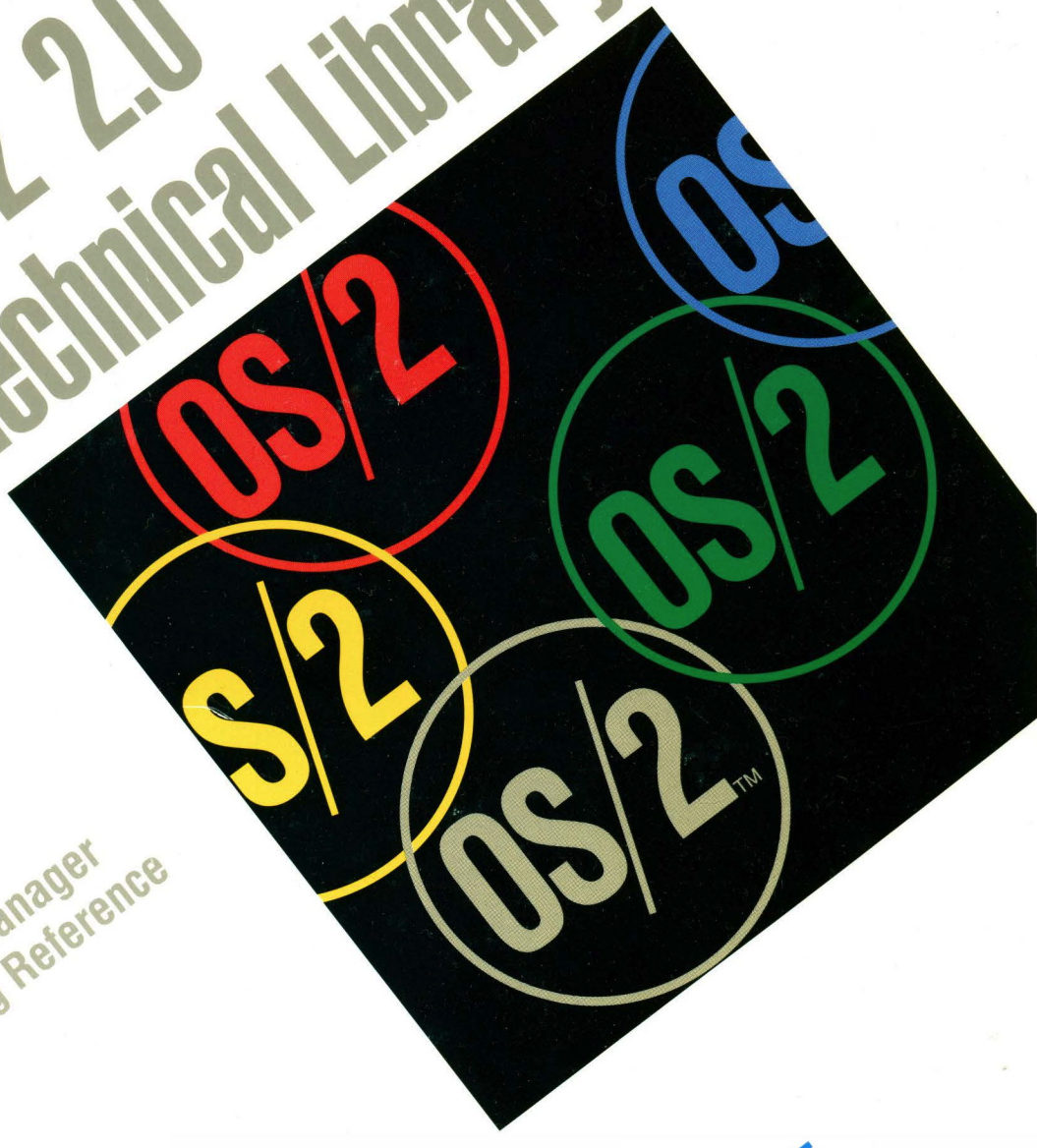
OS/2 2.0 Technical Library



Presentation Manager Programming Reference Volume II

Version 2.00

IBM Programming Family

OS/2 2.0 Technical Library

Presentation Manager Programming Reference Volume II

Version 2.00

Programming Family

Note

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

First Edition (March 1992)

About this Book

The *Presentation Manager Programming Reference* is a detailed technical reference, in three volumes, for application programmers creating programs using the Presentation Manager interface.

Chapter 1 contains important information. You should read it before using this book.

This reference does not give guidance on how to use the functions, nor does it contain information about how the functions are related to each other. It is intended to be used in conjunction with the *Programming Guide Volumes II and III*.

Prerequisite Knowledge

The OS/2 2.0 Technical Library is intended for professional application developers knowledgeable in at least one programming language in which OS/2 programs can be written. The information in the Technical Library assumes that you are new to programming with OS/2 and the Presentation Manager. You should understand the OS/2 services available to users.

Related Publications

The *Application Design Guide* and the *Programming Guide Volumes I, II, and III* introduce the programming concepts that you should understand before you begin developing applications to run on the OS/2 operating system. *Getting Started* describes the online programming books, tools, programming aids, and sample programs that make up the IBM Developer's Toolkit for OS/2 2.0.

Organization of this Book

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

Volume I (Functions)

Volume II (Functions and Workplace)

Volume III (Related Information and Data Types)

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 protectible 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 Commercial Relations, IBM Corporation, Purchase, NY 10577.

The following terms, denoted by an asterisk(*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

IBM
Common User Access
CUA
Operating System/2
OS/2
Presentation Manager
SAA
System Application Architecture

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows:

Adobe	Adobe Systems Incorporated
Helvetica	Linotype AG
LaserJet	Hewlett-Packard Company
Intel	Intel Corporation
Microsoft	Microsoft Corporation
PostScript	Adobe Systems Incorporated
Times New Roman	Monotype Corporation
Windows	Microsoft Corporation

Chapter 8. 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	
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
Dialog Boxes	
WinCreateDlg	WinFileDlg
WinDefDlgProc	WinFontDlg
WinDefFileDlgProc	WinFreeFileDlgList

C Name	C Name
WinDefFontDlgProc	WinGetDlgMsg
WinDismissDlg	WinLoadDlg
WinDlgBox	WinProcessDlg
WinEnableControl	
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	
Dynamic Data Exchange (DDE) Support	
WinDdeInitiate	WinDdeRespond
WinDdePostMsg	
Error Processing	
WinFreeErrorInfo	WinGetLastError
WinGetErrorInfo	
Help Manager	
WinAssociateHelpInstance	WinDestroyHelpInstance
WinCreateHelpInstance	WinLoadHelpTable
WinCreateHelpTable	WinQueryHelpInstance
Initialization and Termination	
WinCancelShutdown	WinQueryAnchorBlock
WinInitialize	WinQueryVersion
WinTerminate	
Keyboard	
WinEnablePhysInput	WinIsPhysInputEnabled

C Name	C Name
WinFocusChange	WinQueryFocus
WinGetKeyState	WinSetFocus
WinGetPhysKeyState	WinSetKeyboardStateTable
Library Support	
WinDeleteLibrary	WinLoadLibrary
WinDeleteProcedure	WinLoadProcedure
List Box	
WinDeleteLboxItem	WinQueryLboxItemTextLength
WinInsertLboxItem	WinSetLboxItemText
WinQueryLboxCount	WinQueryLboxSelectedItem
WinQueryLboxItemText	
Menus	
WinCheckMenuItem	WinIsMenuItemValid
WinCreateMenu	WinLoadMenu
WinEnableMenuItem	WinPopupMenu
WinIsMenuItemChecked	WinSetMenuItemText
WinIsMenuItemEnabled	
Message Management	
WinBroadcastMsg	WinQueryQueueInfo
WinCreateMsgQueue	WinQueryQueueStatus
WinDestroyMsgQueue	WinRegisterUserDatatype
WinDispatchMsg	WinRegisterUserMsg
WinGetMsg	WinSendDlgItemMsg
WinInSendMsg	WinSendMsg
WinLoadMessage	WinSetClassMsgInterest
WinPeekMsg	WinSetMsgInterest
WinPostMsg	WinSetMsgMode
WinPostQueueMsg	WinSetSynchroMode
WinQueryMsgPos	WinWaitMsg
WinQueryMsgTime	WinRequestMutexSem
WinWaitEventSem	WinWaitMuxWaitSem
Mouse Capture	
WinQueryCapture	WinSetCapture
Mouse Tracking	
WinShowTrackRect	WinTrackRect
Pointer	
WinCreatePointer	WinQueryPointerPos
WinCreatePointerIndirect	WinQuerySysPointer
WinDestroyPointer	WinSetPointer
WinLoadPointer	WinSetPointerPos

C Name	C Name
WinQueryPointer	WinShowPointer
WinQueryPointerInfo	
Rectangles	
WinCopyRect	WinOffsetRect
WinEqualRect	WinPtInRect
WinInflateRect	WinSetRect
WinIntersectRect	WinSetRectEmpty
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 Colors	
WinQuerySysColor	WinSetSysColors
System Modal Management	
WinQuerySysModalWindow	WinSetSysModalWindow
System Values	
WinQuerySysValue	WinSetSysValue
Timers	
WinGetCurrentTime	WinStopTimer
WinStartTimer	

C Name	C Name
Window Management	
Activation, Size and Position	
WinGetMaxPosition	WinSaveWindowPos
WinGetMinPosition	WinSetActiveWindow
WinQueryActiveWindow	WinSetMultWindowPos
WinQueryWindowPos	WinSetWindowPos
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	
Window Hierarchies	
WinBeginEnumWindows	WinMultWindowFromIDs
WinEndEnumWindows	WinQueryWindow
WinEnumDlgItem	WinSetOwner
WinGetNextWindow	WinSetParent
WinIsChild	
Mixed Memory Model Support	
WinQueryClassThunkProc	WinSetClassThunkProc
WinQueryWindowModel	WinSetWindowThunkProc
WinQueryWindowThunkProc	
Window Text	
WinQueryDlgItemShort	WinQueryWindowTextLength
WinQueryDlgItemText	WinSetDlgItemShort
WinQueryDlgItemTextLength	WinSetDlgItemText
WinQueryWindowText	WinSetWindowText
Window Words	
WinQueryWindowPtr	WinSetWindowPtr
WinQueryWindowULong	WinSetWindowULong
WinQueryWindowUShort	WinSetWindowUShort

C Name	C Name
WinSetWindowBits	

```
#define INCL_WINATOM   /* Or use INCL_WIN or INCL_PM */
```

ATOM WinAddAtom (HATOMTBL hatomtblAtomTbl, PSZ pszAtomName)

This function adds an atom to an atom table.

Parameters

hatomtblAtomTbl (HATOMTBL) – input
Atom-table handle.

This is the handle returned by a previous call to WinCreateAtomTable or WinQuerySystemAtomTable.

pszAtomName (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 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	An invalid atom-table handle was specified.
PMERR_INVALID_INTEGER_ATOM	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND	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.

WinAddAtom – Add Atom

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 — Add Switch Entry

#define INCL_WINSWITCHLIST /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

```
HSWITCH WinAddSwitchEntry (PSWCNTRL pswctlSwitchData)
```

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.

Parameters

pswctlSwitchData (PSWCNTRL) — input
Switch data.

Contains information about the newly created Window List entry.

If the *szSwtitle[MAXNAMEL + 1]* field of the SWCNTRL structure is 0, the system uses the name under which the application is started. This only applies for programs written for OS/2 Versions 1.1 and later, and only for the first call to this function since the program started. Otherwise, a NULL entry name is invalid.

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

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	The limit on the number of Window List entries has been reached with WinAddSwitchEntry.
PMERR_INVALID_WINDOW	The window specified with a Window List function is not a valid frame window.
PMERR_INVALID_SESSION_ID	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.

WinAddSwitchEntry — Add Switch Entry

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       */

hswitch = WinAddSwitchEntry(&swctl);
```

#define INCL_WINDIALOGS /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinAlarm (HWND hwndDeskTop, ULONG flStyle)

This function generates an audible alarm.

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 X'0025' through X'7FFF'. The alarm is not generated if system value SV_ALARM is set to FALSE. The alarms are dependent on the device capability.

Different alarms are selected by use of these values:

WA_WARNING

WA_NOTE

WA_ERROR

Returns

Alarm-generated indicator:

TRUE Alarm generated

FALSE Alarm not generated.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

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

WinAlarm — Sound Alarm

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 – Associate Help Instance

```
#define INCL_WINHELP  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinAssociateHelpInstance (HWND hwndHelpInstance, HWND hwndApp)

This function associates the specified instance of the help manager with the window chain of the specified application window.

Parameters

hwndHelpInstance (HWND) – input
Handle of an instance of the help manager.

This is the handle returned by the WinCreateHelpInstance call.

NULLHANDLE Dissociates 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

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

WinAssociateHelpInstance – Associate Help Instance

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 – Begin Window Enumeration

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

HENUM WinBeginEnumWindows (HWND hwndParent)

This function begins the enumeration process for all of the immediate child windows of a specified window.

Parameters

hwndParent (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

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	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 *Henum* parameter and does not change during the enumeration by the WinGetNextWindow call. The windows are enumerated in the z-order at the time enumeration is begun, with the topmost child window enumerated first.

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 — Begin Paint

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
HPS WinBeginPaint (HWND hwnd, HPS hps, PRECTL prclRect)
```

This function obtains a presentation space whose associated update region is set ready for drawing in a specified window.

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.

prclRect (PRECTL) — output
Bounding rectangle:

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 No bounding rectangle; that is, repainting is not required.

Other Specifies the smallest rectangle bounding the update region, in window coordinates.

Returns

Presentation-space handle:

NULLHANDLE Error occurred

Other Presentation-space handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND An invalid window handle was specified.

PMERR_INV_HPS An invalid presentation-space handle was specified.

Remarks

This function is generally made during the processing of a WM_PAINT message when the application needs to update the content of the window.

If the presentation space already exists, its update region is set and the device context of the window is associated with the presentation space. Otherwise, a cache presentation space is obtained specifically for the window.

The update region associated with *hwnd* is reset to 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 — Broadcast Message

```
#define INCL_WINMESSAGEMGR  /* Or use INCL_WIN or INCL_PM */

BOOL WinBroadcastMsg (HWND hwndParent, ULONG ulMsgId, MPARAM mpParam1,
                      MPARAM mpParam2, ULONG flCmd)
```

This function broadcasts a message to multiple windows.

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

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
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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 – Calculate Frame Rectangle

```
#define INCL_WINFRAMEMGR  /* Or use INCL_WIN or INCL_PM */

BOOL WinCalcFrameRect (HWND hwnd, PRECTL prclRect, BOOL fFrame)
```

This function calculates a client rectangle from a frame rectangle, or a frame rectangle from a client rectangle.

Parameters

hwnd (HWND) – input
Frame-window handle.

prclRect (PRECTL) – input/output
Window rectangle.

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.

fFrame (BOOL) – input
Frame indicator:

TRUE Frame rectangle provided

FALSE Client-area rectangle provided.

Returns

Rectangle-calculated indicator:

TRUE Rectangle successfully calculated

FALSE Error occurred, or the calculated rectangle is empty.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 *hwnd* is hidden; *hwnd* 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 — Call Message Filter

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */

BOOL WinCallMsgFilter (HAB hab, PQMSG pqmsgpqmsg, ULONG ulFilter)
```

This function calls a message-filter hook.

Parameters

hab (HAB) — input
Anchor-block handle.

pqmsgpqmsg (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 one of the standard MSGF_* values (see MsgFilterHook) or an application-specific value.

