAccel returns TRUE if the message matches an accelerator in the table. *pQmsg* is modified by WinTranslateAccel if a match is found.

If a menu item exists that matches the accelerator-command value, and that item is disabled, *pQmsg* is translated to a WM_NULL message, rather than a WM_COMMAND, WM_SYSCOMMAND, or WM_HELP message. If the command is WM_SYSCOMMAND or WM_HELP (and if a WM_SYSCOMMAND or FID_SYSMENU child window is searched) the menu child window of *hwnd* that has the FID_MENU identifier is searched.

It is possible to have accelerators that do not correspond to items in a menu. If the command value does not match any items in the menu, the message is still translated.

Generally, applications do not have to call this function; it is usually called automatically by WinGetMsg and WinPeekMsg, when a WM_CHAR message is received with the window handle of the active window as the first parameter. The standard frame window procedure always passes WM_COMMAND messages to the FID_CLIENT window. Because the message is physically changed by WinTranslateAccel, applications do not see the WM_CHAR messages that result in WM_COMMAND, WM_SYSCOMMAND, or WM_HELP messages.

Related Functions

- WinCopyAccelTable
- WinCreateAccelTable
- WinDestroyAccelTable
- WinLoadAccelTable
- WinQueryAccelTable
- WinSetAccelTable

Related Messages

- WM_CHAR
- WM_COMMAND
- WM_HELP
- WM_NULL
- WM_SYSCOMMAND

Example Code

This example uses the WinTranslateAccel API to translate WM_CHAR messages destined for the frame window.

```
#define INCL_WINWINDOWMGR
#define INCL_WINACCELERATORS
#include <OS2.H>

HACCEL haccel;
HWND hwndFrame, hwndClient; /* window handles. */
HAB hab;                    /* anchor block.   */
QMSG qmsg;

hwndFrame = WinQueryWindow(hwndClient,
                           QW_PARENT); /* get handle of parent, */
                                       /* which is frame window. */


    /* Now  get the accel table for the frame window  */
haccel = WinQueryAccelTable(hab,
                            hwndFrame);

WinTranslateAccel(hab,
                  hwndFrame,
                  haccel,
                  &qmsg);

switch(qmsg.msg)
{
   case WM_COMMAND:

   case WM_SYSCOMMAND:

   case WM_HELP:
   break;
}
```

WinUnionRect

This function calculates a rectangle that bounds the two source rectangles.

Syntax

```
#define INCL_WINRECTANGLES /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinUnionRect (HAB hab, PRECTL prclDest, PRECTL prclSrc1,
                   PRECTL prclSrc2)
```

Parameters

hab (HAB) – input
Anchor-block handle.

prclDest (PRECTL) – output
Bounding rectangle.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT may also be used, if supported by the language.

prclSrc1 (PRECTL) – input
First source rectangle.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT may also be used, if supported by the language.

prclSrc2 (PRECTL) – input
Second source rectangle.

Note: The value of each field in this structure must be in the range –32768 through 32767. The data type WRECT may also be used, if supported by the language.

Returns

rc (BOOL) – returns
Nonempty indicator.

TRUE *prclDest* is a nonempty rectangle
FALSE Error, or *prclDest* is an empty rectangle.

Remarks

prclSrc1 and *prclSrc2* must not be NULL pointers, although the rectangles they point to can be empty (see the WinIsRectEmpty function).

If one of the source rectangles is empty, the other is returned.

Related Functions

- WinCopyRect
- WinEqualRect
- WinFillRect
- WinInflateRect
- WinIntersectRect
- WinIsRectEmpty
- WinOffsetRect
- WinPtInRect
- WinSetRect
- WinSetRectEmpty
- WinSubtractRect

Example Code

This example uses the WinUnionRect call to find a rectangle that bounds two source rectangles.

```
#define INCL_WINRECTANGLES
#include <OS2.H>
HAB hab;
RECTL resultrcl;   /* result. */
RECTL rcla={25,    /* x coordinate of left-hand edge of */
                   /* rectangle. */
           25,     /* y coordinate of bottom edge of    */
                   /* rectangle. */
           125,    /* x coordinate of right-hand edge of */
                   /* rectangle. */
           125};   /* y coordinate of top edge of rectangle. */

RECTL rclb = {15,  /* x coordinate of left-hand edge of */
                   /* rectangle. */
              15,  /* y coordinate of bottom edge of    */
                   /* rectangle. */
              125, /* x coordinate of right-hand edge of */
                   /* rectangle. */
              125};

WinUnionRect(hab,
             &resultrcl,
             &rcla,
             &rclb);
```

WinUnlockSystem

This function is specific to version 2.1, or higher, of the OS/2 operating system.

This function causes an application program to attempt to unlock the system.

Syntax

```
#define INCL_WINMESSAGEMGR
#include <os2.h>
BOOL WinUnlockSystem  (HAB hab, PSZ pszPassword)
```

Parameters

hab (HAB) – input
The application anchor block.

pszPassword (PSZ) – input
Password string.

Returns

rc (BOOL) – returns
Return value.

TRUE	The system was successfully unlocked.
FALSE	An error occurred or the system was already in an unlocked state.

Remarks

This function causes an application program to attempt to unlock the system by passing a text string that can be compared against the system lockup password that is stored in an encrypted form by the system.

If the string passed in matches the system lockup password, the lockup dialog is dismissed and the user is able to use his machine. Otherwise, the system remains in the locked up state.

OS/2 does not provide a method for querying *pszPassword* to find out the password. It is up to the developer to ask the user what the password is.

In order to execute WinUnlockSystem after a WinLockupSystem has been called, the WinUnlockSystem must be called from a separate thread because WinLockupSystem will not return until a password has been entered from the keyboard.

The LockupHook hook allows a PM application to customize system lockups.

In order for the window to appear as the top most window on the lockup screen, the WS_CLIPSIBLINGS flag must be used for the *flStyle* parameter in the WinCreateWindow or WinCreateStdWindow call.

Related Functions

- LockupHook
- WinLockupSystem

WinUpdateWindow

This function forces the update of a window and its associated child windows.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinUpdateWindow  (HWND hwnd)
```

Parameters

hwnd (HWND) – input
Window handle.

Returns

rc (BOOL) – returns
Window-updated indicator.

TRUE	Window successfully updated
FALSE	Window not successfully updated.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

If *hwnd* is an asynchronous window (that is, it does not have a style of WS_SYNCPAINT), only it and its asynchronous children are updated. They are sent WM_PAINT messages from this function.

If *hwnd* is a synchronous window, only it and its synchronous children are updated. They are sent WM_PAINT messages from this function. If the window is owned by a different thread from the thread issuing the call, the message is sent asynchronously (using WinPostMsg) and not synchronously (using WinSendMsg).

If *hwnd* is a child of a nonclip-children parent, the update region of *hwnd* is subtracted from the update region of the parent, if the parent has one. This is so that any parent-window drawing after *hwnd* does not draw over whatever is drawn by *hwnd*.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint

- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinValidateRect
- WinValidateRegion

Related Messages

- WM_PAINT

Example Code

This example uses the WinUpdateWindow call to send a WM_PAINT message to a window procedure.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
#define WM_USERDEF WM_USER + 1
main()
{

}
static MRESULT ClientWindowProc( HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2 )
{
 switch(msg)
 {
  case WM_PAINT:
  break;
  case WM_USERDEF:
   WinUpdateWindow(hwnd);
 }
}
```

WinUpper

This function converts a string to uppercase.

Syntax

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinUpper  (HAB hab, ULONG ulCodepage, ULONG ulCountry,
                 PSZ pszString)
```

Parameters

hab (HAB) – input
Anchor-block handle.

ulCodepage (ULONG) – input
Code page.

0	Use the current-process code page
Other	Use the specified code page.

ulCountry (ULONG) – input
Country code.

0	Use the default country code specified in CONFIG.SYS
Other	Use the specified country code.

pszString (PSZ) – in/out
String to be converted to uppercase.

Returns

ulRetLen (ULONG) – returns
Length of converted string.

Possible returns from WinGetLastError

PMERR_INVALID_STRING_PARM (0x100B)	The specified string parameter is invalid.

Remarks

The following is the list of valid code pages:

Country	Code Page
Canadian-French	863
Desktop Publishing	1004
Iceland	861

Latin 1 Multilingual	850
Latin 2 Multilingual	852
Nordic	865
Portuguese	860
Turkey	857
U.S. (IBM PC)	437

Code page 1004 is compatible with Microsoft Windows.

The following EBCDIC code pages, based on character set 697, are also available for output:

Country	**Code Page**
Austrian/German	273
Belgian	500
Brazil	037
Czechoslovakia	870
Danish/Norwegian	277
Finnish/Swedish	278
French	297
Hungary	870
Iceland	871
International	500
Italian	280
Poland	870
Portuguese	037
Spanish	284
Turkey	1026
U.K.-English	285
U.S.-English	037
Yugoslavia	870

Code pages 274 (Belgian) and 282 (Portuguese) can be used to provide access to old data. The following is the list of valid country codes:

Country	**Code**
Arabic	785
Australian	61
Belgian	32
Canadian-French	2
Danish	45
Finnish	358
French	33
German	49
Hebrew	972
Italian	39
Japanese	81
Korean	82
Latin-American	3
Netherlands	31
Norwegian	47

Portuguese	351
Simpl. Chinese	86
Spanish	34
Swedish	46
Swiss	41
Trad. Chinese	88
UK-English	44
US-English	1
Other country	0

Related Functions

- WinCompareStrings
- WinLoadString
- WinNextChar
- WinPrevChar
- WinSubstituteStrings
- WinUpperChar

Example Code

This example shows how the WinUpper call can be used to convert a strings in NLS languages to uppercase.

```
#define INCL_WINCOUNTRY
#include <OS2.H>
#include <stdio.h>
main()
{
HAB hab;
char szString[] = "hablas español?";
hab = WinInitialize(0);
WinUpper(hab,
         850,
         34,
         szString);
WinTerminate(hab);
   }
```

WinUpperChar

This function translates a character to uppercase.

Syntax

```
#define INCL_WINCOUNTRY /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG WinUpperChar (HAB hab, ULONG ulCodepage, ULONG ulCountry,
                    ULONG ullnchar)
```

Parameters

hab (HAB) – input
Anchor-block handle.

ulCodepage (ULONG) – input
Code page.

0	Use the current-process code page
Other	Use the specified code page.

ulCountry (ULONG) – input
Country code.

0	Use the default country code specified in CONFIG.SYS
Other	Use the specified country code.

ullnchar (ULONG) – input
Character to be translated to uppercase.

Returns

ulOutchar (ULONG) – returns
Translated character.

0	Error occurred
Other	The translated character.

Possible returns from WinGetLastError

PMERR_INVALID_STRING_PARM (0x100B)	The specified string parameter is invalid.

Remarks

The case-mapping used is the same as provided by the OS/2 DosCaseMap call.

Related Functions

- WinCompareStrings
- WinLoadString
- WinNextChar
- WinPrevChar
- WinSubstituteStrings
- WinUpper

Example Code

This example shows how the WinUpperChar call can be used to convert a characters in NLS languages to uppercase.

```
#define INCL_WINCOUNTRY
#include <OS2.H>
#include <stdio.h>
main()
{
HAB hab;
char szString[] = "Ê";
hab = WinInitialize(0);
WinUpper(hab,
         850,
         49,
         szString);
WinTerminate(hab);
}
```

WinValidateRect

This function subtracts a rectangle from the update region of an asynchronous paint window, marking that part of the window as visually valid.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinValidateRect  (HWND hwnd, PRECTL prclRect,
                       BOOL fIncludeClippedChildren)
```

Parameters

hwnd (HWND) – input
Handle of window whose update region is changed.

If this parameter is HWND_DESKTOP or a desktop-window handle, the function applies to the whole screen (or desktop).

prclRect (PRECTL) – input
Rectangle to be subtracted from the window's update region.

Note: The value of each field in this structure must be in the range -32768 through 32767. The data type WRECT may also be used, if supported by the language.

fIncludeClippedChildren (BOOL) – input
Validation-scope indicator.

TRUE	Include descendants of *hwnd* in the valid rectangle
FALSE	Include descendants of *hwnd* in the valid rectangle, only if parent is not WS_CLIPCHILDREN.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

The call is not used for CS_SYNCPAINT windows.

This function has no effect on the window if any part of the window has been made invalid since the last call to WinBeginPaint, WinQueryUpdateRect, or WinQueryUpdateRegion.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRegion

Example Code

The window needs painting. This is done asynchronously on the drawing thread. The window update region is copied into a local region and passed to the drawing thread. The window must be validated now (to prevent further unnecessary paint messages).

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HRGN hrgnUpdate;
HPS hps;
HWND hwnd;
/* Window needs paint           */
        case WM_PAINT:

/* assume we stop any asynchronous drawing.  */
/* by posting a message to the asynchronous  */
/* drawing thread.                           */

hrgnUpdate=(HRGN)GpiCreateRegion(hps,    /* Create empty region  */
                                 0L,
                                 (PRECTL)NULL);

WinQueryUpdateRegion(hwnd,              /* Save the window update */
                     hrgnUpdate);       /* region.                */

WinValidateRect(hwnd,                   /* Validate window now to */
                (PRECTL)NULL,           /*   stop more paint msgs */
                  TRUE);

/* assume a message is posted to the drawing thread, passing */
/* the update region: (MPARAM)hgrnUpdate.                    */

   mr = (MRESULT) 0L;              /*  Message processed          */
   break;                          /*       End window painting   */
```

WinValidateRegion

This function subtracts a region from the update region of an asynchronous paint window, marking that part of the window as visually valid.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinValidateRegion  (HWND hwnd, HRGN hrgn,
                         BOOL fIncludeClippedChildren)
```

Parameters

hwnd (HWND) – input
Handle of window whose update region is changed.

If this parameter is HWND_DESKTOP or a desktop window handle, the function applies to the whole screen (or desktop).

hrgn (HRGN) – input
Handle of subtracted region.

This is the region that is subtracted from the window's update region.

fIncludeClippedChildren (BOOL) – input
Validation-scope indicator.

TRUE	Include descendants of *hwnd* in the valid region
FALSE	Include descendants of *hwnd* in the valid region, only if parent is not WS_CLIPCHILDREN.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.
PMERR_HRGN_BUSY (0x2034)	An internal region busy error was detected. The region was locked by one thread during an attempt to access it from another thread.

PMERR_INVALID_FLAG (0x1019)	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

The call is not used for CS_SYNCPAINT windows.

The call has no effect on the window if any part of the window has been made invalid since the last call to WinBeginPaint, WinQueryUpdateRect, or WinQueryUpdateRegion.

Related Functions

- WinBeginPaint
- WinEnableWindowUpdate
- WinEndPaint
- WinExcludeUpdateRegion
- WinGetClipPS
- WinGetPS
- WinGetScreenPS
- WinInvalidateRect
- WinInvalidateRegion
- WinIsWindowShowing
- WinIsWindowVisible
- WinLockVisRegions
- WinOpenWindowDC
- WinQueryUpdateRect
- WinQueryUpdateRegion
- WinRealizePalette
- WinReleasePS
- WinShowWindow
- WinUpdateWindow
- WinValidateRect

Example Code

This example shows how an application can incrementally repaint an asynchronous-paint window one area at a time. While a window is invalid (has a non-null update region), WM_PAINT messages are returned by WinGetMsg. The application uses WinQueryUpdateRegion to obtain a region that requires repainting, and WinValidateRegion to validate the region (reset the update region to null).

```
#define INCL_WINWINDOWMGR
#define INCL_GPIREGIONS
#include <OS2.H>
HRGN hrgnUpdt, sRgnType;
HPS hpsPaint;
HWND hwnd;
/* Window needs paint           */
        case WM_PAINT:
           .

/* assume we stop any asynchronous drawing.  */
/* by posting a message to the asynchronous  */
/* drawing thread.                           */

hrgnUpdt = (HRGN)GpiCreateRegion(hpsPaint,
                           (ULONG)0,
                           (PRECTL)NULL);

sRgnType = (HRGN)WinQueryUpdateRegion(hwnd,
                                 hrgnUpdt);

/* if the region is not null and the call is not in error, */
/* validate the region.                                    */

if ((sRgnType != NULL) &&
   (sRgnType != RGN_ERROR)) {
   WinValidateRegion(hwnd, hrgnUpdt, FALSE);
/*
here we would send the update region handle to an
asynchronous drawing thread. We have already validated the
region, so no more WM_PAINT messages will be sent due to this
region.
*/
      } else { GpiDestroyRegion(hpsPaint, hrgnUpdt);}
```

WinWaitEventSem

WinWaitEventSem waits for an event semaphore to be posted or for a Presentation Manager message.

Syntax

```
#define INCL_WINMESSAGEMGR
#include <os2.h>
APIRET WinWaitEventSem  (HEV hev, ULONG ulTimeout)
```

Parameters

hev (HEV) – input
The handle of the event semaphore to wait for.

ulTimeout (ULONG) – input
Time-out in milliseconds.

This is the maximum amount of time the user wants to allow the thread to be blocked.

This parameter can also have the following values:

0	(SEM_IMMEDIATE_RETURN) WinWaitEventSem returns without blocking the calling thread.
-1	(SEM_INDEFINITE_WAIT) WinWaitEventSem blocks the calling thread indefinitely.

Returns

rc (APIRET) – returns
Return Code.

WinWaitEventSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
95	ERROR_INTERRUPT
640	ERROR_TIMEOUT

Remarks

WinWaitEventSem is similar to DosWaitEventSem and enables a thread to wait for an event semaphore to be posted or for a window message sent by the WinSendMsg function from another thread to be received.