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  pqmsgpqmsg;       /* Message to be passed to the
                           message-filter hook                  */
ULONG  ulFilter;        /* Filter                               */
POINTL ptrPos = {5L,5L};/* pointer position                     */

/* initialize message structure */
pqmsgpqmsg.hwnd  = HWND_DESKTOP;
pqmsgpqmsg.msg   = WM_CLOSE;
pqmsgpqmsg.mp1   = MPVOID;
pqmsgpqmsg.mp2   = MPVOID;
pqmsgpqmsg.time  = 0L;
pqmsgpqmsg.ptl   = ptrPos;

/* call hook in message box mode */
ulFilter = MSGF_MESSAGEBOX;

fHookRet = WinCallMsgFilter(hab, &pqmsgpqmsg, ulFilter);
```

WinCancelShutdown – Cancel Shutdown

```
#define INCL_WINMESSAGEMGR  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
BOOL WinCancelShutdown (HMQ hmq, BOOL fCancelAlways)
```

This function cancels a request for an application to shut down.

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

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 process 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

- WinTerminate
- 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 — Change Switch Entry

```
#define INCL_WINSWITCHLIST   /* Or use INCL_WIN or INCL_PM */
```

```
ULONG WinChangeSwitchEntry (HSWITCH hswitchSwitch, PSWCNTRL pswctlSwitchData)
```

This function changes the information in a Window List entry.

Parameters

hswitchSwitch (HSWITCH) — input
Handle to the Window List entry to be changed.

pswctlSwitchData (PSWCNTRL) — 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

Return code:

0	Successful completion
Other	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_SWITCH_HANDLE	An invalid Window List entry handle was specified.
PMERR_INVALID_WINDOW	The window specified with a Window List function 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 — Set Checkstate of Button

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

USHORT WinCheckButton (HWND hwndDlg, USHORT usId, USHORT usChkstate)

This macro sets the checked state of the specified button control. It returns the previous check state.

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

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;
        }
```

WinCheckMenuItem — Check Menu Item

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinCheckMenuItem (HWND hwndMenu, USHORT usId, BOOL fCheck)

This macro sets the check state of the specified menu item to the flag.

Parameters

hwndMenu (HWND) — input
Menu window handle.

usId (USHORT) — input
Item identifier.

fCheck (BOOL) — input
Check flag.

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 — Close Clipboard

#define INCL_WINCLIPBOARD /* Or use INCL_WIN or INCL_PM */

BOOL WinCloseClipbrd (HAB hab)

This function closes the clipboard, allowing other applications to open it with the WinOpenClipbrd function.

Parameters

hab (HAB) — input
Anchor-block handle.

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);
```

#define INCL_WINCOUNTRY /* Or use INCL_WIN or INCL_PM */

```
ULONG WinCompareStrings (HAB hab, ULONG idCodepage, ULONG idCountryCode,
                         PSZ pszString1, PSZ pszString2, ULONG flOptions)
```

Compares two null-terminated strings defined using the same code page.

Parameters

hab (HAB) – input
Anchor-block handle.

idCodepage (ULONG) – input
Code page identity of both strings.

idCountryCode (ULONG) – input
Country code.

pszString1 (PSZ) – input
String 1.

pszString2 (PSZ) – input
String 2.

flOptions (ULONG) – input
Reserved.

0 Reserved value.

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	The specified string parameter is invalid.

Related Functions

- WinLoadString
- WinNextChar
- WinPrevChar
- WinSubstituteStrings
- WinUpper
- WinUpperChar

WinCompareStrings — Compare Strings

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 — Copy Accelerator Table

```
#define INCL_WINACCELERATORS  /* Or use INCL_WIN or INCL_PM */
```

```
ULONG WinCopyAccelTable (HACCEL hAccel, PACCELTABLE pacctAccelTable,
                         ULONG ulCopyMax)
```

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.

Parameters

hAccel (HACCEL) — input
Accelerator-table handle.

pacctAccelTable (PACCELTABLE) — input/output
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

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 An invalid accelerator-table handle was specified.

Related Functions

- WinCreateAccelTable
- WinDestroyAccelTable
- WinLoadAccelTable
- WinQueryAccelTable
- WinSetAccelTable
- WinTranslateAccel

WinCopyAccelTable — Copy Accelerator Table

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;
```

#define INCL_WINRECTANGLES /* Or use INCL_WIN or INCL_PM */

BOOL WinCopyRect (HAB hab, PRECTL prclDest, PRECTL prclSrc)

This function copies a rectangle from *prclSrc* to *prclDest*.

Parameters

hab (HAB) – input
Anchor-block handle.

prclDest (PRECTL) – output
Destination rectangle.

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.

prclSrc (PRECTL) – input
Source rectangle.

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.

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 – Translate Character with Code Page

```
#define INCL_WINCOUNTRY  /* Or use INCL_WIN or INCL_PM */
```

```
UCHAR WinCpTranslateChar (HAB hab, ULONG idCpSource, UCHAR ucSource,
                          ULONG idCpDest)
```

This function translates a character from one code page to another.

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

If nonzero, the translated or substitution (X'FF') character.

Possible returns from WinGetLastError

PMERR_INVALID_STRING_PARM	The specified string parameter is invalid.
PMERR_INVALID_SRC_CODEPAGE	The source code page parameter is invalid.
PMERR_INVALID_DST_CODEPAGE	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 (X'FF') was returned.

See the WinQueryCpList function for Code Page acceptable in this function.

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 — Translate String with Code Page

```
#define INCL_WINCOUNTRY   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinCpTranslateString (HAB hab, ULONG idCpSource, PSZ pszSource,
                           ULONG idCpDest, ULONG cbLenDest, PSZ pszDest)
```

This function translates a string from one code page to another.

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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_STRING_PARM	The specified string parameter is invalid.
PMERR_INVALID_SRC_CODEPAGE	The source code page parameter is invalid.
PMERR_INVALID_DST_CODEPAGE	The destination code page parameter is invalid.
PMERR_PARAMETER_OUT_OF_RANGE	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 (X'FF') was returned.

See the WinQueryCpList function for Code Pages acceptable in this function.

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 – Create Accelerator Table

```
#define INCL_WINACCELERATORS   /* Or use INCL_WIN or INCL_PM */
```

HACCEL WinCreateAccelTable (HAB hab, PACCELTABLE pacctAccelTable)

This function creates an accelerator table from the accelerator definitions in memory.

Parameters

hab (HAB) – input
Anchor-block handle.

pacctAccelTable (PACCELTABLE) – input
Accelerator table.

Returns

Accelerator-table handle.

Remarks

This function returns a different *hAccel* 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>

HACCEL  hAccel;         /* Accelerator-table handle             */
ACCELTABLE  pacctAccelTable;/* Accelerator-table data area      */
HAB     hab;            /* anchor-block handle                  */
/* in memory accelerator table */
ACCEL   acctable[] = {
         AF_SYSCOMMAND | AF_ALT | AF_VIRTUALKEY,VK_F4,SC_CLOSE,
          AF_SYSCOMMAND | AF_ALT | AF_VIRTUALKEY,VK_F7,SC_MOVE,
         AF_SYSCOMMAND | AF_ALT | AF_VIRTUALKEY,VK_F8,SC_SIZE};

/* initialize accelerator table structure, including in memory
   accelerator table */
pacctAccelTable.cAccel = 3;
pacctAccelTable.codepage = 437;
pacctAccelTable.aaccel[0] = acctable[0];

hAccel = WinCreateAccelTable(hab, &pacctAccelTable);
```

WinCreateAtomTable — Create Atom Table

```
#define INCL_WINATOM   /* Or use INCL_WIN or INCL_PM */

HATOMTBL WinCreateAtomTable (ULONG ulInitial, ULONG ulBuckets)
```

This function creates an empty atom table of the specified size.

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

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 — Create Cursor

```
#define INCL_WINCURSORS  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinCreateCursor (HWND hwnd, LONG lx, LONG ly, LONG lcx, LONG lcy, ULONG ulrgf,
                      PRECTL prclClip)
```

This function creates or changes a cursor for a specified window.

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 –32 768 through 32 767. The data type WRECT can also be used, if supported by the language.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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.

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 – Create Dialog

#define INCL_WINDIALOGS /* Or use INCL_WIN or INCL_PM */

```
HWND WinCreateDlg (HWND hwndParent, HWND hwndOwner, PFNWP pDlgProc,
                   PDLGTEMPLATE pdlgtDlgTmp, PVOID pCreateParams)
```

This function creates a dialog window.

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.

pDlgProc (PFNWP) – input
Dialog procedure for the created dialog window.

pdlgtDlgTmp (PDLGTEMPLATE) – input
Dialog template.

pCreateParams (PVOID) – input
Application-defined data area.

This is passed to the dialog procedure in the WM_INITDLG message.

Returns

Dialog-window handle:

NULLHANDLE	Dialog window not created
Other	Dialog-window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_INTEGER_ATOM	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND	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 *pdlgtDlgTmp* 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 — Create Frame Controls

```
#define INCL_WINFRAMEMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinCreateFrameControls (HWND hwndFrame, PFRAMECDATA pFcdata, PSZ pszTitle)

This function creates the standard frame controls for a specified window.

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 (FRAMECDATA) — input
Frame-control data.

This includes a combination of frame creation flags (FCF_*), that specifies which frame controls are to be created. For further information, see "Frame Creation Flags" on page 15-1.

pszTitle (PSZ) — input
Title string.

A string that is displayed in the WC_TITLEBAR control when FCF_TITLEBAR is specified in *pFcdata*.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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 Chapter 15, Frame Control Window Processing.

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 — Create Help Instance

```
#define INCL_WINHELP  /* Or use INCL_WIN or INCL_PM */
```

HWND WinCreateHelpInstance (HAB hab, PHELPINIT phinitHMInitStructure)

This function creates an instance of the help manager with which to request help manager functions.

Parameters

hab (HAB) — input
Anchor-block handle.

The handle of the application anchor block returned from the WinInitialize function.

phinitHMInitStructure (PHELPINIT) — input/output
Help manager initialization structure.

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 – Create Help Table

```
#define INCL_WINHELP  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinCreateHelpTable (HWND hwndHelpInstance, PHELPTABLE phtHelpTable)

This function is used to identify or change the help table.

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

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;
}
```

WinCreateMenu — Create Menu

```
#define INCL_WINMENUS   /* Or use INCL_WIN or INCL_PM */
```

HWND WinCreateMenu (HWND hwndOwner, PVOID pmtMenutmp)

This function creates a menu window from the menu template.

Parameters

hwndOwner (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.

pmtMenutmp (PVOID) — input
Menu template in binary format.

Returns

Menu-window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 FAR *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 — Create Message Queue

#define INCL_WINMESSAGEMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

HMQ WinCreateMsgQueue (HAB hab, LONG lQueuesize)

This function creates a message queue.

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.

Other Maximum queue size.

Returns

Message-queue handle:

NULLHANDLE Queue cannot be created.

Other Message-queue handle.

Possible returns from WinGetLastError

PMERR_HEAP_MAX_SIZE_REACHED	The heap has reached its maximum size (64KB), and cannot be increased.
PMERR_HEAP_OUT_OF_MEMORY	An attempt to increase the size of the heap failed.
PMERR_RESOURCE_NOT_FOUND	The specified resource identity could not be found.
PMERR_NOT_IN_A_PM_SESSION	An attempt was made to access function that is only available from PM programs from a non-PM session.
PMERR_QUEUE_ALREADY_EXISTS	An attempt to create a message queue for a thread failed because one 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.

The system default queue size allows for 10 messages.

Related Functions

- WinCancelShutdown
- WinInitialize
- WinTerminate
- WinBroadcastMsg
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- 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 — Create Workplace Object

#define INCL_WINWORKPLACE

```
HOBJECT WinCreateObject (PSZ pszClassName, PSZ pszTitle, PSZ pszSetupString,
                         PSZ pszLocation, ULONG ulFlags)
```

This WinCreateObject function creates an instance of object class *pszClassName*, with title *pszTitle*, and places the icon and title in the location referred to by *pszLocation*.

Parameters

pszClassName (PSZ) — input

A pointer to a zero-terminated string which contains the name of the class of which this object is a member.

pszTitle (PSZ) — input

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.
"<WP_DESKTOP>"	The 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:

- CO_FAILIFEXISTS
- CO_REPLACEIFEXISTS

Returns

Success indicator:

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 separated by commas, that change the behavior of the object. Each object class documents its keynames and the parameters parameters it expects to see immediately following. Note that ALL parameters have safe defaults, so it is never necessary to pass unnecessary parameters to an object.

These are the keyname – value pairs supported by the WPObject class:

KEYNAME	VALUE	Description
TITLE	Title	This sets the object's title. This is equivalent to calling the wpSetTitle method.
ICON	filename	This sets the object's icon. This is equivalent to calling the wpSetIconData method.
HELPPANEL	id	This sets the object's default help panel. This is equivalent to calling the wpSetDefaultHelp method.
TEMPLATE	YES	This sets the object's template property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_TEMPLATE.
	NO	This resets the object's template property.
NODELETE	YES	This sets the object's no delete property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_NODELETE.
	NO	This resets the object's no delete property.
NOCOPY	YES	This sets the object's no copy property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_NOCOPY.
	NO	This resets the object's no copy style.
NOMOVE	YES	This sets the object's no move property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_NOMOVE.
	NO	This resets the object's no move property.
NOLINK	YES	This sets the object's no link property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_NOLINK.
	NO	This resets the object's no link property.

Related Functions

- WinRegisterObjectClass
- WinDeregisterObjectClass
- WinDestroyObject
- WinReplaceObjectClass
- WinSetObjectData

WinCreatePointer — Create Pointer

```
#define INCL_WINPOINTERS   /* Or use INCL_WIN or INCL_PM */
```

```
HPOINTER WinCreatePointer (HWND hwndDeskTop, HBITMAP hbmBitMap, BOOL fPointerSize,
                           LONG lxHotspot, LONG lyHotspot)
```

This function creates a pointer from a bit map.

Parameters

hwndDeskTop (HWND) — input
Desktop-window handle or HWND_DESKTOP.

hbmBitMap (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.

fPointerSize (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.

lxHotspot (LONG) — input
x-offset of hot spot within pointer from its lower left corner (in pels).

lyHotspot (LONG) — input
y-offset of hot spot within pointer from its lower left corner (in pels).

Returns

Pointer handle:

NULLHANDLE	Error
Other	Handle of the newly created pointer.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_HBITMAP_BUSY	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
- WinSetPointer
- WinSetPointerPos
- 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 – Create Pointer Indirect

```
#define INCL_WINPOINTERS  /* Or use INCL_WIN or INCL_PM */
```

HPOINTER WinCreatePointerIndirect (HWND hwndDeskTop, PPOINTERINFO pptrIPointerInfo)

This function creates a colored pointer or icon from a bit map.

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle or HWND_DESKTOP.

pptrIPointerInfo (POINTERINFO) – input
Pointer information structure.

The fields in this structure must be set before the call is made:

- *ulPointer* is set to TRUE if a pointer is being created, or to FALSE if an icon is being created
- *xHotspot* and *yHotspot* are set to the relative position in the icon or pointer that is associated with the mouse position
- *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*, and the height of *hbmPointer* must be double that of *hbmColor* (to allow for both the AND and the XOR mask).

Returns

Pointer handle:

NULLHANDLE	Error
Other	Handle of the newly created pointer or icon.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_HBITMAP_BUSY	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
- WinSetPointer
- WinSetPointerPos
- 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 — Create Standard Window

```
#define INCL_WINFRAMEMGR  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
HWND WinCreateStdWindow (HWND hwndParent, ULONG flStyle, PULONG pflCreateFlags,
                         PSZ pszClassClient, PSZ pszTitle, ULONG flStyleClient,
                         HMODULE Resource, ULONG ulId, PHWND phwndClient)
```

This function creates a standard window.

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" on page 12-2) and the FS_* (see "Frame Control Styles" on page 15-3) 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 /FCFVALS/.

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" on page 11-2. 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 DosGetModHandle 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.

phwndClient (PHWND) — output
Client-window handle.

This is returned if a client window is created.

Returns

Frame-window handle.

This is NULLHANDLE if no window is created.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND	The specified atom name is not in the atom table.

WinCreateStdWindow — Create Standard Window

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 Chapter 15, "Frame Control Window Processing" on page 15-1. 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 – Create Switch Entry

```
#define INCL_WINSWITCHLIST   /* Or use INCL_WIN or INCL_PM */
```

HSWITCH WinCreateSwitchEntry (HAB hab, PSWCNTRL pswctlSwitchData)

This function adds an entry to the Window List.

Parameters

hab (HAB) – input
Anchor-block handle.

pswctlSwitchData (PSWCNTRL) – input
Switch data.

Contains information about the newly created Window List entry.