This function can be called by any thread in the process that created the semaphore. Threads in other processes can also call this function, but they must first gain access to the semaphore by calling DosOpenEventSem.

Since the processing of a window message may take longer than the value specified by the *ulTimeout* parameter, this function may not return within the time specified by that value.

Related Functions

- WinPostMsg
- WinSendMsg

Example Code

This example causes the calling thread to wait until the specified event semaphore is posted. Assume that the handle of the semaphore has been placed into *hev* already.

ulTimeout is the number of milliseconds that the calling thread will wait for the event semaphore to be posted. If the specified event semaphore is not posted during this time interval, the request times out.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#define INCL_WINMESSAGEMGR
#include <os2.h>
#include <stdio.h>

#ifndef ERROR_TIMEOUT
    #define  ERROR_TIMEOUT    640
    #define  ERROR_INTERRUPT   95
#endif

HEV     hev;        /* Event semaphore handle */
ULONG   ulTimeout;  /* Number of milliseconds to wait */
ULONG   rc;         /* Return code */

   ulTimeout = 60000;  /* Wait for a maximum of 1 minute */

   rc = WinWaitEventSem(hev, ulTimeout);

   if (rc == ERROR_TIMEOUT)
     {
       printf("WinWaitEventSem call timed out");
       return;
     }

   if (rc == ERROR_INTERRUPT)
     {
       printf("WinWaitEventSem call was interrupted");
       return;
     }

   if (rc != 0)
     {
       printf("WinWaitEventSem error: return code = %ld", rc);
       return;
     }
```

WinWaitMsg

This function waits for a filtered message.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WinWaitMsg (HAB hab, ULONG ulFirst, ULONG ulLast)
```

Parameters

hab (HAB) – input
Anchor-block handle.

ulFirst (ULONG) – input
First message identity.

ulLast (ULONG) – input
Last message identity.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This function causes the current thread to wait for a message to arrive on the message queue associated with *hab*. This must be the next message since the queue was last inspected by a *rc* return from WinGetMsg or WinPeekMsg. It must also conform to the filtering criteria specified by *ulFirst* and *ulLast*.

For details of the filtering performed by *ulFirst* and *ulLast*, see the WinGetMsg function.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg

- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode

Example Code

In this example the pointer is kept hidden until mouse activity is detected. The WinWaitMsg call is used to wait for any mouse message.

```
#define INCL_WINWINDOWMGR
#define INCL_WINPOINTERS
#define INCL_WINDESKTOP
#define INCL_WININPUT
#include <OS2.H>
HWND hwnd;
HPOINTER hpointer;
HAB hab;

hpointer = WinQueryPointer(HWND_DESKTOP); /* get the pointer */
                                          /* handle.       */

WinShowPointer(hwnd,FALSE);   /* hide the mouse.             */

WinWaitMsg(hab,
           WM_MOUSEFIRST,     /* all the mouse messages from */
          WM_BUTTON3DBLCLK);  /* WM_MOUSEFIRST to            */
                              /* WM_BUTTON3DBLCLK inclusive. */

WinShowPointer(hwnd,TRUE);    /* If there has been any mouse */
                              /* activity, show the mouse.   */
```

WinWaitMuxWaitSem

WinWaitMuxWaitSem waits for a muxwait semaphore to clear or for a Presentation Manager message.

Syntax

```
#define INCL_WINMESSAGEMGR
#include <os2.h>
APIRET WinWaitMuxWaitSem  (HMUX hmux, ULONG ulTimeout,
                           PULONG pulUser)
```

Parameters

hmux (HMUX) – input
The handle of the muxwait semaphore to wait for.

ulTimeout (ULONG) – input
Time-out in milliseconds.

This is the maximum amount of time the user wants to allow the thread to be blocked.

This parameter can also have the following values:

SEM_IMMEDIATE_RETURN (0)	WinWaitMuxWaitSem returns without blocking the calling thread.
SEM_INDEFINITE_WAIT (minus.1)	WinWaitMuxWaitSem blocks the calling thread indefinitely.

pulUser (PULONG) – output
Pointer to receive the user field.

A pointer to receive the user field (from the muxwait semaphore data structure) of the semaphore that was posted or released.

If DCMW_WAIT_ANY was specified in the *flAttr* parameter when the muxwait semaphore was created, this will be the user field of the semaphore that was posted or released. If the muxwait semaphore consists of mutex semaphores, any mutex semaphore that is released is owned by the caller.

If DCMW_WAIT_ALL was specified in the *flAttr* parameter when the muxwait semaphore was created, this will be the user field of the *last* semaphore that was posted or released. (If the thread did not block, the last semaphore that was posted or released will also be the last semaphore in the muxwait-semaphore list.) If the muxwait semaphore consists of mutex semaphores, all of the mutex semaphores that are released are owned by the caller.

Returns

ulrc (APIRET) – returns
Return Code.

WinWaitMuxWaitSem returns the following values:

0	NO_ERROR
6	ERROR_INVALID_HANDLE
8	ERROR_NOT_ENOUGH_MEMORY
87	ERROR_INVALID_PARAMETER
95	ERROR_INTERRUPT
103	ERROR_TOO_MANY_SEM_REQUESTS
105	ERROR_SEM_OWNER_DIED
286	ERROR_EMPTY_MUXWAIT
287	ERROR_MUTEX_OWNED
292	ERROR_WRONG_TYPE
640	ERROR_TIMEOUT

Remarks

WinWaitMuxWaitSem is similar to DosWaitMuxWaitSem and enables a thread to wait for a muxwait semaphore to clear or for a window message sent by the WinSendMsg function from another thread to be received.

This function can be issued by any thread in the process that created the semaphore. Threads in other processes can also issue this function, but they must first gain access to the semaphore by issuing DosOpenMuxWaitSem.

Since the processing of a window message may take longer than the value specified by the *ulTimeout* parameter, this function may not return within the time specified by that value.

Related Functions

- WinPostMsg
- WinSendMsg

Example Code

This example waits for a muxwait semaphore to clear. Assume that the handle of the semaphore has been placed into *hmux* already. *ulTimeout* is the number of milliseconds that the calling thread will wait for the muxwait semaphore to clear. If the specified muxwait semaphore is not cleared during this time interval, the request times out.

```
#define INCL_DOSSEMAPHORES   /* Semaphore values */
#define INCL_WINMESSAGEMGR
#include <os2.h>
#include <stdio.h>

#ifndef ERROR_TIMEOUT
    #define  ERROR_TIMEOUT    640
    #define  ERROR_INTERRUPT   95
#endif

HMUX     hmux;      /* Muxwait semaphore handle */
ULONG    ulTimeout; /* Number of milliseconds to wait */
ULONG    ulUser;    /* User field for the semaphore that was
                       posted or released (returned) */
ULONG    rc;        /* Return code */

   ulTimeout = 60000;  /* Wait for a maximum of 1 minute */

   rc = WinWaitMuxWaitSem(hmux, ulTimeout, &ulUser);
                       /* On successful return, the ulUser    */
                       /*   variable contains the user        */
                       /*   identifier of the semaphore       */
                       /*   that caused the wait to           */
                       /*   terminate.  If the caller had     */
                       /*   to wait for all the semaphores    */
                       /*   within the muxwait semaphore to   */
                       /*   clear, then the value corresponds */
                       /*   to the last semaphore within the  */
                       /*   muxwait semaphore to clear.  If   */
                       /*   the caller had to wait for any    */
                       /*   semaphore with the muxwait        */
                       /*   semaphore to clear, then the      */
                       /*   value corresponds to that         */
                       /*   semaphore.                        */

   if (rc == ERROR_TIMEOUT)
     {
       printf("WinWaitMuxWaitSem call timed out");
       return;
     }

   if (rc == ERROR_INTERRUPT)
     {
       printf("WinWaitMuxWaitSem call was interrupted");
       return;
     }

   if (rc != 0)
     {
       printf("WinWaitMuxWaitSem error: return code = %ld", rc);
       return;
     }
```

WinWindowFromDC

This function returns the handle of the window corresponding to a particular device context.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinWindowFromDC  (HDC hdc)
```

Parameters

hdc (HDC) – input
Device-context handle.

The device context must first be opened by the WinOpenWindowDC function.

Returns

hwnd (HWND) – returns
Window handle.

NULLHANDLE	Error occurred. For example, the device context has not been opened by the WinOpenWindowDC function.
Other	Window handle.

Possible returns from WinGetLastError

PMERR_INV_HPS (0x207F)	An invalid presentation-space handle was specified.
PMERR_INV_HDC (0x207C)	An invalid device-context handle or (micro presentation space) presentation-space handle was specified.

Related Functions

- WinEnableWindow
- WinIsThreadActive
- WinIsWindow
- WinIsWindowEnabled
- WinQueryDesktopWindow
- WinQueryObjectWindow
- WinQueryWindowDC
- WinQueryWindowProcess

- WinQueryWindowRect
- WinWindowFromID
- WinWindowFromPoint

Example Code

If a device context handle is specified, this example determines which window is associated with that device context.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND hwnd;
HDC hdc;

/* Assume the device context for a window has been opened in     */
/* some other window procedure. We would like to get             */
/* a handle to that window.                                      */

/* This function is called in some other window:            */
/*   hdc = WinOpenWindowDC(hwnd);                             */


hwnd = WinWindowFromDC(hdc);
```

WinWindowFromID

This function returns the handle of the child window with the specified identity.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinWindowFromID  (HWND hwndParent, ULONG id)
```

Parameters

hwndParent (HWND) – input
Parent-window handle.

id (ULONG) – input
Identity of the child window.

It must be greater or equal to 0 and less or equal to 0xFFFF.

Returns

hwnd (HWND) – returns
Window handle.

NULLHANDLE	No child window of the specified identity exists
Other	Child-window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

To obtain the window handle for an item within a dialog box, set *hwndParent* to the dialog-box window's handle, and set *id* to the identity of the item in the dialog template.

Related Functions

- WinEnableWindow
- WinIsThreadActive
- WinIsWindow
- WinIsWindowEnabled
- WinQueryDesktopWindow
- WinQueryObjectWindow
- WinQueryWindowDC
- WinQueryWindowProcess
- WinQueryWindowRect

- WinWindowFromDC
- WinWindowFromPoint

Example Code

This example calls WinWindowFromID to get the window handle of the system menu and calls WinSendMsg to send a message to disable the Close menu item.

```
#define INCL_WINFRAMEMGR   /* for FID_ definitions. */
#define INCL_WINMENUS      /* for MIA_ definitions. */
#define INCL_WINWINDOWMGR
#include <OS2.H>

HWND hwndSysMenu, hwndDlg;

 hwndSysMenu = WinWindowFromID(hwndDlg, FID_SYSMENU);
 WinSendMsg(hwndSysMenu, MM_SETITEMATTR,
     MPFROM2SHORT(SC_CLOSE, TRUE),
     MPFROM2SHORT(MIA_DISABLED, MIA_DISABLED));
```

WinWindowFromPoint

This function finds the window below a specified point, that is a descendant of a specified window.

Syntax

```
#define INCL_WINWINDOWMGR /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
HWND WinWindowFromPoint  (HWND hwndParent, PPOINTL pptlPoint,
                          BOOL fEnumChildren)
```

Parameters

hwndParent (HWND) – input
Window handle whose child windows are to be tested.

HWND_DESKTOP	The desktop-window handle, implying that all main windows are tested. In this instance, *pptlPoint* must be relative to the bottom left corner of the screen.
Other	Parent-window handle.

pptlPoint (PPOINTL) – input
The point to be tested.

Specified in window coordinates relative to the window specified by the *hwndParent* parameter.

fEnumChildren (BOOL) – input
Test control.

TRUE	Test all the descendant windows, including child windows of child windows
FALSE	Test only the immediate child windows.

Returns

hwndFound (HWND) – returns
Window handle beneath *pptlPoint*.

NULLHANDLE	*pptlPoint* is outside *hwndParent*
Parent	*pptlPoint* is not inside any of the children of *hwndParent*
Other	Window handle is beneath *pptlPoint*.

Possible returns from WinGetLastError

PMERR_INVALID_HWND (0x1001)	An invalid window handle was specified.

Remarks

This function checks only the descendants of the specified window.

Related Functions

- WinEnableWindow
- WinIsThreadActive
- WinIsWindow
- WinIsWindowEnabled
- WinQueryDesktopWindow
- WinQueryObjectWindow
- WinQueryWindowDC
- WinQueryWindowProcess
- WinQueryWindowRect
- WinWindowFromDC
- WinWindowFromID

Example Code

This example calls WinWindowFromPoint to find out if any main windows are beneath point 100,100.

```
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND hwndunderneath;
POINTL point = { 100L, 100L};
hwndunderneath = WinWindowFromPoint(HWND_DESKTOP,
                                    &point,
                                    FALSE); /* do not test the */
                                            /* descendents of  */
                                            /* the main        */
                                            /* windows.        */
```

Chapter 8. Functions Supplied by Applications

This chapter describes dialog procedures, window procedures, and hooks. It shows the input parameters and returns that the operating system expects an application to use in application procedures and that can be called by the operating system in response to certain events.

Procedures and hooks are application code that is called by the system in response to certain events.

The names and parameter lists of functions are contained in header files that are incorporated into the application when it is compiled. Their addresses are contained in .LIB files that are incorporated at link time.

The names of procedures and hooks are defined by the application, and their parameter lists are defined by the system. Function prototypes for these procedures and hooks are in PMWIN.H. The prototypes have sample names that can be changed by the programmer before they are inserted into the application source code.

The application passes the address of these procedures and hooks in the following ways:

Dialog procedures	During the WinLoadDlg, WinDlgBox, WinFileDlg, or WinFontDlg function
Window procedures	During the WinRegisterClass or WinSubclassWindow functions
Hooks	During the WinSetHook function
Thunks	During the WinSetClassThunkProc or WinSetWindowThunkProc functions.

The following table shows the procedures and hooks in alphabetic order.

C Name	C Name
Procedures	
DialogProc	WndProc
ThunkProc	
Hooks	
CheckMsgFilterHook	JournalRecordHook
CodePageChangeHook	LoaderHook
DestroyWindowHook	LockupHook
FindWordHook	MsgControlHook
FlushBufHook	MsgFilterHook
HelpHook	MsgInputHook
InputHook	SendMsgHook
JournalPlaybackHook	

Hook Functions

The first section of the chapter describes the hook functions that the application can provide. These procedures are:

- CheckMsgFilterHook
- CodePageChangedHook
- DestroyWindowHook
- FindWordHook
- FlushBufHook
- HelpHook
- InputHook
- JournalPlaybackHook
- JournalRecordHook
- LoaderHook
- LockupHook
- MsgControlHook
- MsgFilterHook
- MsgInputHook
- SendMsgHook

DialogProc

This is a window procedure that automatically subclasses each instance of a dialog box.

Syntax

```
#define INCL_WINDIALOGS /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
MRESULT DialogProc  (HWND hwnd, USHORT usmsg, MPARAM mpParam1,
                     MPARAM mpParam2)
```

Parameters

hwnd (HWND) – input
Handle of the window to which the message applies.

usmsg (USHORT) – input
Message identity.

mpParam1 (MPARAM) – input
Message parameter 1.

mpParam2 (MPARAM) – input
Message parameter 2.

Returns

mresReply (MRESULT) – returns
Message-return data.

Remarks

This procedure is the same as any other window procedure, except that it can receive predefined window messages specific to dialog box windows.

Note: It does *not* receive the WM_CREATE message, but the same information is carried by the WM_INITDLG message, that is generated during the creation of a dialog-box window.

hwnd is always the window handle of the dialog-box window.

The dialog procedure typically processes only some of the messages passed to it. Any messages that it does not process must be passed to WinDefFileDlgProc if the dialog box is the standard file selection dialog, WinDefFontDlgProc if the dialog box is the standard font selection dialog box, or for all other dialog boxes, WinDefDlgProc (not WinDefWindowProc), because these perform the standard dialog-box processing for those messages.

Related Messages

- WM_CREATE
- WM_INITDLG

FindWordHook

This hook allows an application to control where the WinDrawText function breaks a character string that is too long for the drawing rectangle.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL FindWordHook  (USHORT usCodepage, PSZ pszText, ULONG cb,
                    ULONG ich, PULONG pichStart, PULONG pichEnd,
                    PULONG pichNext)
```

Parameters

usCodepage (USHORT) – input
Codepage to use.

This parameter contains the codepage identifier of the string to be formatted.

pszText (PSZ) – input
Text to break.

This parameter contains a pointer to the actual string.

cb (ULONG) – input
Maximum text size.

This parameter contains a value specifying the number of bytes in the string.

ich (ULONG) – input
Break near here.

This parameter contains the index of the character in the string that intersects the right edge of the drawing rectangle.

pichStart (PULONG) – output
Where break began.

This parameter contains the index of the starting character of the intersecting word.

pichEnd (PULONG) – output
Where break ended.

This parameter contains the index of the ending character of the intersecting word.

pichNext (PULONG) – output
Where next word begins.

This parameter contains the index of the starting character of the next word in the string.

Returns

rc (BOOL) – returns
Success indicator.

TRUE If the find-word hook function returns TRUE, WinDrawText will only draw the string up to, but not including, the specified word.

FALSE If the find-word hook function returns FALSE, WinDrawText formats the string in the default manner.

Remarks

The system calls this hook from within the WinDrawText function, if the DT_WORDBREAK flag is set. It lets the application have control of where the function WinDrawText should break for a string that is too long.

FlushBufHook

This function is specific to version 2.1, or higher, of the OS/2 operating system.