If the *szSwtitle*[*MAXNAMEL*+*1*] field of the SWCNTRL structure is NULL, the system uses the name under which the application is started. This only applies for OS/2 Version 2.0 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

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	The limit on the number of Window List entries has been reached with WinAddSwitchEntry.
PMERR_INVALID_WINDOW	The window specified with a Window List function is not a valid frame window.
PMERR_INVALID_SESSION_ID	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 — Create Window

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

```
HWND WinCreateWindow (HWND hwndParent, PSZ pszClassName, PSZ pszName,
                      ULONG flStyle, LONG lxcoord, LONG lycoord, LONG lWidth,
                      LONG lHeight, HWND hwndOwner, HWND hwndBehind, ULONG id,
                      PVOID pCtlData, PVOID pPresParams)
```

This function creates a new window of class *pszClassName* and returns *hwnd*.

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.

pszClassName (PSZ) — input
Registered-class name.

pszClassName is either an application-specified name as defined by WinRegisterClass or the name of a preregistered WC_* class; see Control Window Message Processing. Preregistered class names are of the form '#nnnnn', where nnnnn is 1 through 5 digits corresponding to the value of the WC_* class-name constant.

This parameter can also be specified directly as a WC_* constant.

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.

lxcoord (LONG) — input
x-coordinate of window position.

The value is in window coordinates relative to the origin of the parent window.

lycoord (LONG) — input
y-coordinate of window position.

The value is in window coordinates relative to the origin of the parent window.

lWidth (LONG) — input
Width of window, in window coordinates.

lHeight (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.

hwndBehind (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.

pCtlData (PVOID) — input
Control data.

This is class-specific data passed to the window procedure by the WM_CREATE message.

pPresParams (PVOID) — input
Presentation parameters.

This is class-specific presentation data passed to the window procedure by the WM_CREATE message.

Returns

Window handle:

NULLHANDLE	Error occurred
Other	Window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND	The specified atom name is not in the atom table.

Remarks

The appearance and behavior of a window are determined by its style, which is the combination of the style established by *pszClassName* and *flStyle* ORed together. Any of the standard styles WS_* (see "Window Styles" on page 12-2) can be used in addition to any class-specific styles that may be defined.

A window is usually created enabled and invisible. For more information on the initial state of a created window, see the list of the standard window styles.

Messages may be received from other processes or threads when this function is called.

This function sends the WM_CREATE message to the window procedure of the window being created.

This function sends the WM_ADJUSTWINDOWPOS message after the WM_CREATE message, and before the window is displayed (if applicable). The values passed are those given to the WinCreateWindow function. If the window has style WS_VISIBLE, the window is created visible.

The WM_SIZE message is not sent by the WinCreateWindow function while the window is being created. Any required size processing can be performed during the processing of the WM_CREATE message.

WinCreateWindow – Create Window

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 'new button' as a child of the Desk Top, located at (0,0) of size 200x100. window.

```
#define INCL_WINWINDOWMGR       /* Window Manager Functions     */
#define INCL_WINLISTBOXES       /* List Box definitions         */
#include <os2.h>

HWND    hwnd;           /* cursor display window                */
ULONG  ButtonId;        /* window id  (app supplied)            */
ULONG flStyle;          /* window style                         */

flStyle = WS_VISIBLE;                /* create window visible  */

/* create button window */
hwnd = WinCreateWindow(HWND_DESKTOP,  /* parent window          */
                       WC_LISTBOX,    /* class name             */
                       "new button",  /* window text            */
                       flStyle,       /* window style           */
                       0, 0,          /* position (x,y)         */
                       200, 100,      /* size (width,height)    */
                       NULLHANDLE,    /* owner window           */
                       HWND_TOP,      /* sibling window         */
                       ButtonId,      /* window id              */
                       NULL,          /* control data           */
                       NULL);         /* presentation parms     */
```

WinDdeInitiate –
Dynamic Data Exchange Initiate (NLS)

```
#define INCL_WINDDE   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinDdeInitiate (HWND hwndClient, PSZ pszAppName, PSZ pszTopicName,
                     PCONVCONTEXT pContext)
```

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.

Parameters

hwndClient (HWND) – input
Client's window handle.

This window will typically not be visible.

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

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 —
Dynamic Data Exchange Post Message (NLS)

```
#define INCL_WINDDE   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinDdePostMsg (HWND hwndTo, HWND hwndFrom, USHORT usMsgId,
                    PDDESTRUCT pData, ULONG ulOptions)
```

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.

Parameters

hwndTo (HWND) — input
Window handle of target.

hwndFrom (HWND) — input
Window handle of originator.

usMsgId (USHORT) — input
Message identifier.

Identifies the message to be posted.

The following messages are valid:

WM_DDE_ACK
WM_DDE_ADVISE
WM_DDE_DATA
WM_DDE_EXECUTE
WM_DDE_POKE
WM_DDE_REQUEST
WM_DDE_TERMINATE
WM_DDE_UNADVISE.

pData (PDDESTRUCT) — input
DDE structure passed.

Points to a DDESTRUCT structure. This parameter is always sent to an application as a 16-bit address. For a 32-bit application, a conversion to a flat address is required. Use public macros DOSFLATTOSEL or DOSSELTOFLAT to convert 0:32 to 16:16 or vice versa.

ulOptions (ULONG) — input
Options.

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 *usContext* 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 Indicates that the receiver is not to free the memory on behalf of the originator.

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

WinDdePostMsg — Dynamic Data Exchange Post Message (NLS)

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 – Dynamic Data Exchange Respond (NLS)

```
#define INCL_WINDDE   /* Or use INCL_WIN or INCL_PM */
```

```
MRESULT WinDdeRespond (HWND hwndClient, HWND hwndServer, PSZ pszAppName,
                       PSZ pszTopicName, PCONVCONTEXT pContext)
```

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.

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.

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.

pContext (PCONVCONTEXT) – input
Conversation context.

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.

WinDdeRespond – Dynamic Data Exchange Respond (NLS)

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 – Default Dialog Procedure

#define INCL_WINDIALOGS /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

```
MRESULT WinDefDlgProc (HWND hwndDlg, ULONG ulMsgId, MPARAM mpParam1,
                       MPARAM mpParam2)
```

This function invokes the default dialog procedure with *hwndDlg*, *ulMsgid*, *mpParam1*, and *mpParam2*.

Parameters

hwndDlg (HWND) – input
Dialog-window handle.

ulMsgId (ULONG) – input
Message identity.

mpParam1 (MPARAM) – input
Parameter 1.

mpParam2 (MPARAM) – input
Parameter 2.

Returns

Message-return data.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.

Remarks

The action taken by the default dialog procedure is such that the values passed in *mpParam1* and *mpParam2*, and the values returned in *mresReply* are defined for each *ulMsgid*.

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

WinDefDlgProc — Default Dialog Procedure

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 mp1,
                            MPARAM mp2)
{
    switch (ulMessage) {

        /*
         * Process whatever messages you want here and send the rest
         * to WinDefDlgProc.
         */

        default:
            return (WinDefDlgProc(hwnd, ulMessage, mp1, mp2));
    }
}
```

#define INCL_WINSTDFILE

MRESULT WinDefFileDlgProc (HWND hwndDlg, ULONG ulMsgid, MPARAM mpParam1, MPARAM mpParam2)

This function is the default dialog procedure for the file dialog.

Parameters

hwndDlg (HWND) – input
Dialog-window handle.

ulMsgid (ULONG) – input
Message identity.

mpParam1 (MPARAM) – input
Parameter 1.

mpParam2 (MPARAM) – input
Parameter 2.

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  hwndDlg, ULONG  ulMsgid, MPARAM Param1,
                      MPARAM Param2)
{
switch(ulMsgid)
   {

   /*************************************************************/
   /* Process user-supported messages                           */
   /*************************************************************/
   .
   .
   .
   default:
      return (WinDefFileDlgProc(hwndDlg, ulMsgid, Param1, Param2));
   }
}
```

WinDefFontDlgProc — Standard Font Dialog Default Procedure

#define INCL_WINSTDFONT

MRESULT WinDefFontDlgProc (HWND hwndDlg, ULONG ulMsgid, MPARAM mpParam1, MPARAM mpParam2)

This function is the default dialog procedure for the font dialog.

Parameters

hwndDlg (HWND) — input
Dialog-window handle.

ulMsgid (ULONG) — input
Message identity.

mpParam1 (MPARAM) — input
Parameter 1.

mpParam2 (MPARAM) — input
Parameter 2.

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  hwndDlg, ULONG  ulMsgid, MPARAM Param1,
                      MPARAM Param2)
{
switch(ulMsgid)
   {

   /*************************************************************/
   /* Process user-supported messages                           */
   /*************************************************************/
   .
   .
   .
   default:
      return (WinDefFontDlgProc(hwndDlg, ulMsgid, Param1, Param2));
   }
}
```

WinDefWindowProc — Default Window Procedure

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

MRESULT WinDefWindowProc (HWND hwnd, ULONG ulMsgid, MPARAM mpParam1,
                          MPARAM mpParam2)
```

This function invokes the default window procedure.

Parameters

hwnd (HWND) — input
Window handle.

ulMsgid (ULONG) — input
Message identity.

mpParam1 (MPARAM) — input
Parameter 1.

mpParam2 (MPARAM) — input
Parameter 2.

Returns

Message-return data.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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 Chapter 11, "Introduction to Message Processing" on page 11-1.)

Related Functions

- WinCalcFrameRect
- WinCreateFrameControls
- WinCreateStdWindow
- WinCreateWindow
- WinDestroyWindow
- WinQueryClassInfo
- WinQueryClassName
- WinRegisterClass
- WinSubclassWindow

WinDefWindowProc — Default Window Procedure

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 mp1,
                       MPARAM mp2)
{
switch(ulMsgid)
   {
   /*
   .
   . process user supported messages
   .
   */

   default:
      return (WinDefWindowProc(hwnd, ulMsgid, mp1, mp2));
   }
}
```

WinDeleteAtom – Delete Atom

```
#define INCL_WINATOM   /* Or use INCL_WIN or INCL_PM */
```

ATOM WinDeleteAtom (HATOMTBL hatomtblAtomTbl, ATOM atom)

This function deletes an atom from an atom table.

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

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	An invalid atom-table handle was specified.
PMERR_INVALID_ATOM	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

WinDeleteAtom — Delete Atom

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 – Delete Listbox Item

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

SHORT WinDeleteLboxItem (HWND hwndLbox, SHORT sIndex)

This macro deletes the indexed item from the List Box. It returns the number of items left.

Parameters

hwndLbox (HWND) – input
Listbox handle.

sIndex (SHORT) – input
Index of the listbox item.

Returns

Number of items left.

Remarks

This macro expands to:

```
#define WinDeleteLboxItem(hwndLbox, sIndex)
  ((SHORT)WinSendMsg(hwndLbox,
                     LM_DELETEITEM,
                     MPFROMSHORT(sIndex),
                     (MPARAM)NULL))
```

This function requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- LM_DELETEITEM

WinDeleteLboxItem – Delete Listbox Item

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>

SHORT  sIndex;          /* selected item index                  */
SHORT  sLeft;           /* items left after delete              */
HWND  hwndLbox;         /* list box window handle               */
MPARAM  mp1;            /* Parameter 1 (rectl structure)        */
MPARAM  mp2;            /* Parameter 2 (frame boolean)          */

case WM_CONTROL:
     /* switch on control code */
     switch(SHORT2FROMMP(mp1))
        {
        case LN_SELECT:
             hwndLbox = HWNDFROMMP(mp2);

             /* query index of selected item */
             sIndex = WinQueryLboxSelectedItem(hwndLbox);

             /* delete selected listbox item */
             sLeft = WinDeleteLboxItem(hwndLbox, sIndex);
             break;
        }
```

#define INCL_WINLOAD /* Or use INCL_WIN or INCL_PM */

BOOL WinDeleteLibrary (HAB hab, HLIB hlibLibhandle)

This function deletes the library *hlibLibhandle*, which is previously loaded by the WinLoadLibrary function.

Parameters

hab (HAB) – input
Anchor-block handle.

hlibLibhandle (HLIB) – input
Library handle to be deleted.

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 — Delete Procedure

```
#define INCL_WINLOAD  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinDeleteProcedure (HAB hab, PFNWP pwndproc)

This function deletes the window or dialog procedure that was previously loaded using the WinLoadProcedure function.

Parameters

hab (HAB) — input
Anchor-block handle.

pwndproc (PFNWP) — input
Window procedure identifier to be deleted.

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 – Deregister Workplace Object Class

#define INCL_WINWORKPLACE

BOOL WinDeregisterObjectClass (PSZ pszClassName)

The WinDeregisterObjectClass function deregisters (removes) a workplace object class.

Parameters

pszClassName (PSZ) – input

A pointer to a zero-terminated string which contains the name of the object class being removed from the workplace.

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 — Destroy Accelerator Table

#define INCL_WINACCELERATORS /* Or use INCL_WIN or INCL_PM */

BOOL WinDestroyAccelTable (HACCEL haccelAccel)

This function destroys an accelerator table.

Parameters

haccelAccel (HACCEL) — input
Accelerator-table handle.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HACCEL 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 – Destroy Atom Table

```
#define INCL_WINATOM   /* Or use INCL_WIN or INCL_PM */
```

```
HATOMTBL WinDestroyAtomTable (HATOMTBL hatomtblAtomTbl)
```

This function destroys an atom table, which is created by WinCreateAtomTable

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

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 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

WinDestroyAtomTable – Destroy Atom Table

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);
```

```
#define INCL_WINCURSORS  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

BOOL WinDestroyCursor (HWND hwnd)

This function destroys the current cursor, if it belongs to the specified window.

Parameters

hwnd (HWND) – input
Window handle to which the cursor belongs.

This can be the desktop-window handle or HWND_DESKTOP.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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.

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 — Destroy Help Instance

```
#define INCL_WINHELP  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinDestroyHelpInstance (HWND hwndHelpInstance)

This function destroys the specified instance of the help manager.

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

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 — Destroy Message Queue

#define INCL_WINMESSAGEMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

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

This function destroys the message queue.

Parameters

hmq (HMQ) — input
Message-queue handle.

Returns

Queue-destroyed indicator:

TRUE Queue destroyed

FALSE Queue not destroyed.

Possible returns from WinGetLastError

PMERR_INVALID_HMQ 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

- WinCancelShutdown
- WinBroadcastMsg
- WinCreateMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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 – Destroy Workplace Object

#define INCL_WINWORKPLACE

BOOL WinDestroyObject (HOBJECT object)

The WinDestroyObject function is called to delete a workplace object.

Parameters

object (HOBJECT) – input

Handle to a workplace object.

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.

Related Functions

- WinCreateObject
- WinSetObjectData

WinDestroyPointer — Destroy Pointer

```
#define INCL_WINPOINTERS   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinDestroyPointer (HPOINTER hptrPointer)

This function destroys a pointer or icon.

Parameters

hptrPointer (HPOINTER) — input
Handle of pointer to be destroyed.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HPTR 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
- WinSetPointer
- WinSetPointerPos
- WinShowPointer

WinDestroyPointer — Destroy Pointer

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 — Destroy Window

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

```
BOOL WinDestroyWindow (HWND hwnd)
```

This call destroys a window and its child windows.

Parameters

hwnd (HWND) — input
Window handle.

Returns

Window-destroyed indicator:

TRUE Window destroyed

FALSE Window not destroyed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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, *fSuccess* 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 the GpiCreatePS function, then it is disassociated from the window, but not destroyed. That is, this function performs the GpiAssociate function to disassociate the presentation space but does not perform the GpiDestroyPS function. If the presentation space was obtained by use of the WinGetPS function, it is released by this function; that is, this function performs the WinReleasePS function.

Messages sent by this call are:

WM_DESTROY Always sent to the window being destroyed after the window has been hidden on the device, but before its 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.

WinDestroyWindow — Destroy Window

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 – Dismiss Dialog

```
#define INCL_WINDIALOGS   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

BOOL WinDismissDlg (HWND hwndDlg, ULONG ulResult)

This function hides the modeless dialog window, or destroys the modal dialog window, and causes the WinProcessDlg or WinDlgBox functions to return.

Parameters

hwndDlg (HWND) – input
Dialog-window handle.

ulResult (ULONG) – input
Reply value.

Returned to the caller of the WinProcessDlg or WinDlgBox functions.

Returns

Dialog-dismissed indicator:

TRUE Dialog successfully dismissed

FALSE Dialog not successfully dismissed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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

WinDismissDlg — Dismiss Dialog

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   mp1;
HWND     hwnd;

case WM_COMMAND:
    switch (SHORT1FROMMP(mp1)) {
        case ID_ENTER:    /* OK button selected  */
            WinDismissDlg(hwnd, TRUE);
            return (0L);

        case ID_CANCEL:   /* Cancel button selected */
            WinDismissDlg(hwnd, FALSE);
            return (0L);
```

WinDispatchMsg — Dispatch Message

```
#define INCL_WINMESSAGEMGR   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

MRESULT WinDispatchMsg (HAB hab, PQMSG pqmsgMsg)

This function invokes a window procedure.

Parameters

hab (HAB) — input
Anchor-block handle.

pqmsgMsg (PQMSG) — input
Message structure.

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 Chapter 11, "Introduction to Message Processing" on page 11-1.

Related Functions

- WinCancelShutdown
- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

WinDispatchMsg — Dispatch Message

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 — Load and Process Modal Dialog

#define INCL_WINDIALOGS /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

```
ULONG WinDlgBox (HWND hwndParent, HWND hwndOwner, PFNWP pDlgProc,
                 HMODULE Resource, ULONG ulDlgid, PVOID pCreateParams)
```

This function loads and processes a modal dialog window and returns the result value established by the WinDismissDlg call.

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.

pDlgProc (PFNWP) — input
Dialog procedure for the created dialog window.

Resource (HMODULE) — input
Resource identity containing the dialog template.

NULLHANDLE Use the application's .EXE file.

Other Module handle returned from the DosLoadModule or DosGetModHandle call.

ulDlgId (ULONG) — input
Dialog-template identity within the resource file.

It is also used as the identity of the created dialog window.

pCreateParams (PVOID) — input
Application-defined data area.

This is passed to the dialog procedure in the WM_INITDLG message.

Returns

Reply value.

Value established by the WinDismissDlg call or DID_ERROR if an error occurs.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_INTEGER_ATOM	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND	The specified atom name is not in the atom table.
PMERR_RESOURCE_NOT_FOUND	The specified resource identity could not be found.

WinDlgBox — Load and Process Modal Dialog

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 — Set Capture" on page 8-442).

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.

Note: 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.

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 — Draw Bit Map

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinDrawBitmap (HPS hps, HBITMAP hbm, PRECTL prclSrc, PPOINTL pptlDest,
                    LONG lForeColor, LONG lBackColor, ULONG flRgf)
```

This function draws a bit map using the current image colors and mixes.

Parameters

hps (HPS) — input
Handle of presentation space in which the bit map is drawn.

hbm (HBITMAP) — input
Bit-map handle.

prclSrc (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.

pptlDest (PPOINTL) — input
Bit-map destination.

The bottom left corner of the bit-map destination is specified in device coordinates.

lForeColor (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 *lForeColor*. Ignored if DBM_IMAGEATTRS is specified.

lBackColor (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 *lBackColor*. Ignored if DBM_IMAGEATTRS is specified.

flRgf (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 *pptlDest* points to a RECT data structure, representing a rectangle in the destination presentation space, to which the bit map should be stretched or compressed. If compression is required, some rows and columns of the bit map are eliminated.

DBM_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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_FLAG	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	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). The presentation space handle can be to either a micro-presentation space or a normal presentation space (see GpiCreatePS).

If *hbm* refers to a color bit map, no color conversion is performed.

The current position in the presentation space is not changed by this function.

Related Functions

- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinFillRect
- WinGetSysBitmap
- WinInvertRect
- WinQueryPresParam
- WinRemovePresParam
- WinScrollWindow
- WinSetPresParam

WinDrawBitmap — Draw Bit Map

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  mp1;            /* Parameter 1 (menu item id)           */
MPARAM  mp2;            /* Parameter 2 (menu handle)            */

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     */
        }
```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM */

```
BOOL WinDrawBorder (HPS hps, PRECTL prclRectangle, LONG lVertSideWidth,
                    LONG lHorizSideWidth, LONG lBorderColor, LONG lInteriorColor,
                    ULONG flCmd)
```

This function draws the borders and interior of a rectangle.

Parameters

hps (HPS) – input
Presentation-space handle.

prclRectangle (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.

lVertSideWidth (LONG) – input
Width of border rectangle vertical sides.

This is the width of the left and right sides in device coordinates.

lHorizSideWidth (LONG) – input
Width of border rectangle horizontal sides.

This is the width of the top and bottom sides in device coordinates.

lBorderColor (LONG) – input
Color of edge of border.

Not used if DB_AREAATTRS is specified.

lInteriorColor (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.

DB_ROP
A group of flags that specify the mix to be used, for both the border and the interior.

DB_PATCOPY
Use the ROP_PATCOPY raster operation (see GpiBitBlt). This is a copy of the pattern to the destination.

DB_PATINVERT
Use the ROP_PATINVERT raster operation (see GpiBitBlt). This is an exclusive-OR of the pattern with the destination.

DB_DESTINVERT
Use the ROP_DESTINVERT raster operation (see GpiBitBlt). This inverts the destination.

DB_AREAMIXMODE
Map the current area foreground mix attribute into a Bitblt raster operation (see GpiBitBlt). The area background mix mode is ignored.

DB_INTERIOR
The area contained within the given rectangle, and not included within the borders (as given by *lVertSideWidth* and *lHorizSideWidth*), is drawn.

DB_AREAATTRS

- If this is specified:

  For any border, the pattern used is the pattern as currently defined in the area attribute.

  For any interior, the pattern used is the same as if a GpiSetAttrs function 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 a GpiSetAttrs function for the area attributes is made with a foreground color of *lBorderColor*, and a background color of *lInteriorColor*.

  For any interior, the pattern used is the same as if a GpiSetAttrs function for the area attributes is made with a foreground color of *lInteriorColor*, and a background color of *lBorderColor*.

DB_STANDARD
lVertSideWidth and *lHorizSideWidth* 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 *lInteriorColor*.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_FLAG	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	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). DB_DESTINVERT inverts the destination.

If DB_AREAMIXMODE is given, the foreground mix mode from the area attribute is mapped into an equivalent ROP_ value (see GpiBitBlt). The area background mix mode is ignored.

Either or both *lVertSideWidth* or *lHorizSideWidth* 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 — Draw Pointer

```
#define INCL_WINPOINTERS   /* Or use INCL_WIN or INCL_PM */

BOOL WinDrawPointer (HPS hps, LONG lx, LONG ly, HPOINTER hptrPointer,
                     ULONG ulHalftone)
```

This function draws a pointer in the passed *hps* at the passed coordinates [*lx*, *ly*].

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).

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.

This is equivalent to a bit-map handle and is returned from calls such as the GpiLoadBitmap function.

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.

Returns

Success indicator:

TRUE	Successful completion
FALSE	Function failed.

Possible returns from WinGetLastError

PMERR_INVALID_HPTR	An invalid pointer handle was specified.
PMERR_INVALID_FLAG	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

- WinDrawBitmap
- WinDrawBorder
- WinDrawText
- WinFillRect
- WinGetSysBitmap
- WinInvertRect
- WinQueryPresParam
- WinRemovePresParam
- WinScrollWindow
- WinSetPresParam
- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinSetPointer
- WinSetPointerPos
- WinShowPointer

Example Code

This example draw 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 — Draw Text

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

```
LONG WinDrawText (HPS hps, LONG lCount, PCH pchText, PRECTL prclRectangle,
                  LONG lForeColor, LONG lBackColor, ULONG flCmd)
```

This function draws a single line of formatted text into a specified rectangle.

Parameters

hps (HPS) — input
Presentation-space handle.

lCount (LONG) — input
Count of the number of characters in the string:

–1 The string is null-terminated and its length is to be calculated by this function.

Other Count of the number of characters in the string.

pchText (PCH) — input
Character string to be drawn.

A carriage-return or line-feed character terminates the line, even if the line is less than *lCount*.

prclRectangle (PRECTL) — input/output
Text rectangle.

Rectangle within which the text is to be formatted, in world coordinates. Points on the boundary of this rectangle are deemed to be inside the rectangle.

The return value is only of interest in the instance where DT_QUERYEXTENT is set in *flCmd*.

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.

lForeColor (LONG) — input
Foreground color.

Ignored if DT_TEXTATTRS is specified.

lBackColor (LONG) — input
Background color.

The background is drawn with the current background mix. The default is BM_LEAVEALONE, that is, *lBackColor* 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	No drawing is performed. *prclRectangle* 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 *lForeColor* and *lBackColor* are ignored.
DT_ERASERECT	If this is specified, the rectangle defined by *prclRectangle* is erased before drawing the text. Otherwise, the background of the characters themselves can be erased if the character background mix (see GpiSetAttrs and GpiSetBackMix) is set to BM_OVERPAINT.
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.

WinDrawText – Draw Text

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	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).

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 – Empty Clipboard

#define INCL_WINCLIPBOARD /* Or use INCL_WIN or INCL_PM */

BOOL WinEmptyClipbrd (HAB hab)

This function empties the clipboard, removing and freeing all handles to data that is in the clipboard.

Parameters

hab (HAB) – input
Anchor-block handle.

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);
```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM */

BOOL WinEnableControl (HWND hwndDlg, USHORT usId, BOOL fEnable)

This macro sets the enable state of the item in the dialog template to the enable flag.

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

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 — Enable Menu Item

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinEnableMenuItem (HWND hwndMenu, USHORT usId, BOOL fEnable)

This macro sets the state of the specified menu item to the enable flag.

Parameters

hwndMenu (HWND) — input
Menu window handle.

usId (USHORT) — input
Item identifier.

fEnable (BOOL) — input
Enable flag.

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  mp1;            /* Parameter 1 (rectl structure)        */
MPARAM  mp2;            /* Parameter 2 (frame boolean)          */
USHORT  usItemId;       /* menu item id                         */
HWND    hwndMenu;       /* menu handle                          */

case WM_INITMENU:
     usItemId = SHORT1FROMMP(mp1);
     hwndMenu = HWNDFROMMP(mp2);

     /* enable menu item */
     fResult = WinEnableMenuItem(hwndMenu, usItemId, TRUE);
```

WinEnablePhysInput – Enable Physical Input

#define INCL_WININPUT /* Or use INCL_WIN or INCL_PM */

BOOL WinEnablePhysInput (HWND hwndDeskTop, BOOL fNewInputState)

This function enables or disables queuing of physical input (keyboard or mouse).

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle:

HWND_DESKTOP The desktop-window handle

Other Specified desktop-window handle.

fNewInputState (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

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 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 — Set Window Enabled State

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinEnableWindow (HWND hwnd, BOOL fNewEnabled)

This function sets the window enabled state.

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

Window enabled indicator:

TRUE Window enabled state successfully updated

FALSE Window enabled state not successfully updated.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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 pushbutton 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

WinEnableWindow — Set Window Enabled State

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 — Enable Window Update

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */

BOOL WinEnableWindowUpdate (HWND hwnd, BOOL fNewVisibility)
```

This function sets the window visibility state for subsequent drawing.

Parameters

hwnd (HWND) — input
Window handle.

fNewVisibility (BOOL) — input
New visibility state:

TRUE Set window state visible

FALSE Set window state invisible.

Returns

Visibility-changed indicator:

TRUE Window visibility successfully changed

FALSE Window visibility not successfully changed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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 the window. The window can be redrawn by use of the WinShowWindow function.

WS_VISIBLE is set to *fNewVisibility* without causing redrawing. This implies that if the window was previously visible, it remains visible on the device when WS_VISIBLE is reset, and conversely, if the window was previously invisible, it is not shown when WS_VISIBLE is set. If *fNewVisibility* is set to TRUE, any subsequent drawing into the window is visible. If *fNewVisibility* is set to FALSE, any subsequent drawing into the window is not visible.

If the value of the WS_VISIBLE style bit has been changed, the WM_SHOW message is sent to the window of *hwnd* before the call returns. This function is usually used to disable drawing before making a series of changes to a window to prevent unnecessary drawing. To show a window and ensure that it is redrawn after calling the WinEnableWindowUpdate function with *fNewVisibility* set to FALSE, use the WinShowWindow function with *fNewVisibility* set to TRUE.

Any alteration to the appearance of a window disabled for window update is not presented. Therefore, the application must ensure that the window is redrawn. In particular, if a window is destroyed while in this state its image is not removed from the display. After window updating is reenabled, the application should ensure that the window gets totally invalidated so that it repaints.

WinEnableWindowUpdate – Enable Window Update

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_ERASEWINDOW
- 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 — End Window Enumeration

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */

BOOL WinEndEnumWindows (HENUM henum)
```

This function ends the enumeration process for a specified enumeration.

Parameters

henum (HENUM) — input
Enumeration handle.

Returned by previous call to the WinBeginEnumWindows call.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HENUM	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

WinEndEnumWindows — End Window Enumeration

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 – End Paint

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

BOOL WinEndPaint (HPS hps)

This function indicates that the redrawing of a window is complete, generally as part of the processing of a WM_PAINT message.

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

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

WinEndPaint — End Paint

- 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 — Enumerate Clipboard Formats

```
#define INCL_WINCLIPBOARD   /* Or use INCL_WIN or INCL_PM */
```

ULONG WinEnumClipbrdFmts (HAB hab, ULONG ulPrev)

This function enumerates the list of clipboard data formats available in the clipboard.

Parameters

hab (HAB) — input
Anchor-block handle.

ulPrev (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

Next clipboard-data format index:

0 Enumeration is complete; that is, there are no more clipboard formats available.

Other Index of the next available clipboard-data format in the clipboard.

Remarks

The clipboard should be open before this function is used.

Related Functions

- WinCloseClipbrd
- WinEmptyClipbrd
- WinOpenClipbrd
- WinQueryClipbrdData
- WinQueryClipbrdFmtInfo
- WinQueryClipbrdOwner
- WinQueryClipbrdViewer
- WinSetClipbrdData
- WinSetClipbrdOwner
- WinSetClipbrdViewer

WinEnumClipbrdFmts — Enumerate Clipboard Formats

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 – Enumerate Dialog Item

```
#define INCL_WINDIALOGS   /* Or use INCL_WIN or INCL_PM */

HWND WinEnumDlgItem (HWND hwndDlg, HWND hwnd, ULONG ulCode)
```

This function returns the window handle of a dialog item within a dialog window.

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 *ulCode* is EDI_FIRSTTABITEM or EDI_LASTTABITEM.

ulCode (ULONG) – input
Item-type code.

Determines the type of dialog item to return.

EDI_PREVTABITEM	Previous item with style WS_TABSTOP. Wraps around to end of dialog item list when beginning is reached.
EDI_NEXTTABITEM	Next item with style WS_TABSTOP. Wraps around to beginning of dialog item list when end is reached.
EDI_FIRSTTABITEM	First item in dialog with style WS_TABSTOP. *hwnd* is ignored.
EDI_LASTTABITEM	Last item in dialog with style WS_TABSTOP. *hwnd* is ignored.
EDI_PREVGROUPITEM	Previous item in the same group. Wraps around to end of group when the start of the group is reached. For information on the WS_GROUP style, see "Window Styles" on page 12-2.
EDI_NEXTGROUPITEM	Next item in the same group. Wraps around to beginning of group when the end of the group is reached.
EDI_FIRSTGROUPITEM	First item in the same group.
EDI_LASTGROUPITEM	Last item in the same group.

Returns

Item-window handle.

As dictated by *ulCode*.

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	An invalid window handle was specified.

WinEnumDlgItem – Enumerate Dialog Item

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 – Enumerate Object Classes

#define INCL_WINWORKPLACE

BOOL WinEnumObjectClasses (POBJCLASS pObjClass, PULONG pSize)

The WinEnumObjectClasses function will return a list of all workplace object classes that have been registered.