This hook allows applications to save data before the system reboots.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL FlushBufHook  (HAB hab)
```

Parameters

hab (HAB) – input
Application anchor block.

Returns

rc (BOOL) – returns
Return code.

TRUE	Successful completion.
FALSE	An error occurred.

Remarks

This hook is called to notify applications that the system is rebooting due to a Ctrl+Alt+Del sequence being entered. It enables applications to write existing information to the hardfile immediately, avoiding loss of data before the system reboots.

Note: Do not use any of the Win or Gpi functions inside the hook routine. At the point in time that the hook routine is executing, these sub-systems may not be fully available because they might be partially shutdown.

Related Functions

- WinSetHook

HelpHook

This hook processes help requests.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL HelpHook  (HAB hab, ULONG usMode, ULONG idTopic,
                ULONG idSubTopic, RECTL prcPosition)
```

Parameters

hab (HAB) – input
Anchor-block handle.

usMode (ULONG) – input
Help mode.

This has one of the following values, indicating the mode from which help has been requested:

HLPM_FRAME	Standard (standard window)
HLPM_WINDOW	Standard (standard window)
HLPM_MENU	Menu mode

idTopic (ULONG) – input
Topic identifier.

- In menu mode this is a pull-down window identity
- In message-box mode this is the message-box identity
- In standard mode this is a window identity.

idSubTopic (ULONG) – input
Subtopic identifier.

- In menu mode this is a command identity
- In message-box mode this is a control identity
- In standard mode this is the identity of the window with the focus (**–1** if none).

prcPosition (RECTL) – input
Rectangle.

This indicates the screen area (in screen coordinates) from where the help was requested. It is provided to enable the help library to avoid covering that area.

- In menu mode it is the bounding rectangle of the selected item, or o the top level menu if value of the *idSubTopic* parameter is **–1**.
- In message-box mode it is the bounding rectangle of the button.

- In standard mode it is the bounding rectangle of the window with the focus, or of the window sent the message if the value of the *idSubTopic* parameter is **-1**.

Note: The data type WRECT can also be used, if supported by the language.

Returns

rc (BOOL) – returns
Indicator as to whether next hook in the chain is called.

The message is always passed to the application.

TRUE The next hook in the chain is not called.
FALSE The next hook in the chain is called.

Remarks

Help-processing is done in two stages. The first stage is the creation of the WM_HELP message. This is done:

- From a WM_CHAR message by ACCELERATOR table translation, when the HELP accelerator option is specified.
- From an action-bar selection, when the MIS_HELP style is specified on the action-bar button.
- From a dialog-box push button, when the BS_HELP style is specified on the push button.
- From a message box, when the MB_HELP style is specified on the message box.

The WM_HELP message is sent to the active window, but will be seen by a modal loop if one is active.

The second stage of processing of help is the processing of the WM_HELP message.

The frame window procedure sees the WM_HELP message because the frame is usually the active window. It processes the WM_HELP message as follows:

- If the window with the focus is the FID_CLIENT frame control, it passes WM_HELP to the FID_CLIENT window.
- If the parent of the window with the focus is the FID_CLIENT frame control, it calls the help hook, specifying:

  usMode = HLPM_WINDOW
  idTopic = frame-window id
  idSubTopic = focus-window id.

- If the parent of the focus window is not the FID_CLIENT frame control (for example, it may be the frame itself, or a second-level dialog control), it calls the hook, specifying:

  usMode = HLPM_WINDOW
  idTopic = focus-window parent id
  idSubTopic = focus-window id.

The message box window procedure sees the WM_HELP message, because it subclasses the frame window. It processes the WM_HELP message by calling the help hook, specifying:

usMode = HLPM_MESSAGE
idTopic = message id
idSubTopic = control id.

The menu window procedure sees the WM_HELP message because it runs a modal loop. It processes the WM_HELP message by calling the help hook, specifying:

usMode = HLPM_MENU
idTopic = menu id of pulldown
idSubTopic = menu id of item.

The WinDefWindowProc function sees the WM_HELP message for a FID_CLIENT window if the client does not handle it itself. It calls the help hook, specifying:

usMode = HLPM_WINDOW
idTopic = active-window id
idSubTopic = focus-window id.

An application sees the WM_HELP message in its dialog procedure. The application can ignore the WM_HELP message, in which case the frame-window procedure action occurs (as described above) or it can simulate a call to the help hook itself, using:

usMode = HLPM_APPLICATION
idTopic = any value
idSubTopic = any value.

The input focus is never given to any of the standard frame controls, so help for these cannot be obtained.

Related Messages

- WM_CHAR
- WM_HELP

InputHook

This hook filters messages from the input queue.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL InputHook (HAB hab, PQMSG pQmsg, ULONG fs)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pQmsg (PQMSG) – input
A PQMSG data structure.

fs (ULONG) – input
Message removal options.

PM_REMOVE	Message is being removed from queue
PM_NOREMOVE	Message is not being removed from queue.

Returns

rc (BOOL) – returns
Processed indicator.

TRUE	The message is not passed on to the next hook in the chain or to the application
FALSE	The message is passed on to the next hook in the chain or to the application.

Remarks

This hook is called when messages are removed from an application queue, before being returned by WinGetMsg or WinPeekMsg. It is called from within these functions just before resuming the application with the message that is returned. There are no restrictions on calls that may be made at this time.

JournalPlaybackHook

This hook plays back recorded messages.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
ULONG JournalPlaybackHook (HAB hab, BOOL fSkip, PQMSG pQmsg)
```

Parameters

hab (HAB) – input
Anchor-block handle.

fSkip (BOOL) – input
Indicator as to whether the next message should be played back.

TRUE The journal playback hook skips to the next message. The *pQmsg* parameter is NULL in this case. The next hook in the chain is not called.

FALSE The journal playback hook returns the next available message. The same message is returned each time, until it is skipped with a call where this parameter is TRUE.

pQmsg (PQMSG) – input
Data structure where the message to be played back is returned.

When this hook is called, the *time* field of the QMSG structure is initialized to the current time. This can be used to determine whether the next message is ready or not. This value must be used for any delta calculations performed by the hook procedure, rather than the result of WinGetCurrentTime.

Returns

ulTime (ULONG) – returns
Waiting time.

The time to wait (in milliseconds) before processing the current message.

Remarks

This hook is called whenever a message is required to be played back.

JournalRecordHook

This hook records user-input messages.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
VOID JournalRecordHook  (HAB hab, PQMSG pQmsg)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pQmsg (PQMSG) – input
Data structure that contains the message to be recorded.

The *hwnd* field of the QMSG structure is also set when the hook is called.

Returns

There is no return value for this hook.

Remarks

This hook is called *after* raw input is translated to WM_CHAR or WM_BUTTON1DBLCLK messages.

The next hook in the chain is always called, and the message is always passed to the application.

JournalPlaybackHook hook does not receive any input played back by this hook. This prevents feedback situations where input is played back a number of times.

Related Messages

- WM_CHAR
- WM_BUTTON1DBLCLK

LoaderHook

This hook allows the library and procedure loading and deleting calls to be intercepted.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL LoaderHook  (HAB hab, LONG idContext, PSZ pszLibname, PHLIB hlib,
                  PSZ pszProcname, PFNWP wndProc, PBOOL pfSuccess)
```

Parameters

hab (HAB) – input
Anchor-block handle.

idContext (LONG) – input
Origin of call to hook.

LHK_DELETEPROC	WinDeleteProcedure
LHK_DELETELIB	WinDeleteLibrary
LHK_LOADPROC	WinLoadProcedure
LHK_LOADLIB	WinLoadLibrary

pszLibname (PSZ) – input
Library name.

This is the same as the library name in the *pszLibname* parameter of the WinLoadLibrary function.

hlib (PHLIB) – in/out
Pointer to a library handle.

This is the same as the library handle in the *hlibLibhandle* parameter of the WinLoadProcedure function or the *hlibLibhandle* parameter of the WinDeleteLibrary function.

If the *idContext* parameter is set to LHK_LOADLIB, then this hook must set the value of this parameter to the handle of the loaded library or to NULLHANDLE if the load fails.

pszProcname (PSZ) – input
Procedure name.

This is the same as the procedure name in the *pszProcname* parameter of the WinLoadProcedure function.

wndProc (PFNWP) – input
Window procedure identifier.

This is the same as the library name in the *pwndproc* parameter of the WinDeleteProcedure function.

If the *idContext* parameter is set to LHK_LOADPROC, then this hook must set the value of this parameter to the handle of the loaded procedure or to NULL if the load fails.

pfSuccess (PBOOL) – in/out
Success indicator.

TRUE	Library or procedure loaded or deleted successfully.
FALSE	Library or procedure not loaded or deleted successfully.

Returns

rc (BOOL) – returns
Processing indicator.

TRUE	Do not call next hook in chain
FALSE	Call next hook in chain.