Parameters

pObjClass (POBJCLASS) – input

A pointer to a buffer to be filled with information about the registered workplace object classes.

pSize (PULONG) – input/output

Length of the *pObjClass* buffer in bytes. If *pObjClass* is NULL, the actual size of *pObjClass* is returned in *pSize*

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 — Equal Rectangle

```
#define INCL_WINRECTANGLES   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinEqualRect (HAB hab, PRECTL prclRect1, PRECTL prclRect2)

This function compares two rectangles for equality.

Parameters

hab (HAB) — input
Anchor-block handle.

prclRect1 (PRECTL) — input
First rectangle.

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.

prclRect2 (PRECTL) — input
Second rectangle.

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.

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 – Exclude Update Region

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

LONG WinExcludeUpdateRegion (HPS hps, HWND hwnd)

This function subtracts the update region (invalid region) of a window from the clipping region of a presentation space.

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

Complexity value.

This indicates the resulting form of the clipping area. The values and meanings of this parameter are defined in the GpiCombineRegion function.

Complexity of resulting region/error indicator:

EXRGN_NULL	Null Region
EXRGN_RECT	Rectangle region
EXRGN_COMPLEX	Complex region
EXRGN_ERROR	Error.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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 – Standard File Dialog

#define INCL_WINSTDFILE

HWND WinFileDlg (HWND hwndParent, HWND hwndOwner, PFILEDLG pfdFiledlg)

This function creates and displays the file dialog and returns the user's selection or selections.

Parameters

hwndParent (HWND) – input
Parent-window handle.

Parent-window handle of the created dialog window.

HWND_DESKTOP The desktop window.

Other Specified window.

hwndOwner (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 i the description of the WinLoadDlg function.

pfdFiledlg (PFILEDLG) – input
Pointer.

Pointer to a FILEDLG structure.

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 *pfdFiledlg* 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 pushbutton pressed to dismiss the dialog, DID_OK or DID_CANCEL, unless the application supplied additional pushbuttons 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 X'0000' through X'0FFF' 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 pfdFiledlg;      /* 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                                          */
/********************************************************************/
pfdFiledlg.cbSize = sizeof(FILEDLG); /* Size of structure           */
pfdFiledlg.fl = FDS_HELPBUTTON | FDS_CENTER | FDS_OPEN_DIALOG;
                                     /* FDS_* flags                 */
pfdFiledlg.pszTitle = pszTitle;      /* Dialog title string         */
strcpy(pfdFiledlg.szFullFile, pszFullFile); /* Initial path,        */
                                            /* file name, or        */
                                            /* file filter          */

/********************************************************************/
/* Display the dialog and get the file                              */
/********************************************************************/

hwndDlg = WinFileDlg(HWND_DESKTOP, hwndMain, &pfdFiledlg);

if (hwndDlg && (pfdFiledlg.lReturn == DID_OK))
{
   /*****************************************************************/
   /* Upon successful return of a file, open it for reading and     */
   /* further processing                                            */
   /*****************************************************************/
}
```

WinFillRect – Fill Rectangle

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinFillRect (HPS hps, PRECTL prclRect, LONG lColor)
```

This function draws a filled rectangular area.

Parameters

hps (HPS) – input
Presentation-space handle.

This can be either a micro-presentation space or a normal presentation space.

prclRect (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 –32 768 through 32 767. 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 the GpiCreateLogColorTable and GpiSetColor functions.)

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

- WinDrawBitmap
- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinGetSysBitmap
- WinInvertRect
- WinQueryPresParam
- WinRemovePresParam
- WinScrollWindow
- WinSetPresParam
- WinCopyRect
- WinEqualRect
- WinInflateRect
- WinIntersectRect
- WinIsRectEmpty
- WinOffsetRect
- WinPtInRect
- 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                  */
RECTL 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 — Find Atom

```
#define INCL_WINATOM   /* Or use INCL_WIN or INCL_PM */
```

ATOM WinFindAtom (HATOMTBL hatomtblAtomTbl, PSZ pszAtomName)

This function finds an atom in the atom table.

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 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	An invalid atom-table handle was specified.
PMERR_INVALID_INTEGER_ATOM	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND	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 — Flash Window

```
#define INCL_WINFRAMEMGR  /* Or use INCL_WIN or INCL_PM */

BOOL WinFlashWindow (HWND hwnd, BOOL fFlash)
```

This function starts or stops a window flashing.

Parameters

hwnd (HWND) — input
Handle of window to be flashed.

fFlash (BOOL) — input
Start-flashing indicator:

TRUE Start window flashing

FALSE Stop window flashing.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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 – Change Focus Window

```
#define INCL_WININPUT   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinFocusChange (HWND hwndDeskTop, HWND hwndNewFocus,
                     ULONG flFocusChange)
```

This function changes the focus window.

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.

FC_SETACTIVEFOCUS
Set the focus to the child window that previously had the focus of the first window in the parentage of *hwndNewFocus*, which has the CS_FRAME style.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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

WinFocusChange — Change Focus Window

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  mp1;            /* Parameter 1 (select boolean)         */

case WM_SETSELECTION:
     /* if window is being selected, change focus to the window */
     if (SHORTFROMMP(mp1))
        {
        if ((hwndNewFocus =
             WinQueryFocus(HWND_DESKTOP)) != 0L)
           fSuccess = WinFocusChange(HWND_DESKTOP, hwndNewFocus,
                                     0L);
        }
```

WinFontDlg — Standard Font Dialog

#define INCL_WINSTDFONT

HWND WinFontDlg (HWND hwndParent, HWND hwndOwner, PFONTDLG pfntdFontdlg)

This dialog allows the user to select a font.

Parameters

hwndParent (HWND) — input
Parent-window handle.

Parent-window handle of the created dialog window.

HWND_DESKTOP The desktop window.

Other Specified window.

hwndOwner (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.

pfntdFontdlg (PFONTDLG) — input
Pointer.

Pointer to an initialized FONTDLG structure.

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 *pfntdFontdlg* parameter is required and the FONTDLG structure must be properly initialized.

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 pushbutton pressed to dismiss the dialog, DID_OK or DID_CANCEL, unless the application supplied additional pushbuttons 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 X'0000' through X'0FFF' 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.

WinFontDlg — Standard 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;
pfdFontdlg.usFamilyBufLen = strlen(szFamilyname);
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 — Free Error Information

#define INCL_WINERRORS /* Or use INCL_WIN or INCL_PM */

BOOL WinFreeErrorInfo (PERRINFO perriErrorInfo)

This function releases memory allocated for an error-information block.

Parameters

perriErrorInfo (PERRINFO) — input
Error-information block whose memory is to be released.

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 – Free Standard File Dialog File List

#define INCL_WINSTDFILE

BOOL WinFreeFileDlgList (PAPSZ papszFQFilename)

This function frees the storage allocated by the file dialog when the FDS_MULTIPLESEL dialog flag is set.

Parameters

papszFQFilename (PAPSZ) – input
Pointer.

Pointer to a table of pointers of fully-qualified file names returned by the dialog.

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 – Free File Icon

```
#define INCL_WINWORKPLACE
```

BOOL WinFreeFileIcon (HPOINTER hptr)

The WinFreeFileIcon function will free the pointer to an icon allocated by WinLoadFileIcon.

Parameters

hptr (HPOINTER) – input

A pointer to an icon loaded by WinLoadFileIcon.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Related Functions

- WinSetFileIcon
- WinLoadFileIcon

WinGetClipPS – Get Clipped Presentation Space

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

```
HPS WinGetClipPS (HWND hwnd, HWND hwndClipWindow, ULONG ulClipflags)
```

This function obtains a clipped cache presentation space.

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

Presentation-space handle that can be used for drawing.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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.

WinGetClipPS –
Get Clipped Presentation Space

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 — Get Current Time

#define INCL_WINTIMER /* Or use INCL_WIN or INCL_PM */

ULONG WinGetCurrentTime (HAB hab)

This function returns the current time.

Parameters

hab (HAB) — input
Anchor-block handle.

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 — Get Dialog Message

```
#define INCL_WINDIALOGS   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
BOOL WinGetDlgMsg (HWND hwndDlg, PQMSG pqmsgmsg)
```

This function obtains a message from the application's queue associated with the specified dialog.

Parameters

hwndDlg (HWND) — input
Dialog-window handle.

pqmsgmsg (PQMSG) — output
Message structure.

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 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

- WinCreateDlg
- WinDefDlgProc
- WinDismissDlg
- WinDlgBox
- WinLoadDlg
- WinProcessDlg
- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Related Messages

- WM_QUIT

WinGetDlgMsg — Get Dialog Message

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                        */
QMSG    qmsg;           /* 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));
   }
}
```

```
#define INCL_WINERRORS  /* Or use INCL_WIN or INCL_PM */
```

PERRINFO WinGetErrorInfo (HAB hab)

This function returns detailed error information.

Parameters

hab (HAB) – input
Anchor-block handle.

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 – Get Key State

```
#define INCL_WININPUT   /* Or use INCL_WIN or INCL_PM */
```

```
LONG WinGetKeyState (HWND hwndDeskTop, LONG lVk)
```

This function returns the state of the key at the time that the last message obtained from the queue was posted.

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle:

HWND_DESKTOP The desktop-window handle

Other Specified desktop-window handle.

lVk (LONG) – input
Virtual key value.

Contains the virtual key value in the low-order byte, and zero in the high-order byte.

Returns

Key state.

This value is the OR combination of the following bits:

X'0001' The key has been pressed an odd number of times since the system has been started.

X'8000' The key is down.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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.

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   lVk;             /* virtual key value                    */

case WM_TIMER:
     /* get key state of mouse button 1 */
     lVk = VK_BUTTON1;
     lKeyState = WinGetKeyState(HWND_DESKTOP, lVk);

     /* emit high pitched beep if mouse button 1 is depressed
        when timer message occurred; otherwise, emit low pitched
        beep */
     if (lVk & 0x8000)
        DosBeep(1000,100L);
     else
        DosBeep(200,100L);
```

WinGetLastError — Get Last Error

#define INCL_WINERRORS /* Or use INCL_WIN or INCL_PM */

ERRORID WinGetLastError (HAB hab)

This function returns the error state set by the failure of a Presentation Manager function.

Parameters

hab (HAB) — input
Anchor-block handle.

Returns

Last-error state.

Remarks

Returns the last nonzero error code, and sets the error code to zero.

The current error state is reset to zero.

In multiple thread applications where there are multiple anchor blocks, errors are stored in the anchor block created by the WinInitialize 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.

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 — Get Maximum Position

```
#define INCL_WINFRAMEMGR   /* Or use INCL_WIN or INCL_PM */

BOOL WinGetMaxPosition (HWND hwnd, PSWP pSwp)
```

The WinGetMaxPosition function fills an SWP structure with the maximized-window size and position.

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

Success indicator:

TRUE Successful completion.

FALSE Error occurred.

WinGetMaxPosition — Get Maximum Position

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  mp1;            /* Parameter 1 (window position)        */
PSWP   pSwp;            /* Set window position structure        */

case WM_MINMAXFRAME:
     pSwp = (PSWP)PVOIDFROMMP(mp1);

     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 — Get Minimum Position

```
#define INCL_WINFRAMEMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinGetMinPosition (HWND hwnd, PSWP pSwp, PPOINTL pptlPoint)

This function returns the position to which a window is minimized.

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.

pptlPoint (PPOINTL) — input
Preferred position:

NULL System is to choose the position

Other System is to choose the position nearest to the specified point.

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

WinGetMinPosition — Get Minimum Position

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  mp1;            /* Parameter 1 (window position)        */
PSWP   pSwp;            /* Set window position structure        */

case WM_MINMAXFRAME:
     pSwp = (PSWP)PVOIDFROMMP(mp1);

     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;
        }
```

#define INCL_WINMESSAGEMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

```
BOOL WinGetMsg (HAB hab, PQMSG pqmsgmsg, HWND hwndFilter, ULONG ulFirst,
                ULONG ulLast)
```

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.

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

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 An invalid window handle was specified.

Remarks

If system or queue hooks are installed, they are called before this function returns.

fResult 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.

WinGetMsg — Get Message

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

- WinCancelShutdown
- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Related Messages

- WM_CHAR
- 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                 */
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);
```

WinGetNextWindow — Get Next Window

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */

HWND WinGetNextWindow (HENUM henum)
```

This function gets the window handle of the next window in a specified enumeration list.

Parameters

henum (HENUM) — input
Enumeration handle.

Returned by previous call to the WinBeginEnumWindows call.

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 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. Window locking is not required in OS/2 release 1.2 and above.

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 – Get Physical Key State

```
#define INCL_WININPUT   /* Or use INCL_WIN or INCL_PM */
```

```
LONG WinGetPhysKeyState (HWND hwndDeskTop, LONG lScancode)
```

This function returns the physical key state.

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle:

HWND_DESKTOP The desktop-window handle

Other Specified desktop-window handle.

lScancode (LONG) – input
Hardware scan code.

Contains the scan code value in the low-order byte, and zero in the high-order byte.

Returns

Key state:

This value is the OR combination of the following bits:

X'0001' The key has been pressed an odd number of times since the system has been started.

X'0002' The key has been pressed since the last time this function was issued, or since the system has been started if this is the first time the call has been issued.

X'8000' The key is down.

Possible returns from WinGetLastError

PMERR_INVALID_HWND An invalid window handle was specified.

Remarks

This function returns information about the asynchronous (interrupt level) state of the virtual key indicated by the *lScancode* 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 = VK_CAPSLOCK;
lKeyState = WinGetPhysKeyState(HWND_DESKTOP, lScancode);

/* emit high pitched beep if caps lock is currently depressed;
   otherwise, emit low pitched beep */
if (lScancode & 0x8000)
   DosBeep(1000,100L);
else
   DosBeep(200,100L);
```

WinGetPS –
Get Presentation Space

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

HPS WinGetPS  (HWND  hwnd)
```

This function gets a cache presentation space.

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

Presentation-space handle that can be used for drawing in the window.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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 the GpiCreatePS function. 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.

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 –
Get Screen Presentation Space

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

```
HPS WinGetScreenPS (HWND hwndDeskTop)
```

This function returns a presentation space that can be used for drawing anywhere on the screen.

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle:

HWND_DESKTOP The desktop-window handle

Other Specified desktop-window handle.

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 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 — Get System Bit Map

```
#define INCL_WINPOINTERS   /* Or use INCL_WIN or INCL_PM */
```

```
HBITMAP WinGetSysBitmap (HWND hwndDeskTop, ULONG ulIndex)
```

This function returns a handle to one of the standard bit maps provided by the system.

Parameters

hwndDeskTop (HWND) — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

ulIndex (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	Pushbutton 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_CLOSEBUTTON	Hide button
SBMP_CLOSEBUTTONDEP	Hide button in depressed state
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

System bit-map handle.

NULLHANDLE	Error occurred
Other	System bit-map handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE	The value of a parameter was not within the defined valid range for that parameter.
PMERR_RESOURCE_NOT_FOUND	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 the GpiDeleteBitmap function.

Related Functions

- WinDrawBitmap
- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinFillRect
- WinInvertRect
- WinQueryPresParam
- WinRemovePresParam
- WinScrollWindow
- WinSetPresParam

WinGetSysBitmap – Get System Bit Map

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 — Inflate Rectangle

```
#define INCL_WINRECTANGLES  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinInflateRect (HAB hab, PRECTL prcIrect, LONG lcx, LONG lcy)

This function expands a rectangle.

Parameters

hab (HAB) — input
Anchor-block handle.

prcIrect (PRECTL) — input/output
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.

lcx (LONG) — input
Horizontal expansion.

lcy (LONG) — input
Vertical expansion.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This function adjusts the size of the rectangle by applying the *lcx* parameter horizontally at both vertical edges and the *lcy* parameter vertically at both horizontal edges.

The *lcx* parameter is subtracted from the left and added to the right of the rectangle, and the *lcy* parameter is subtracted from the bottom and added to the top of the rectangle.

If the values of the *lcx* and *lcy* 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

WinInflateRect – Inflate Rectangle

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 mp1;             /* Parameter 1  (x,y) point value       */

case WM_BUTTON1DBLCLK:
     ptl.x = (LONG) SHORT1FROMMP(mp1);
     ptl.y = (LONG) SHORT2FROMMP(mp1);

     if (WinPtInRect(hab, &prclRect1, &ptl))
        fSuccess = WinInflateRect(hab, &prclRect1, lcx, lcy);
```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

HAB WinInitialize (ULONG flOptions)

This function initializes the PM facilities for use by an application.

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

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

WinInitialize – Initialize

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 – In Send Message

```
#define INCL_WINMESSAGEMGR   /* Or use INCL_WIN or INCL_PM */
```

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

This function determines whether the current thread is processing a message sent by another thread.

Parameters

hab (HAB) – input
Anchor-block handle.

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.

WinInSendMsg – In Send Message

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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  mp1;            /* 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(mp1);

     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 — Insert Listbox Item

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

SHORT WinInsertLboxItem (HWND hwndLbox, SHORT sIndex, PSZ pszText)

This macro inserts text into a list box at index, index may be a LIT_ constant. This macro returns the actual index where it was inserted.

Parameters

hwndLbox (HWND) — input
List box handle.

sIndex (SHORT) — input
Index of the list box item.

pszText (PSZ) — input
Text to be inserted.

Returns

Actual index where it was inserted.

Remarks

This macro expands to:

```
#define WinInsertLboxItem(hwndLbox, sIndex, pszText)
  ((SHORT)WinSendMsg(hwndLbox,
                     LM_INSERTITEM,
                     MPFROMSHORT(sIndex),
                     MPFROMP(pszText)))
```

This function requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- LM_INSERTITEM

WinInsertLboxItem — Insert Listbox Item

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>

SHORT  sIndex;          /* inserted item index                  */
HWND  hwndLbox;         /* list box window handle               */
MPARAM  mp1;            /* 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(mp1);

    /* insert 3 items into list box */
    sIndex = WinInsertLboxItem(hwndLbox, LIT_END, pszItems[0]);
    sIndex = WinInsertLboxItem(hwndLbox, LIT_END, pszItems[1]);
    sIndex = WinInsertLboxItem(hwndLbox, LIT_END, pszItems[2]);
      .
      .
```

WinIntersectRect — Intersect Rectangle

```
#define INCL_WINRECTANGLES   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinIntersectRect (HAB hab, PRECTL prclDest, PRECTL prclRect1, PRECTL prclRect2)

This function calculates the intersection of the two source rectangles and returns the result in the destination rectangle.

Parameters

hab (HAB) — input
Anchor-block handle.

prclDest (PRECTL) — output
Intersection rectangle.

Is the intersection of *prclRect1* and *prclRect2*.

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.

prclRect1 (PRECTL) — input
First rectangle.

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.

prclRect2 (PRECTL) — input
Second rectangle.

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.

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 *prclDest*.

Related Functions

- WinCopyRect
- WinEqualRect
- WinFillRect
- WinInflateRect
- WinIsRectEmpty
- WinOffsetRect
- WinPtInRect
- WinSetRect
- WinSetRectEmpty
- WinSubtractRect
- WinUnionRect

WinIntersectRect – Intersect Rectangle

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                  */
RECTL prclRect1 = {0,0,100,100}; /* rectangle  1                */
RECTL prclRect2 = {0,0,200,200}; /* rectangle  2                */
RECTL prclDest;         /* destination rectangle                */

fSuccess = WinIntersectRect(hab, &prclDest, &prclRect1,
                            &prclRect2);
```

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinInvalidateRect (HWND hwnd, PRECTL prclPrc, BOOL fIncludeClippedChildren)

This function adds a rectangle to a window's update region.

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.

prclPrc (PRECTL) — input
Update rectangle.

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 window is to be added into the window's update region.

Other Rectangle to be added to the window's update region.

fIncludeClippedChildren (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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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,

WinInvalidateRect — Invalidate Rectangle

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_ERASEWINDOW
- 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 */
```

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinInvalidateRegion (HWND hwnd, HRGN hrgn, BOOL fIncludeClippedChildren)

This function adds a region to a window's update region.

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.

fIncludeClippedChildren (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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_HRGN_BUSY	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	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,

WinInvalidateRegion – Invalidate Region

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_ERASEWINDOW
- 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 — Invert Rectangle

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinInvertRect (HPS hps, PRECTL prclRect)

This function inverts a rectangular area.

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 –32 768 through 32 767. The data type WRECT can also be used, if supported by the language.

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

WinInvertRect — Invert Rectangle

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 mp1;             /* Parameter 1  (x,y) point value       */

case WM_BUTTON1UP:
     ptl.x = (LONG) SHORT1FROMMP(mp1);
     ptl.y = (LONG) SHORT2FROMMP(mp1);

     if (WinPtInRect(hab, &prclRect1, &ptl))
        {
        hps = WinBeginPaint(hwnd, NULLHANDLE, &prclRect1);
        fSuccess = WinInvertRect(hps, &prclRect1);
        WinEndPaint(hps);
        }
```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM */

BOOL WinIsChild (HWND hwndChild, HWND hwndParent)

This function tests if one window is a descendant of another window.

Parameters

hwndChild (HWND) – input
Child-window handle.

hwndParent (HWND) – input
Parent-window handle.

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 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 – Is Control Enabled

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinIsControlEnabled (HWND hwndDlg, USHORT usId)

This macro returns the state (enable/disable) of the specified item in the dialog template within a dialog box.

Parameters

hwndDlg (HWND) – input
Dialog window handle.

usId (USHORT) – input
Identity of the specified item.

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 — Is Menu Item Checked

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinIsMenuItemChecked (HWND hwndMenu, USHORT usId)

This macro returns the state (checked/not checked) of the identified menu item.

Parameters

hwndMenu (HWND) — input
Menu window handle.

usId (USHORT) — input
Identity of the menu item.

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 — Is Menu Item Enabled

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinIsMenuItemEnabled (HWND hwndMenu, USHORT usId)

This macro returns the state (enable/disable) of the menu item specified.

Parameters

hwndMenu (HWND) — input
Menu window handle.

usId (USHORT) — input
Identity of the menu item.

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 — Is Menu Item Valid

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinIsMenuItemValid (HWND hwndMenu, USHORT usId)

This macro returns TRUE if the specified item is a valid choice.

Parameters

hwndMenu (HWND) — input
Menu window handle.

usId (USHORT) — input
Identity of the menu item.

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 – Is Physical Input Enabled

```
#define INCL_WININPUT  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinIsPhysInputEnabled (HWND hwndDeskTop)

This function returns the status of hardware input (on/off).

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle:

HWND_DESKTOP The desktop-window handle

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 — Is Rectangle Empty

#define INCL_WINRECTANGLES /* Or use INCL_WIN or INCL_PM */

BOOL WinIsRectEmpty (HAB hab, PRECTL prclprc)

This function checks whether a rectangle is empty.

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 –32 768 through 32 767. The data type WRECT can also be used, if supported by the language.

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);
```

WinIsThreadActive – Is Thread Active

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */

BOOL WinIsThreadActive (HAB hab)
```

This function determines whether the active window belongs to the calling execution thread.

Parameters

hab (HAB) – input
Anchor-block handle of calling thread.

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 — Is Window

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinIsWindow (HAB hab, HWND hwnd)
```

This function determines if a window handle is valid.

Parameters

hab (HAB) — input
Anchor-block handle.

hwnd (HWND) — input
Window handle.

Returns

Validity indicator:

TRUE Window handle is valid

FALSE Window handle is not valid.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 — Query Window Enabled State

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinIsWindowEnabled (HWND hwnd)
```

This function returns the enabled/disabled state of a window.

Parameters

hwnd (HWND) — input
Window handle.

Returns

Enabled-state indicator:

TRUE Window is enabled

FALSE Window is not enabled.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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 – Query Window Showing

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

```
BOOL WinIsWindowShowing (HWND hwnd)
```

This function determines whether any part of the window *hwnd* is physically visible.

Parameters

hwnd (HWND) – input
Window handle.

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 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 — Query Window Visibility

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinIsWindowVisible (HWND hwnd)
```

This function returns the visibility state of a window.

Parameters

hwnd (HWND) — input
Window handle.

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 An invalid window handle was specified.

Remarks

Because *fVisible* reflects only the values of WS_VISIBLE style bits, *fVisible* may be set to TRUE even if *hwnd* is totally obscured by other windows.

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);
        }
```

WinLoadAccelTable — Load Accelerator Table

```
#define INCL_WINACCELERATORS  /* Or use INCL_WIN or INCL_PM */
```

HACCEL WinLoadAccelTable (HAB hab, HMODULE Resource, ULONG idAccelTable)

This function loads an accelerator table.

Parameters

hab (HAB) — input
Anchor-block handle.

Resource (HMODULE) — input
Resource identity containing the accelerator table.

Module handle returned by the DosLoadModule or DosGetModHandle 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.

Returns

Accelerator-table handle.

Possible returns from WinGetLastError

PMERR_RESOURCE_NOT_FOUND	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 — Load Dialog

```
#define INCL_WINDIALOGS  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
HWND WinLoadDlg (HWND hwndParent, HWND hwndOwner, PFNWP pDlgProc,
                 HMODULE Resource, ULONG idDlgId, PVOID pCreateParams)
```

This function creates a dialog window from the dialog template *idDlgid* in *Resource* and returns the dialog window handle.

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.

pDlgProc (PFNWP) — input
Dialog procedure for the created dialog window.

Resource (HMODULE) — input
Resource identity containing the dialog template.

NULLHANDLE	Use the application's .EXE file.
Other	Module handle returned from the DosLoadModule or DosGetModHandle functions.

idDlgid (ULONG) — input
Dialog-template identity within the resource file.

It is also used as the identity of the created dialog window.

pCreateParams (PVOID) — input
Application-defined data area.

This is passed to the dialog procedure in the WM_INITDLG message.

Returns

Dialog-window handle:

NULL	Dialog window not created
Other	Dialog window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_INTEGER_ATOM	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND	The specified atom name is not in the atom table.
PMERR_RESOURCE_NOT_FOUND	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

WinLoadDlg — Load Dialog

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 — Load File Icon

#define INCL_WINWORKPLACE

HPOINTER WinLoadFileIcon (PSZ pszFileName, BOOL fPrivate)

The WinLoadFileIcon function will return a pointer to an icon which is associated with the file specified by *pszFileName*.

Parameters

pszFileName (PSZ) — input

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

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.

WinLoadFileIcon – Load File Icon

Related Functions

- WinSetFileIcon
- WinFreeFileIcon

WinLoadHelpTable — Load Help Table

```
#define INCL_WINHELP   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinLoadHelpTable (HWND hwndHelpInstance, ULONG idHelpTable, HMODULE Module)

This function identifies the module handle and identity of the help table to the instance of the help manager.

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.

Module (HMODULE) — input
Handle of the module which contains the help table and help subtable resources.

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

WinLoadHelpTable — 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 — Load Library

```
#define INCL_WINLOAD   /* Or use INCL_WIN or INCL_PM */
```

HLIB WinLoadLibrary (HAB hab, PSZ pszLibname)

This function makes the library available to the application.

Parameters

hab (HAB) — input
Anchor-block handle.

pszLibname (PSZ) — input
Library name.

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 — Load Menu

```
#define INCL_WINMENUS   /* Or use INCL_WIN or INCL_PM */
```

HWND WinLoadMenu (HWND hwndOwner, HMODULE Resource, ULONG idMenuid)

This function creates a menu window from the menu template *idMenuid* from *Resource*, and returns in *hwndMenu* the window handle for the created window.

Parameters

hwndOwner (HWND) — input
Owner- and parent-window handle:

HWND_DESKTOP The desktop window

HWND_OBJECT Object window

Other Specified window.

Resource (HMODULE) — input
Resource identifier.

NULLHANDLE The resource is in the &periodEXE file of the application.

Other The module handle returned by the DosLoadModule or DosGetModHandle call.

idMenuid (ULONG) — input
Menu identifier within the resource file.

Returns

Menu-window handle.

Remarks

The menu window is created with its parent and owner set to *hwndOwner*, and with identity FID_MENU. If *hwndOwner* 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 — Load Message

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

```
LONG WinLoadMessage (HAB hab, HMODULE hmodMod, ULONG ulId, LONG lcchMax,
                     PSZ pszBuffer)
```

This function loads a message from a resource, copies the message to the specified buffer, and appends a terminating null character.

Parameters

hab (HAB) — input
Anchor-block handle.

hmodMod (HMODULE) — input
Module handle.

ulId (ULONG) — input
Message identifier.

lcchMax (LONG) — input
Specifies the size of buffer.

pszBuffer (PSZ) — input
Points to the buffer that receives the message

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 — Load Pointer

```
#define INCL_WINPOINTERS  /* Or use INCL_WIN or INCL_PM */

HPOINTER WinLoadPointer (HWND hwndDeskTop, HMODULE Resource, ULONG idPointer)
```

This function loads a pointer from a resource file into the system.

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 DosGetModHandle call referencing a dynamic-link library containing the resource.

idPointer (ULONG) — input
Identifier of the pointer to be loaded.

Returns

Pointer handle:

NULLHANDLE Error has occurred

Other Handle of loaded pointer.

Possible returns from WinGetLastError

PMERR_INVALID_HWND An invalid window handle was specified.

PMERR_RESOURCE_NOT_FOUND 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
- WinSetPointer
- WinSetPointerPos
- 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 — Load Procedure

```
#define INCL_WINLOAD  /* Or use INCL_WIN or INCL_PM */
```

PFNWP WinLoadProcedure (HAB hab, HLIB hlibLibhandle, PSZ pszProcname)

This function loads the window or dialog procedure from a specified dynamic link library.

Parameters

hab (HAB) — input
Anchor-block handle.

hlibLibhandle (HLIB) — input
Library handle.

pszProcname (PSZ) — input
Procedure name.

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);
```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM */

```
LONG WinLoadString (HAB hab, HMODULE Resource, ULONG idString, LONG lBufferMax,
                    PSZ pszBuffer)
```

This function loads a string from a resource.

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 DosGetModHandle call referencing a dynamic-link library containing the resource.

idString (ULONG) – input
String identifier.

lBufferMax (LONG) – input
Size of buffer.

pszBuffer (PSZ) – output
Buffer that is to receive the string.

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 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" on page 32-27). 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.

WinLoadString – Load String

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);
```

WinLockVisRegions — Lock Visible Regions

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinLockVisRegions (HWND hwndDeskTop, BOOL fLock)

This function locks or unlocks the visible regions of all the windows on the screen, preventing any of the visible regions from changing.

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

Success indicator.

TRUE Successful.
FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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.

WinLockVisRegions – Lock Visible Regions

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 – Lock Window Update

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinLockWindowUpdate (HWND hwndDeskTop, HWND hwndLockUpdate)

This function disables or enables output to a window and its descendants.

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

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.

WinLockWindowUpdate — Lock Window Update

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);
```

#define INCL_WINRECTANGLES /* Or use INCL_WIN or INCL_PM */

BOOL WinMakePoints (HAB hab, PWPOINT pwptppt, ULONG ccount)

This function converts points to graphics points.

Parameters

hab (HAB) – input
Anchor-block handle.

pwptppt (PWPOINT) – input/output
Points to be converted.

The data type of these points after conversion is POINTL.

ccount (ULONG) – input
Number of points to be converted.

Must be positive.

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 — Make Rectangle

#define INCL_WINRECTANGLES /* Or use INCL_WIN or INCL_PM */

BOOL WinMakeRect (HAB hab, PWRECT pwrcprc)

This function converts a rectangle to a graphics rectangle.

Parameters

hab (HAB) — input
Anchor-block handle.

pwrcprc (WRECT) — input/output
Rectangle to be converted.

The data type of the rectangle after conversion is RECTL.

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 — Map Dialog Points

#define INCL_WINDIALOGS /* Or use INCL_WIN or INCL_PM */

BOOL WinMapDlgPoints (HWND hwndDlg, PPOINTL aptlPoints, ULONG ulCount, BOOL fOptions)

This function maps points from dialog coordinates to window coordinates, or from window coordinates to dialog coordinates.

Parameters

hwndDlg (HWND) — input
Dialog-window handle.

aptlPoints (PPOINTL) — input/output
Coordinate points to be mapped.