Remarks

If the hook attempts a load or deletion which is unsuccessful, then the hook must establish the relevant error information.

LockupHook

This function is specific to version 2.1, or higher, of the OS/2 operating system.

This hook is called when the system locks itself up.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
VOID LockupHook (HAB hab, HWND hwndLockupFrame)
```

Parameters

hab (HAB) – input
The application anchor block.

hwndLockupFrame (HWND) – input
The frame window of the lockup panel.

Returns

There is no return value for this hook.

Remarks

The *hwndLockupFrame* parameter contains the frame window handle of the lockup dialog. All HK_LOCKUP hooks registered with the system are called when the system locks itself up. All HK_LOCKUP hooks must be system hooks, not message queue hooks.

Application programs that create other lockup password input windows by hooking the HK_LOCKUP system hook can use WinUnlockSystem to force the system to unlock when another form of input is detected other than mouse or keyboard. For example, a pen gesture or some voice input or signature recognition.

Related Functions

- WinLockupSystem
- WinUnlockSystem

MsgControlHook

This hook allows the call which determine the flow of messages to be intercepted.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL MsgControlHook  (HAB hab, LONG idContext, HWND hwnd,
                      PSZ pszClassName, ULONG usMsgClass,
                      LONG idControl, PBOOL fSuccess)
```

Parameters

hab (HAB) – input
Anchor-block handle.

idContext (LONG) – input
Origin of call to hook.

MCHK_CLASSMSGINTEREST	WinSetClassMsgInterest
MCHK_MSGINTEREST	WinSetMsgInterest
MCHK_MSGMODE	WinSetMsgMode
MCHK_SYNCHRONISATION	WinSetSynchroMode

hwnd (HWND) – input
Window handle.

This is the same as the window handle in the *hwnd* parameter of the WinSetMsgInterest function.

pszClassName (PSZ) – input
Window class name.

This is the same as the window class name in the *pszClassName* parameter of the WinSetClassMsgInterest function.

usMsgClass (ULONG) – input
Message class.

This is the same as the message class in the *ulMsgClass* parameter of the WinSetMsgInterest and the WinSetClassMsgInterest functions.

idControl (LONG) – input
Control setting.

The setting varies with the value of the *idContext* parameter.

For MCHK_CLASSMSGINTEREST, it can be SMI_INTEREST, or SMI_NOINTEREST, or SMI_AUTODISPATCH.

For MCHK_MSGINTEREST, it can be SMI_INTEREST, or SMI_NOINTEREST, or SMI_RESET, or SMI_AUTODISPATCH.

For MCHK_MSGMODE, it can be SMD_DELAYED or SMD_IMMEDIATE.

For MCHK_SYNCHRONISATION, it can be SSM_SYNCHRONOUS, or SSM_ASYNCHRONOUS, or SSM_MIXED.

fSuccess (PBOOL) – in/out
Success indicator.

TRUE	Mode or interest successfully set.
FALSE	Mode or interest not successfully set.

Returns

rc (BOOL) – returns
Processing indicator.

TRUE	Do not call next hook in chain
FALSE	Call next hook in chain.

Remarks

If the hook is unable to alter the message control state, then the hook must establish the relevant error information.

MsgFilterHook

This hook filters messages from inside a mode loop.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL MsgFilterHook  (HAB hab, PQMSG pQmsg, ULONG msgf)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pQmsg (PQMSG) – input
A queue message data structure.

msgf (ULONG) – input
Context in which the hook has been called.

MSGF_DIALOGBOX	Dialog-box mode loop.
MSGF_TRACK	Window-movement and size tracking. When this hook is used the TRACKINFO structure specified the *ptiTrackinfo* parameter of the WinTrackRect function is updated to give the current state before the hook is called. Only the *rclTrack* and the *fs* parameters are updated.
MSGF_DRAG	Direct manipulation mode loop.
MSGF_DDEPOSTMSG	DDE post message mode loop.

Returns

rc (BOOL) – returns
Processed indicator.

TRUE	The message is not passed on to the next hook in the chain or to the application
FALSE	The message is passed on to the next hook in the chain or to the application.

Remarks

This hook is called inside any of the system-mode loops, for instance, during size-tracking or move-tracking, or while a dialog box or menu is displayed.

The WM_QUIT message is passed to this hook, if it occurs during a mode loop.

Related Messages

- WM_QUIT

MsgInputHook

This function is specific to version 2.1, or higher, of the OS/2 operating system.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL MsgInputHook  (HAB hab, PQMSG pQmsg, BOOL fSkip,
                    PBOOL pfNoRecord)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pQmsg (PQMSG) – in/out
Queue message data structure.

A queue message data structure to be filled in with a simulated mouse or keyboard message.

fSkip (BOOL) – input
Skip message flag.

TRUE The hook should advance to the next message.

No message is passed in when *fSkip* is set to TRUE. The *pQmsg* parameter is NULL in this case.

When the hook is called with *fSkip* set to TRUE and there are no further messages to pass in, the MsgInputHook hook must be released using WinReleaseHook.

FALSE The current message should be returned.

As long as the hook procedure is called with *fSkip* set to FALSE, the hook must continue to return the *same* message.

pfNoRecord (PBOOL) – output
Record message flag.

TRUE The message will not be recorded in the JournalRecordHook hook.

Unless journal recording is specifically needed by the application, this parameter should always be TRUE.

FALSE The message will be recorded in the JournalRecordHook hook.

Returns

rc (BOOL) – returns
Return code.

TRUE The QMSG structure contains the current message to be passed in for handling.

FALSE The QMSG structure was not filled in. There are no further messages to pass in.

Remarks

This hook is intended for simulated user input, and only mouse and keyboard messages should be passed in. All other messages will be discarded. Mouse and keyboard messages injected into this hook will have the same effect as if they were generated by the mouse or keyboard device driver. The messages are routed in the same manner as normal user input.

The *pfNoRecord* parameter defaults to TRUE, implying that messages received from the MsgInputHook hook will not be recorded in the JournalRecordHook hook. Injected messages should not be recorded unless necessary.

When all messages have been passed in the application must remove the MsgInputHook hook using WinReleaseHook.