The mapped points are substituted.

ulCount (ULONG) — input
Number of coordinate points.

fOptions (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

Coordinates-mapped indicator:

TRUE Coordinates successfully mapped

FALSE Coordinates not successfully mapped.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 — Map Window Points

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinMapWindowPoints (HWND hwndFrom, HWND hwndTo, PPOINTL aptlPoints,
                         LONG lCount)
```

This function maps a set of points from a coordinate space relative to one window into a coordinate space relative to another window.

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.

aptlPoints (PPOINTL) — input/output
Points to be mapped to the new coordinate system.

lCount (LONG) — input
Number of points to be mapped.

aptlPoints can be a RECTL structure, in which case this parameter should have the value 2.

Note: This is not supported in all languages.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 — Message Box

```
#define INCL_WINDIALOGS  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
USHORT WinMessageBox (HWND hwndParent, HWND hwndOwner, PSZ pszText,
                      PSZ pszTitle, USHORT usWindow, ULONG flStyle)
```

This function creates, displays, and operates a message box window.

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.

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.

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.

usWindow (USHORT) — input
Message-box window identity.

This value is passed to the HK_HELP hook if the WM_HELP message is received by the message-box window.

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 pushbutton.
MB_OKCANCEL	Message box contains both OK and CANCEL pushbuttons.
MB_CANCEL	Message box contains a CANCEL pushbutton.
MB_ENTER	Message box contains an ENTER pushbutton.
MB_ENTERCANCEL	Message box contains both ENTER and CANCEL pushbuttons.
MB_RETRYCANCEL	Message box contains both RETRY and CANCEL pushbuttons.
MB_ABORTRETRYIGNORE	Message box contains ABORT, RETRY, and IGNORE pushbuttons.
MB_YESNO	Message box contains both YES and NO pushbuttons.

MB_YESNOCANCEL Message box contains YES, NO, and CANCEL pushbuttons.

Help button

MB_HELP Message box contains a HELP pushbutton.

When this is selected a WM_HELP message is sent to the window procedure of the message box.

Color or Icon

MB_NOICON Message box 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 g 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.

WinMessageBox — Message Box

Returns

User-response value:

MBID_ENTER	ENTER pushbutton was selected
MBID_OK	OK pushbutton was selected
MBID_CANCEL	CANCEL pushbutton was selected
MBID_ABORT	ABORT pushbutton was selected
MBID_RETRY	RETRY pushbutton was selected
MBID_IGNORE	IGNORE pushbutton was selected
MBID_YES	YES pushbutton was selected
MBID_NO	NO pushbutton was selected
MBID_ERROR	Function not successful; an error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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 or Cancel keys are pressed. If the message box window has no CANCEL button, pressing the Escape key has no effect.

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 */
```

WinMultWindowFromIDs — Get Multiple Windows From Identities

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */

LONG WinMultWindowFromIDs (HWND hwndParent, PHWND ahwnd, ULONG ulFirst,
                           ULONG ulLast)
```

This function finds the handles of child windows that belong to a specified window and have window identities within a specified range.

Parameters

hwndParent (HWND) — input
Parent-window handle.

ahwnd (PHWND) — output
Window handles.

This array must contain (*ulLast* — *ulFirst* + 1) elements. The handle of a window, whose identity is WID (in the range *ulFirst* to *ulLast*), has a zero-based index in the array of (WID — *ulFirst*). If there is no window for a window identity within the range, the corresponding element in the array is NULLHANDLE.

ulFirst (ULONG) — input
First window identity value in the range (inclusive).

ulLast (ULONG) — input
Last window identity value in the range (inclusive).

Returns

Number of window handles returned:

0 No window handles returned

Other Number of window handles returned.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 –
Move to Next Character

```
#define INCL_WINCOUNTRY  /* Or use INCL_WIN or INCL_PM */
```

PSZ WinNextChar (HAB hab, ULONG ulCodepage, ULONG ulCountry, PSZ pszCurrentChar)

This function moves to the next character in a string.

Parameters

hab (HAB) – input
Anchor-block handle.

ulCodepage (ULONG) – input
Code page.

ulCountry (ULONG) – input
Country code.

pszCurrentChar (PSZ) – input
Current character in a null-terminated string.

Returns

Next character in the null-terminated string:

NULL End of string reached

Other Next character.

Possible returns from WinGetLastError

PMERR_INVALID_STRING_PARM The specified string parameter is invalid.

Remarks

This function handles DBCS strings.

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 — Offset Rectangle

```
#define INCL_WINRECTANGLES   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinOffsetRect (HAB hab, PRECTL prclrect, LONG lcx, LONG lcy)

This function offsets a rectangle.

Parameters

hab (HAB) — input
Anchor-block handle.

prclrect (PRECTL) — input/output
Rectangle to be offset.

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.

lcx (LONG) — input
x-value of offset.

lcy (LONG) — input
y-value of offset.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This function offsets the coordinates of *prclrect* by adding the value of the *lcx* parameter to both the left and right coordinates, and the value of the *lcy* 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 – Open Clipboard

#define INCL_WINCLIPBOARD /* Or use INCL_WIN or INCL_PM */

BOOL WinOpenClipbrd (HAB hab)

This function opens the clipboard.

Parameters

hab (HAB) – input
Anchor-block handle.

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);
```

WinOpenWindowDC – Open Window Device Context

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

```
HDC WinOpenWindowDC (HWND hwnd)
```

This function opens a device context for a window.

Parameters

hwnd (HWND) – input
Window handle.

Returns

Device-context handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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

WinOpenWindowDC — Open Window Device Context

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 — Peek Message

#define INCL_WINMESSAGEMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

```
BOOL WinPeekMsg (HAB hab, PQMSG pqmsgmsg, HWND hwndFilter, ULONG ulFirst,
                 ULONG ulLast, ULONG flOptions)
```

This function inspects the thread's message queue and returns to the application with or without a message.

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.

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

Message-available indicator:

TRUE Message available

FALSE No message available.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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 *pqmsgmsg* is null if the message is posted to the queue with a *hwnd* that is null.

WinPeekMsg — Peek Message

Related Functions

- WinCancelShutdown
- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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++;
```

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinPopupMenu (HWND hwndParent, HWND hwndOwner, HWND hwndMenu, LONG lx,
                   LONG ly, ULONG idItem, USHORT fsOptions)
```

This function causes a pop-up menu to be presented.

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.

lx (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 *fsOptions* parameter is also set.

ly (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 *fsOptions* parameter is also set.

idItem (ULONG) — input
Item identity.

This is used if either the PU_POSITIONONITEM or PU_SELECTITEM of the *fsOptions* parameter is also set.

fsOptions (USHORT) — 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 pointer.

The position of the pop-up menu can be affected, if either the PU_HCONSTRAIN or or PU_VCONSTRAIN values of the *fsOptions* parameter is also set.

This value also causes the pop-up menu item identified by the *idItem* to be selected.

WinPopupMenu — Pop-up Menu

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

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

WinPopupMenu — Pop-up Menu

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 — Post Message

```
#define INCL_WINMESSAGEMGR   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinPostMsg (HWND hwnd, ULONG ulMsgid, MPARAM mpParam1,
                 MPARAM mpParam2)
```

This function posts a message to the message queue associated with the window defined by *hwnd*.

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

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 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.

WinPostMsg — Post Message

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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 — Post Queue Message

```
#define INCL_WINMESSAGEMGR   /* Or use INCL_WIN or INCL_PM */

BOOL WinPostQueueMsg  (HMQ hmq, ULONG ulMsgId, MPARAM mpParam1,
                       MPARAM mpParam2)
```

This function posts a message to a message queue.

Parameters

hmq (HMQ) — input
Message-queue handle.

ulMsgId (ULONG) — input
Message identifier.

mpParam1 (MPARAM) — input
Parameter 1.

mpParam2 (MPARAM) — input
Parameter 2.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred, or the queue was full.

Possible returns from WinGetLastError

PMERR_INVALID_HMQ	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.

WinPostQueueMsg — Post Queue Message

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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 – Move to Previous Character

#define INCL_WINCOUNTRY /* Or use INCL_WIN or INCL_PM */

```
PSZ WinPrevChar (HAB hab, ULONG ulCodepage, ULONG ulCountry, PSZ pszStart,
                 PSZ pszCurrentChar)
```

This function moves to the previous character in a string.

Parameters

hab (HAB) – input
Anchor-block handle.

ulCodepage (ULONG) – input
Code page.

ulCountry (ULONG) – input
Country code.

pszStart (PSZ) – input
Character string that contains *pszCurrentChar*.

pszCurrentChar (PSZ) – input
Current character.

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	The specified string parameter is invalid.

Remarks

This function handles DBCS strings.

Related Functions

- WinCompareStrings
- WinLoadString
- WinNextChar
- WinSubstituteStrings
- WinUpper
- WinUpperChar

WinPrevChar –
Move to Previous Character

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 — Process Modal Dialog

```
#define INCL_WINDIALOGS   /* Or use INCL_WIN or INCL_PM */
```

ULONG WinProcessDlg (HWND hwndDlg)

This function dispatches messages while a modal dialog window is displayed.

Parameters

hwndDlg (HWND) — input
Dialog-window handle.

Returns

Reply value.

Value established by the WinDismissDlg function.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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.

WinProcessDlg – Process Modal Dialog

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 – Point In Rectangle

```
#define INCL_WINRECTANGLES   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinPtInRect (HAB hab, PRECTL prcIrect, PPOINTL pptIpoint)

This function queries whether a point lies within a rectangle.

Parameters

hab (HAB) – input
Anchor-block handle.

prcIrect (PRECTL) – input
Rectangle to be queried.

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.

pptIpoint (PPOINTL) – input
Point to be queried.

Returns

Success indicator:

TRUE *pptIpoint* lies within *prcIrect*

FALSE *pptIpoint* does not lie within *prcIrect*, or an error occurred.

Related Functions

- WinCopyRect
- WinEqualRect
- WinFillRect
- WinInflateRect
- WinIntersectRect
- WinIsRectEmpty
- WinOffsetRect
- WinSetRect
- WinSetRectEmpty
- WinSubtractRect
- WinUnionRect

WinPtInRect — Point In Rectangle

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 — Query Accelerator Table

```
#define INCL_WINACCELERATORS  /* Or use INCL_WIN or INCL_PM */
```

HACCEL WinQueryAccelTable (HAB hab, HWND hwndFrame)

This function queries the window or queue accelerator table.

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

Accelerator-table handle:

NULLHANDLE	Error occurred
Other	Accelerator-table handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.

Related Functions

- WinCopyAccelTable
- WinCreateAccelTable
- WinDestroyAccelTable
- WinLoadAccelTable
- WinSetAccelTable
- WinTranslateAccel

Related Messages

- WM_QUERYACCELTABLE

WinQueryAccelTable — Query Accelerator Table

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);
```

WinQueryActiveWindow — Query Active Window

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM */

HWND WinQueryActiveWindow (HWND hwndParent)

This function returns the active window for HWND_DESKTOP, or other parent window.

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

Active-window handle:

NULLHANDLE No window is active

Other Active-window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 — Query Anchor Block

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

HAB WinQueryAnchorBlock (HWND hwnd)

This function returns the anchor block handle of the caller.

Parameters

hwnd (HWND) — input
Window handle.

Returns

Anchor block handle.

NULLHANDLE Invalid *hwnd* parameter

Other Anchor block handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 — Query Atom Length

```
#define INCL_WINATOM   /* Or use INCL_WIN or INCL_PM */
```

ULONG WinQueryAtomLength (HATOMTBL hatomtblAtomTbl, ATOM atom)

This function queries the length of an atom represented by the specified 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

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	An invalid atom-table handle was specified.
PMERR_INVALID_ATOM	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

WinQueryAtomLength — Query Atom Length

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 — Query Atom Name

```
#define INCL_WINATOM   /* Or use INCL_WIN or INCL_PM */
```

ULONG WinQueryAtomName (HATOMTBL hatomtblAtomTbl, ATOM atom, PSZ pszBuffer, ULONG ulBufferMax)

This function returns an atom name associated with an atom.

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

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	An invalid atom-table handle was specified.
PMERR_INVALID_ATOM	The specified atom does not exist in the atom table.
PMERR_INVALID_STRING_PARM	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

WinQueryAtomName — Query Atom Name

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));
```

#define INCL_WINATOM /* Or use INCL_WIN or INCL_PM */

ULONG WinQueryAtomUsage (HATOMTBL hatomtblAtomTbl, ATOM atom)

This function returns the number of times an atom has been used.

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

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	An invalid atom-table handle was specified.
PMERR_INVALID_ATOM	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 — Query Checkstate of Button

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

USHORT WinQueryButtonCheckstate (HWND hwndDlg, USHORT usId)

This macro returns the checked state of the button control specified.

Parameters

hwndDlg (HWND) — input
Dialog window handle.

usId (USHORT) — input
Button control identity.

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
#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 – Query Capture

#define INCL_WININPUT /* Or use INCL_WIN or INCL_PM */

HWND WinQueryCapture (HWND hwndDesktop)

This function returns the handle of the window that has the pointer captured.

Parameters

hwndDesktop (HWND) – input
Desktop-window handle:

HWND_DESKTOP The desktop-window handle

Other Specified desktop-window handle.

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 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 – Query Class Information

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinQueryClassInfo (HAB hab, PSZ pszClassName, PCLASSINFO pclsiClassInfo)

This function returns window class information.

Parameters

hab (HAB) – input
Anchor-block handle.

pszClassName (PSZ) – input
Class name.

pclsiClassInfo (PCLASSINFO) – output
Class information structure.

Returns

Class-exists indicator:

TRUE Class does exist

FALSE Class does not exist.

Possible returns from WinGetLastError

PMERR_INVALID_INTEGER_ATOM	The specified atom is not a valid integer atom.
PMERR_INVALID_ATOM_NAME	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND	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 page 11-2. 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

WinQueryClassInfo – Query Class Information

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 – Query Class Name

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

LONG WinQueryClassName (HWND hwnd, LONG lLength, PCH pchBuffer)

This function copies the window class name, as a null-terminated string, into a buffer.

Parameters

hwnd (HWND) – input
Window handle.

If this window is of any of the preregistered WC_* classes (see page 11-2), 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*.

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

Returned class name length.

This is the length, **excluding** the null-termination character.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_STRING_PARM	The specified string parameter is invalid.

Related Functions

- WinCalcFrameRect
- WinCreateFrameControls
- WinCreateStdWindow
- WinCreateWindow
- WinDefWindowProc
- WinDestroyWindow
- WinQueryClassInfo
- WinRegisterClass
- WinSubclassWindow

WinQueryClassName – Query Class Name

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 — Query Class Pointer-Conversion Procedure

```
#define INCL_WINTHUNKAPI  /* Or use INCL_WIN or INCL_PM */
```

PFN WinQueryClassThunkProc (PSZ pszClassName)

This call queries the pointer-conversion procedure associated with a class.

Parameters

pszClassName (PSZ) — input
Window-class name.

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 — Query Clipboard Data

```
#define INCL_WINCLIPBOARD   /* Or use INCL_WIN or INCL_PM */
```

ULONG WinQueryClipbrdData (HAB hab, ULONG ulfmt)

This function obtains a handle to the current clipboard data with a specified format.

Parameters

hab (HAB) — input
Anchor-block handle.

ulfmt (ULONG) — input
Format of the data to be accessed.

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 – Query Clipboard Format Information

```
#define INCL_WINCLIPBOARD   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinQueryClipbrdFmtInfo (HAB hab, ULONG ulfmt, PULONG pulFmtInfo)

This function determines whether a particular format of data is present in the clipboard, and if so, provides information about that format.

Parameters

hab (HAB) – input
Anchor-block handle.

ulfmt (ULONG) – input
Format of the data to be queried.

pulFmtInfo (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, *pulFmtInfo* is set to CFI_HANDLE. If the format is CF_TEXT or CF_DSPTEXT, *pulFmtInfo* is set to CFI_POINTER. If the format is user-defined, *pulFmtInfo* is set to the value used in the WinSetClipbrdData function.

Returns

Format-exists indicator:

TRUE *ulfmt* exists in the clipboard and *pulFmtInfo* is set

FALSE *ulfmt* does not exist in the clipboard and *pulFmtInfo* is not set.

Possible returns from WinGetLastError

PMERR_INVALID_FLAG	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.

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 — Query Clipboard Owner

```
#define INCL_WINCLIPBOARD   /* Or use INCL_WIN or INCL_PM */
```

HWND WinQueryClipbrdOwner (HAB hab)

This function obtains any current clipboard owner window.

Parameters

hab (HAB) — input
Anchor-block handle.

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 — Query Clipboard Viewer

#define INCL_WINCLIPBOARD /* Or use INCL_WIN or INCL_PM */

HWND WinQueryClipbrdViewer (HAB hab)

This function obtains any current clipboard viewer window.

Parameters

hab (HAB) — input
Anchor-block handle.

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 – Query Code Page

```
#define INCL_WINCOUNTRY  /* Or use INCL_WIN or INCL_PM */
```

ULONG WinQueryCp (HMQ hmq)

This function returns the queue code page for the specified message queue.

Parameters

hmq (HMQ) – input
Message queue.

Returns

Code page:

0 Error occurred

Other Queue code page for the specified message queue.

Possible returns from WinGetLastError

PMERR_INVALID_HMQ 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 – Query Code Page List

#define INCL_WINCOUNTRY /* Or use INCL_WIN or INCL_PM */

ULONG WinQueryCpList (HAB hab, ULONG ulcount, PULONG aulCodepage)

This function queries available code pages.

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 Chapter 34, "Code Pages" on page 34-1.

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 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 — Query Cursor Information

```
#define INCL_WINCURSORS  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinQueryCursorInfo (HWND hwndDeskTop, PCURSORINFO pcsriCursorInfo)

This function obtains information about any current cursor.