RegisterUserHook

This hook is called whenever a user message or data type is registered.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL RegisterUserHook (HAB hab, SHORT idContext, USHORT msg,
                       SHORT type1, SHORT dir1, SHORT type2, SHORT dir2,
                       SHORT typer, SHORT uShort, PSHORT arRMP,
                       PBOOL fRegistered)
```

Parameters

hab (HAB) – input
The application anchor block.

idContext (SHORT) – input
Origin of the call to hook.

RUMHK_DATATYPE	WinRegisterUserDatatype was called.
RUMHK_MSG	WinRegisterUserMsg was called.

msg (USHORT) – input
Message identifier.

For messages, this parameter is the message identifier. For data types, this is set to 0.

This must not be less than WM_USER and must not have been defined previously.

type1 (SHORT) – input
Data type.

For messages, this parameter is the data type of message parameter 1. For data types, this is the datatype being registered.

Valid data types are listed below. For data types that are shorter than 4 bytes, the data must be placed in the least-significant bytes, with the most significant bytes nullified (unsigned data types) or signed-extended (signed data types).

DTYP_BIT16	See USHORT.
DTYP_BIT32	See ULONG.
DTYP_BIT8	See UCHAR.
DTYP_BOOL	See BOOL.
DTYP_LONG	See LONG.
DTYP_SHORT	See SHORT.

DTYP_UCHAR	See UCHAR.
DTYP_ULONG	See ULONG.
DTYP_USHORT	See USHORT.
DTYP_P*	Pointer to an system data type. Note that not all of the system data types that exist in CPI are valid.
< -DTYP_USER	Pointer to a user data type. The user data type must have already been defined via WinRegisterUserDatatype.

dir1 (SHORT) – input
Direction of message parameter 1.

For messages, this parameter is the direction of message parameter 1. For data types, this is set to 0.

If the message parameter is a pointer, the direction values listed below apply to the *contents* of the storage location pointed at, as well as to the message parameter itself.

RUM_IN	Input parameter (inspected by the recipient of the message, but not altered)
RUN_OUT	Output parameter (altered by the recipient of the message, without inspecting its value first)
RUN_INOUT	Input/output parameter (inspected by the recipient of the message, and then altered).

type2 (SHORT) – input
Data type of message parameter 2.

For messages, this parameter is the data type of message parameter 2. For data types, this is set to 0.

See the description in *type1*.

dir2 (SHORT) – input
Direction of message parameter 2.

For messages, this parameter is the direction of message parameter 2. For data types, this is set to 0.

See the description in *dir1*.

typer (SHORT) – input
Data type of message reply.

For messages, this parameter is the data type of message reply For data types, this is set to 0.

See the description in *type1*.

uShort (SHORT) – input
Number of data type codes.

For data types, this parameter is the number of data type codes in *arRMP*, This value must not be less than one.

For messages, this parameter is set to 0.

arRMP (PSHORT) – input
Array of data type codes.

For data types, this parameter is an array of data type codes. For messages, this parameter is set to NULL.

Valid data types are the system-defined data types and their pointer equivalents, application-defined data types and their pointer equivalents, and control data types. Note that not all of the data types that occur in the CPI an be specified in this function.

A control data type is followed by one or more entries in the *arRMP* array that are interpreted in a special way. Control data types allow arrays, offsets, and lengths to be defined.

Simple Data Types:

DTYP_ATOM	See ATOM.
DTYP_BIT16	See USHORT.
DTYP_BIT32	See ULONG.
DTYP_BIT8	See UCHAR.
DTYP_BOOL	See BOOL.
DTYP_COUNT2	See USHORT.
DTYP_COUNT2B	See USHORT.
DTYP_COUNT2CH	See USHORT.
DTYP_COUNT4B	See ULONG.
DTYP_CPID	See USHORT.
DTYP_ERRORID	See ERRORID.
DTYP_IDENTITY	See USHORT.
DTYP_IDENTITY4	See ULONG.
DTYP_INDEX2	See USHORT.
DTYP_IPT	See IPT.
DTYP_LENGTH2	See USHORT.
DTYP_LENGTH4	See ULONG.
DTYP_LONG	See LONG.
DTYP_OFFSET2B	See USHORT.
DTYP_PID	See PID.
DTYP_PIX	See PIX.
DTYP_PROGCATEGORY	See PROGCATEGORY.
DTYP_PROPERTY2	See USHORT.
DTYP_PROPERTY4	See LONG.
DTYP_RESID	See HMODULE.
DTYP_SEGOFF	See SEGOFF.
DTYP_SHORT	See SHORT.
DTYP_TID	See TID.
DTYP_TIME	See LONG.
DTYP_UCHAR	See UCHAR.
DTYP_ULONG	See ULONG.

DTYP_USHORT	See USHORT.
DTYP_WIDTH4	See LONG.
DTYP_WNDPROC	See PFNWP.

Handle Data Types:

DTYP_HAB	See HAB.
DTYP_HACCEL	See HACCEL.
DTYP_HAPP	See HAPP.
DTYP_HATOMTBL	See HATOMTBL.
DTYP_HBITMAP	See HBITMAP.
DTYP_HDC	See HDC.
DTYP_HENUM	See HENUM.
DTYP_HINI	See HINI.
DTYP_HLIB	See HLIB.
DTYP_HMF	See HMF.
DTYP_HMQ	See HMQ.
DTYP_HPOINTER	See HPOINTER.
DTYP_HPROGRAM	See HPROGRAM.
DTYP_HPS	See HPS.
DTYP_HRGN	See HRGN.
DTYP_HSEM	See HSEM.
DTYP_HSPL	See HSPL.
DTYP_HSWITCH	See HSWITCH.
DTYP_HWND	See HWND.

Character/String/Buffer Data Types:

DTYP_BYTE	See BYTE.
DTYP_CHAR	See CHAR.
DTYP_STRL	See PSZ.
DTYP_STR16	See STR16.
DTYP_STR32	See STR32.
DTYP_STR64	See STR64.
DTYP_STR8	See STR8.

Structure Data Types:

DTYP_ACCEL	See ACCEL.
DTYP_ACCELTABLE	See ACCELTABLE.
DTYP_ARCPARAMS	See ARCPARAMS.
DTYP_AREABUNDLE	See AREABUNDLE.
DTYP_BITMAPINFO	See BITMAPINFO.
DTYP_BITMAPINFOHEADER	See BITMAPINFOHEADER.
DTYP_BTNCDATA	See BTNCDATA.
DTYP_CATCHBUF	See CATCHBUF.
DTYP_CHARBUNDLE	See CHARBUNDLE.
DTYP_CLASSINFO	See CLASSINFO.
DTYP_CREATESTRUCT	See CREATESTRUCT.
DTYP_CURSORINFO	See CURSORINFO.
DTYP_DEVOPENSTRUC	See DEVOPENSTRUC.

DTYP_DLGTEMPLATE	See DLGTEMPLATE.
DTYP_DLGTITEM	See DLGTITEM.
DTYP_ENTRYFDATA	See ENTRYFDATA.
DTYP_FATTRS	See FATTRS.
DTYP_FFDESCS	See FFDESCS.
DTYP_FIXED	See FIXED.
DTYP_FONTMETRICS	See FONTMETRICS.
DTYP_FRAMECDATA	See FRAMECDATA.
DTYP_GRADIENTL	See GRADIENTL.
DTYP_HCINFO	See HCINFO.
DTYP_IMAGEBUNDLE	See IMAGEBUNDLE.
DTYP_KERNINGPAIRS	See KERNINGPAIRS.
DTYP_LINEBUNDLE	See LINEBUNDLE.
DTYP_MARGSTRUCT	See MLEMARGSTRUCT.
DTYP_MARKERBUNDLE	See MARKERBUNDLE.
DTYP_MATRIXLF	See MATRIXLF.
DTYP_MLECTLDATA	See MLECTLDATA.
DTYP_OVERFLOW	See MLEOVERFLOW.
DTYP_OWNERITEM	See OWNERITEM.
DTYP_POINTERINFO	See POINTERINFO.
DTYP_POINTL	See POINTL.
DTYP_PROGRAMENTRY	See PROGRAMENTRY.
DTYP_PROGTYPE	See PROGTYPE.
DTYP_QMSG	See QMSG.
DTYP_RECTL	See RECTL.
DTYP_RGB	See RGB.
DTYP_RGNRECT	See RGNRECT.
DTYP_SBCDATA	See SBCDATA.
DTYP_SIZEF	See SIZEF.
DTYP_SIZEL	See SIZEL.
DTYP_SWBLOCK	See SWBLOCK.
DTYP_SWCNTRL	See SWCNTRL.
DTYP_SWENTRY	See SWENTRY.
DTYP_SWP	See SWP.
DTYP_TRACKINFO	See TRACKINFO.
DTYP_USERBUTTON	See USERBUTTON.
DTYP_WNDPARAMS	See WNDPARAMS.
DTYP_WPOINT	See WPOINT.
DTYP_WRECT	See WRECT.
DTYP_XYWINSIZE	See XYWINSIZE.

Pointer Data Type:

DTYP_P*	Pointer to an item of data type DTYP_*, where DTYP_* is one of the data types in the preceding lists. The value of a pointer data type is the value of the corresponding non-pointer data type prefixed with a minus to make it negative.

Minimum Application Data Type:

DTYP_USER — Minimum value for application-defined non-pointer data types.

Control Data Type:

DTYP_CTL_ARRAY — This starts a sequence of three array elements that define an array; the array resides in the structure being defined, and may have a fixed number of elements or a variable number of elements.

The following describes the possible elements:

n DTYP_CTL_ARRAY

n+1 data type of array data.

n+2 minus the number of elements in the array (for an array of fixed size), or the index of the element in *typer* corresponding to the structure component which contains the number of elements in the array being defined. This component must have a suitable numeric data type. The array-size element must recede element "n" in *typer*. The index is zero-based.

DTYP_CTL_LENGTH — This starts a sequence of four array elements that define a structure component containing the length of part or all of the structure. The length component resides at this point in the structure.

The following describes the possible elements:

n DYP_CTL_LENGTH

n+1 data type of structure component that contains the length (must be a suitable numeric data type).

n+2 the index of the element in *typer* corresponding to the first structure component that is included in the length; a value of -1 denotes the start of the structure. This index is zero-based.

The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements.

N+3 the index of the element in *typer* corresponding to the last structure component that is included in the length; it must not be less than the value contained in element n+2. A value of -1 denotes the end of the structure. The index is zero-based.

The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements.

If the value is -1, the length includes all offset data residing at the end of the structure.

DTYP_CTL_OFFSET

This starts a sequence of four array elements that define data addressed by an offset. The offset resides at this point in the structure, and contains the offset in bytes of the data from the start of the *outermost* structure in which this component resides. The data addressed by the offset must occupy storage following the fixed part of the structure. The data may be scalar data or array data.

The following describes the possible elements:

n DTYP_CTL_OFFSET

n+1 data type of the structure component that contains the offset (must be a suitable unsigned numeric data type).

n+2 data type of offset data.

n+3 minus the number of elements in the array (for an array of fixed size), or the index of the element in *typer* corresponding to the structure component that contains the number of elements in the array being defined; this component must have a suitable numeric data type; the array-size element may occur earlier or later in the structure. This index is zero-based.

A value of -1 indicates that the data is not an array.

DTYP_CTL_PARRAY

This starts a sequence of three array elements that define a pointer to an array. The pointer resides at this point in the structure, and the array resides elsewhere. The array can have a fixed or variable number of elements.

The following describes the possible elements:

n DTYP_CTL_PARRAY

n+1 data type of array data.

n+2 minus the number of elements in the array (for an array of fixed size) or the index of the element in *typer* corresponding to the structure component that contains the number of elements in the array being defined. This component must have a suitable numeric data type. The array-size element may

occur earlier or later in the structure. The index is zero-based.

fRegistered (PBOOL) – input
Flag indicating that a message or data type was registered.

TRUE	Message/data type was registered.
FALSE	Message/data type was not registered.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Errors occurred.

Remarks

This hook is called when a call is made to either WinRegisterUserDatatype or WinRegisterUserMsg.

SendMsgHook

This hook filters messages sent by the WinSendMsg function.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
VOID SendMsgHook  (HAB hab, PSMHSTRUCT psmh, BOOL fInterTask)
```

Parameters

hab (HAB) – input
Anchor-block handle.

psmh (PSMHSTRUCT) – input
Pointer to a send message hook structure.

This a structure contains the parameters to the WinSendMsg function.

fInterTask (BOOL) – input
Intertask indicator.

TRUE	The message is sent between tasks (intertask).
FALSE	The message is sent within a task (intratask).

Returns

There is no return value for this hook.

Remarks

This hook may be called whenever a window procedure is called via the WinSendMsg function.

It is called in the context of the sender, whereby if the sender has a queue hook installed it is called, but if the receiver has a queue hook installed it is not called.

The next hook in the chain is always called.

ThunkProc

This procedure provides pointer conversion for application-defined messages.

Syntax

```
#define INCL_NONE
#include <os2.h>
MRESULT ThunkProc (HWND hwnd, USHORT usmsg, MPARAM mpParam1,
                   MPARAM mpParam2, PFNWP pWndProc)
```

Parameters

hwnd (HWND) – input
Window handle.

usmsg (USHORT) – input
Message identity.

This is an application-defined message. The value is greater than or equal to WM_USER.

mpParam1 (MPARAM) – input
Message parameter 1.

mpParam2 (MPARAM) – input
Message parameter 2.

pWndProc (PFNWP) – input
Window-procedure identifier.

Returns

mresReply (MRESULT) – returns
Message-return data.

Remarks

Pointer conversion is normally performed automatically by the operating system. An application needs to provide its own pointer-conversion procedures only for application-defined messages which may be passed from 16-bit code to 32-bit code.

A pointer-conversion procedure is associated with a window by the WinSetWindowThunkProc and WinSetClassThunkProc functions.

The logic of the pointer-conversion procedure is as follows:

1. Convert each message parameter, if necessary. This may include converting any data structures to which the parameter points.
2. Call the window procedure referenced by the *pWndProc* parameter, supplying as arguments *hwnd*, *usmsg*, *mpParam1* and *mpParam2*.
3. Collect the return value and, if necessary, convert it.

   Note that structures to which the return value might point cannot be converted.
4. Convert any structures referenced by message parameters which might have been modified by the window procedure. Note that the pointer-conversion procedure should ensure that the original memory is still available before converting the structures.

A pointer-conversion procedure should process only those messages that it recognizes. On receiving unrecognized messages, it should set *usmsg* to 0.

WindowDCHook

This hook is called when a device context is allocated or freed.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL WindowDCHook (HAB hab, HDC hdc, HWND HWND, BOOL flAssociate)
```

Parameters

hab (HAB) – input
The application anchor block.

hdc (HDC) – input
The current device-context handle.

HWND (HWND) – input
The current window handle.

flAssociate (BOOL) – input
Association flag.

TRUE	Device context has been allocated.
FALSE	Device context has been freed.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion.
FALSE	Errors occurred.

WndProc

This defines the window procedure provided by an application.

Syntax

```
#define INCL_WINMESSAGEMGR /* Or use INCL_WIN, INCL_PM, Also in COMON section */
#include <os2.h>
MRESULT WndProc  (HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
```

Parameters

hwnd (HWND) – input
Window handle.

msg (ULONG) – input
Message identity.

mp1 (MPARAM) – input
Message parameter 1.

mp2 (MPARAM) – input
Message parameter 2.

Returns

mresReply (MRESULT) – returns
Message-return data.

Remarks

This procedure is associated with a window by the *pfnWndProc* of the WinRegisterClass function.

The window procedure typically processes only some of the messages passed to it. Those messages it does not process must be passed on to the WinDefWindowProc function, which performs the standard window processing for those messages.

Window Procedures

The first section of the chapter describes the window procedures that the application can provide. These procedures are:

- DialogProc
- ThunkProc
- WndProc

CheckMsgFilterHook

This hook is called whenever WinGetMsg, WinWaitMsg, or WinPeekMsg are used to filter message identities.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL CheckMsgFilterHook  (HAB hab, PQMSG pQmsg, ULONG usFirst,
                          ULONG usLast, ULONG fOptions)
```

Parameters

hab (HAB) – input
Anchor-block handle.

pQmsg (PQMSG) – input
Pointer to the QMSG structure of the message currently being reviewed.

usFirst (ULONG) – input
First message identity specified on a call to the WinGetMsg, WinPeekMsg or WinWaitMsg function.

usLast (ULONG) – input
Last message identity specified on a call to the WinGetMsg, WinPeekMsg or WinWaitMsg function.

fOptions (ULONG) – input
Message removal options.

PM_REMOVE	Message is being removed from queue
PM_NOREMOVE	Message is not being removed from queue.

Returns

rc (BOOL) – returns
Processing indicator.

TRUE The message is accepted by the filtering. Any further Check Message Filter Hooks in the chain are ignored, any filtering specified by the *ulFirst* and *ulLast* parameters of the WinGetMsg, WinPeekMsg or WinWaitMsg functions are ignored, and processing of the message continues.

A hook that always returns TRUE effectively switches off message filtering.

FALSE The message is passed on to the next Check Message Filter Hook in the chain. If the end of the chain has been reached, the filtering specified by the *ulFirst* and *ulLast* parameters of the WinGetMsg, WinPeekMsg or WinWaitMsg functions is applied.

Remarks

This hook enables an application to apply a very specific message filtering, for example, based on the values of message parameters.

This hook is called after window handle filtering and before message filtering. Window handle filtering is controlled by the *hwndFilter* parameter of the WinGetMsg or WinPeekMsg functions. Message filtering is controlled by the *ulFirst* and *ulLast* parameters of the WinGetMsg, WinPeekMsg or WinWaitMsg functions.

This hook is called if the message passes window handle filtering and if non-null message filtering is specified. This means that, on entry to this hook:

- The *hwndFilter* parameter of the WinGetMsg or WinPeekMsg function is either NULLHANDLE or it specifies the window (or a parent of the window) referenced in the *pQmsg* structure.
- At least one of the *usFirst* and *usLast* parameters are nonzero.
- The *msg* field of the *pQmsg* structure might or might not lie inside the range specified by the *usFirst* and *usLast* parameters.

CodePageChangedHook

This hook notifies that a message queue code page has been changed.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
VOID CodePageChangedHook (HMQ hmq, ULONG usOldCodePage,
                          ULONG usNewCodePage)
```

Parameters

hmq (HMQ) – input
Message-queue handle.

The handle of the message queue that is changing its code page.

usOldCodePage (ULONG) – input
Previous code page.

usNewCodePage (ULONG) – input
New code page.

Returns

There is no return value for this hook.

Remarks

This hook is sent to all hooks chained under HK_CODEPAGECHANGE, regardless of the return value.

The new code page is set before this hook is called.

DestroyWindowHook

This hook is called whenever a window is destroyed.

Syntax

```
#define INCL_WINHOOKS /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>
BOOL DestroyWindowHook  (HAB hab, HWND Hwnd, ULONG ulReserved)
```

Parameters

hab (HAB) – input
Anchor-block handle.

Hwnd (HWND) – input
The handle of the window being destroyed.

ulReserved (ULONG) – input
Reserved.

Returns

rc (BOOL) – returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

Remarks

This hook is sent after the WM_DESTROY message has been sent and just before the window becomes invalid.

Related Messages

- WM_DESTROY

Appendix A. Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights or other legally protectable rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood NY 10594, U.S.A.

Trademarks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States or other countries:

IBM
Common User Access
CUA
OS/2
Presentation Manager

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows. Other trademarks are trademarks of their respective companies.

Adobe	Adobe Systems Incorporated
C++	AT&T, Incorporated
Helvetica	Linotype
Microsoft	Microsoft Corporation
Pentium	Intel Corporation
PostScript	Adobe Systems Incorporated
Times New Roman	Monotype
Windows	Microsoft Corporation

Index

E

N

P

Q

R

S

T

U

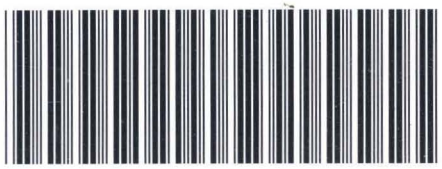
V

W

25H7190
G25H-7190-00

Printed on recycled paper.

Printed in U.S.A.



G25H-7190-00

P25H7190