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

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 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 – Query Desktop Background

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinQueryDesktopBkgnd (HWND hwndDeskTop, PDESKTOP pDeskTopState)

This function returns the desktop structure, which contains the information about the current state of the desktop background.

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle.

HWND_DESKTOP	The desktop window
Other	Specified desktop window.

pDeskTopState (PDESKTOP) – output
Desktop-state structure.

Returns

Success indicator:

TRUE	Desktop-window status provided
FALSE	Desktop-window status not provided.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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

WinQueryDesktopBkgnd — Query Desktop Background

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 — Query Desktop Window

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

HWND WinQueryDesktopWindow (HAB hab, HDC hdc)

This function returns the desktop-window handle.

Parameters

hab (HAB) — input
Anchor-block handle.

hdc (HDC) — input
Device-context handle:

NULLHANDLE Default device (the screen).

Returns

Desktop-window handle:

NULLHANDLE Error occurred

Other Desktop-window handle.

Possible returns from WinGetLastError

PMERR_INV_HDC 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

WinQueryDesktopWindow — Query Desktop Window

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 — Query Dialog Item Short

```
#define INCL_WINDIALOGS   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
BOOL WinQueryDlgItemShort (HWND hwndDlg, ULONG idItem, PSHORT psResult,
                           BOOL fSigned)
```

This function converts the text of a dialog item into an integer value.

Parameters

hwndDlg (HWND) — input
Parent-window handle.

idItem (ULONG) — input
Identity of the child window whose text is to be converted.

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

Success indicator:

TRUE Successful conversion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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

WinQueryDlgItemShort – Query Dialog Item Short

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 — Query Dialog Item Text

```
#define INCL_WINDIALOGS   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

ULONG WinQueryDlgItemText (HWND hwndDlg, ULONG idItem, LONG lMaxText, PSZ pszText)

This function queries a text string in a dialog item.

Parameters

hwndDlg (HWND) — input
Parent-window handle.

idItem (ULONG) — input
Identity of the child window whose text is to be queried.

lMaxText (LONG) — input
Length of *pszText*.

pszText (PSZ) — output
Output string.

This is the text string that is obtained from the dialog item.

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 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

WinQueryDlgItemText — Query Dialog Item Text

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 — Query Dialog Item Text Length

```
#define INCL_WINDIALOGS  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

LONG WinQueryDlgItemTextLength (HWND hwndDlg, ULONG idItem)

This function queries the length of the text string in a dialog item, not including any null termination character.

Parameters

hwndDlg (HWND) — input
Parent-window handle.

idItem (ULONG) — input
Identity of the child window whose text is to be queried.

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

WinQueryDlgItemTextLength – Query Dialog Item Text Length

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);
      /* . */
      /* . */
}
```

#define INCL_WININPUT /* Or use INCL_WIN or INCL_PM */

HWND WinQueryFocus (HWND hwndDeskTop)

This function returns the focus window. It is NULLHANDLE if there is no focus window.

Parameters

hwndDeskTop (HWND) – input
Desktop-window handle:

HWND_DESKTOP The desktop-window handle

Other Specified desktop-window handle.

Returns

Focus-handle.

NULL Error occurred or no focus window.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 — Query Help Instance

```
#define INCL_WINHELP  /* Or use INCL_WIN or INCL_PM */

HWND WinQueryHelpInstance (HWND hwndApp)
```

This function enables the application to query the instance of the help manager associated with the application-supplied window handle.

Parameters

hwndApp (HWND) — input
Handle of the application window.

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 — Counts Number of Items in Listbox

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

SHORT WinQueryLboxCount (HWND hwndLbox)

This macro returns the number of items in the List Box.

Parameters

hwndLbox (HWND) — input
Listbox handle.

Returns

Number of items in the list box.

Remarks

This macro expands to:

```
#define WinQueryLboxCount(hwndLbox)
  ((SHORT)WinSendMsg(hwndLbox,
                     LM_QUERYITEMCOUNT,
                     (MPARAM)NULL,
                     (MPARAM)NULL))
```

This function 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>
SHORT cWindows;
HWND hwndWindowLB;

cWindows = WinQueryLboxCount(hwndWindowLB);
while (cWindows)
{
  /* Loop through all windows, selecting all */

WinSendMsg(hwndWindowLB,
           LM_SELECTITEM,
           (MPARAM)--cWindows,
           (MPARAM)TRUE);
}
```

WinQueryLboxItemText – Query Listbox Item Text

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

SHORT WinQueryLboxItemText (HWND hwndLbox, SHORT sIndex, PSZ pszText, SHORT scchMax)

This macro fills the buffer with the text of the indexed item. It returns the length of the text.

Parameters

hwndLbox (HWND) – input
List box handle.

sIndex (SHORT) – input
Index of the listbox item.

pszText (PSZ) – input
Pointer to a null terminated string.

scchMax (SHORT) – input
Maximum number of characters allocated to the string.

Returns

Actual text length copied.

Remarks

This macro expands to:

```
#define WinQueryLboxItemText(hwndLbox, sIndex, pszText, scchMax)
  ((SHORT)WinSendMsg(hwndLbox,
                     LM_QUERYITEMTEXT,
                     MPFROM2SHORT((sIndex), (scchMax)),
                     MPFROMP(pszText)))
```

This function requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- LM_INSERTITEM

WinQueryLboxItemText — Query Listbox Item Text

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>
SHORT 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 — Query Listbox Item Text Length

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM */

SHORT WinQueryLboxItemTextLength (HWND hwndLbox, SHORT sIndex)

This macro returns the length of the text of the indexed item in the List Box.

Parameters

hwndLbox (HWND) — input
Listbox handle.

sIndex (SHORT) — input
Index of the item in the List Box.

Returns

Text length of the indexed item.

Remarks

This macro expands to:

```
#define WinQueryLboxItemTextLength(hwndLbox, sIndex)
  ((SHORT)WinSendMsg(hwndLbox,
                     LM_QUERYITEMTEXTLENGTH,
                     MPFROMSHORT(sIndex),
                     (MPARAM)NULL))
```

This function requires the existence of a message queue.

Related Functions

- WinSendMsg

Related Messages

- LM_QUERYITEMTEXTLENGTH

WinQueryLboxItemTextLength — Query Listbox Item Text Length

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>
SHORT 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 — Query the Selected Item in Listbox

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

SHORT WinQueryLboxSelectedItem (HWND hwndLbox)

This macro returns the index of the selected item in the List Box (for single selection only).

Parameters

hwndLbox (HWND) — input
List box handle.

Returns

Index of the selected item.

Remarks

This macro expands to:

```
#define WinQueryLBoxSelectedItem (hwndLbox)
  ((SHORT)WinSendMsg(hwndLbox,
                     LM_QUERYSELECTION,
                     MPFROMSHORT(LIT_FIRST),
                     (MPARAM)NULL))
```

This function 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.

```
#define INCL_WINLISTBOXES
#define INCL_WINWINDOWMGR
#include <OS2.H>
HWND hwndLB;
SHORT index;
char szTemp[256];

index = WinQueryLboxSelectedItem(hwndLB);

WinQueryLboxItemText(hwndLB,
                     index,
                     szTemp,
          WinQueryLboxItemTextLength(hwndLB,index));
```

WinQueryMsgPos — Query Message Position

```
#define INCL_WINMESSAGEMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinQueryMsgPos (HAB hab, PPOINTL pptlptrpos)

This function returns the pointer position, in screen coordinates, when the last message obtained from the current message queue is posted.

Parameters

hab (HAB) — input
Anchor-block handle.

pptlptrpos (PPOINTL) — output
Pointer position in screen coordinates.

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
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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 – Query Message Time

```
#define INCL_WINMESSAGEMGR   /* Or use INCL_WIN or INCL_PM */
```

ULONG WinQueryMsgTime (HAB hab)

This function returns the message time for the last message retrieved by the WinGetMsg or WinPeekMsg functions from the current message queue.

Parameters

hab (HAB) – input
Anchor-block handle.

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
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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 */
```

WinQueryObjectWindow — Query Object Window

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */

HWND WinQueryObjectWindow (HWND hwndDeskTop)
```

This function returns the desktop object window handle.

Parameters

hwndDeskTop (HWND) — input
Desktop-window handle:

HWND_DESKTOP The desktop-window handle

Other Specified desktop-window handle.

Returns

Object-window handle.

NULLHANDLE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 — Query Pointer

```
#define INCL_WINPOINTERS   /* Or use INCL_WIN or INCL_PM */
```

HPOINTER WinQueryPointer (HWND hwndDeskTop)

This function returns the pointer handle for *hwndDeskTop*.

Parameters

hwndDeskTop (HWND) — input
Desktop-window handle:

HWND_DESKTOP The desktop-window handle

Other Specified desktop-window handle.

Returns

Pointer handle.

NULLHANDLE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND An invalid window handle was specified.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinSetPointer
- WinSetPointerPos
- 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 — Query Pointer Information

```
#define INCL_WINPOINTERS   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinQueryPointerInfo (HPOINTER hptr, PPOINTERINFO pptriPointerInfo)

This function returns pointer information.

Parameters

hptr (HPOINTER) — input
Pointer handle.

pptriPointerInfo (POINTERINFO) — output
Pointer-information structure.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HPTR 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
- WinSetPointer
- WinSetPointerPos
- WinShowPointer

WinQueryPointerInfo — Query Pointer Information

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 — Query Pointer Position

```
#define INCL_WINPOINTERS   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinQueryPointerPos (HWND hwndDeskTop, PPOINTL pptlPoint)

This function returns the pointer position.

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

Pointer position returned indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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
- WinSetPointer
- WinSetPointerPos
- WinShowPointer

WinQueryPointerPos — Query Pointer Position

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 — Query Presentation Parameter

```
#define INCL_WINSYS   /* Or use INCL_WIN or INCL_PM */

ULONG WinQueryPresParam (HWND hwnd, ULONG idAttrType1, ULONG idAttrType2,
                         PULONG pidAttrTypeFound, ULONG cbAttrValueLen,
                         PVOID pAttrValue, ULONG flOptions)
```

This function queries the values of presentation parameters for a window

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) — input/output
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 (PVOID) — output
Attribute value.

The value of the presentation parameter attribute found.

flOptions (ULONG) — input
Options.

Options controlling the query. Any of these can be ORed together.

QPF_NOINHERIT	For the purposes of this query, presentation parameters are not inherited from the owners of the window specified by *hwnd*. If not specified (default), presentation parameters are inherited.
QPF_ID1COLORINDEX	*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.

WinQueryPresParam – Query Presentation Parameter

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 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 — Query Queue Information

```
#define INCL_WINMESSAGEMGR  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

BOOL WinQueryQueueInfo (HMQ hmq, PMQINFO pmqiMqinfo, ULONG cbCopied)

This function returns the information for the specified queue.

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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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 — Query Queue Status

```
#define INCL_WINMESSAGEMGR  /* Or use INCL_WIN or INCL_PM */
```

```
ULONG WinQueryQueueStatus (HWND hwndDeskTop)
```

This function returns a code indicating the status of the message queue associated with the caller.

Parameters

hwndDeskTop (HWND) — input
Desktop-window handle:

HWND_DESKTOP The desktop-window handle

Other Desktop-window handle returned by WinQueryDesktopWindow.

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, or WM_BUTTON2DBLCLK message to be placed in the queue.

QS_MOUSEBUTTON An input event has caused a WM_BUTTON1UP, WM_BUTTON1DOWN, WM_BUTTON1DBLCLK, WM_BUTTON2UP, WM_BUTTON2DOWN, or WM_BUTTON2DBLCLK message to be placed in the queue.

QS_MOUSEMOVE An input event has caused a WM_MOUSEMOVE message to be placed in the queue.

QS_TIMER A timer event has caused a WM_TIMER message to be placed in the queue.

QS_PAINT A WM_PAINT message is available.

QS_SEM1 A WM_SEM1 message is available.

QS_SEM2 A WM_SEM2 message is available.

QS_SEM3 A WM_SEM3 message is available.

QS_SEM4 A WM_SEM4 message is available.

QS_POSTMSG A message has been posted to the queue. Note that this message is probably not one of the messages listed above, but could be a WM_CHAR, WM_MOUSEMOVE 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
- WinRegisterUserDatatype
- WinRegisterUserMsg
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Related Messages

- WM_BUTTON1UP
- WM_BUTTON1DOWN
- WM_BUTTON1DBLCLK
- WM_BUTTON2UP
- WM_BUTTON2DOWN
- WM_BUTTON2DBLCLK
- WM_CHAR
- WM_MOUSEMOVE
- WM_PAINT
- WM_SEM1
- WM_SEM2
- WM_SEM3
- WM_SEM4
- WM_TIMER

WinQueryQueueStatus — Query Queue Status

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>
HAB hab;

if(WinQueryQueueStatus(HWND_DESKTOP) == QS_MOUSEMOVE)
{
   /* . */
   /* . */
}
```

WinQuerySessionTitle — Query Session Title

```
#define INCL_WINSWITCHLIST   /* Or use INCL_WIN or INCL_PM */

ULONG WinQuerySessionTitle (HAB hab, ULONG ulSession, PSZ pszTitle, ULONG ulTitlelen)
```

This function obtains the title under which a specified application is started, or is added to the Window List.

Parameters

hab (HAB) — input
Anchor-block handle.

ulSession (ULONG) — input
IBM Operating System/2 session identity of application whose title is requested:

0 Use the session identity of the caller

Other Use the specified session identity.

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

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

WinQuerySessionTitle — Query Session Title

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);
```

```
#define INCL_WINSWITCHLIST   /* Or use INCL_WIN or INCL_PM */
```

ULONG WinQuerySwitchEntry (HSWITCH hswitchSwitch, PSWCNTRL pswctlSwitchData)

This function obtains a copy of the Window List data for a specific application.

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

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 — Query Switch Handle

```
#define INCL_WINSWITCHLIST   /* Or use INCL_WIN or INCL_PM */

HSWITCH WinQuerySwitchHandle (HWND hwnd, PID idProcess)
```

This function obtains the Window List handle belonging to a window.

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

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 NULLHANDLE 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 – Query Switch List

```
#define INCL_WINSWITCHLIST  /* Or use INCL_WIN or INCL_PM */
```

ULONG WinQuerySwitchList (HAB hab, PSWBLOCK pswblkBlock, ULONG ulLength)

This function obtains information about the entries in the Window List.

Parameters

hab (HAB) – input
Anchor-block handle.

pswblkBlock (PSWBLOCK) – input/output
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

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
- WinQuerySessionTitle
- WinQuerySwitchEntry
- WinQuerySwitchHandle
- WinQueryTaskSizePos
- WinQueryTaskTitle
- WinRemoveSwitchEntry
- WinSwitchToProgram

Example Code

This example calls WinQuerySwitchList to determine the number of items in the Task List, allocates memory for the required buffer, and calls WinQuerySwitchList again to fill the buffer with the information about each program in the Task List.

```
#define INCL_DOSMEMMGR
#define INCL_WINSWITCHLIST
#include <OS2.H>
HAB hab;
HWND hwndFrame;
ULONG cbItems, cbBuf;
ULONG  pBase;
PSWBLOCK pswblk;
SEL sel;

DosAllocMem((PPVOID)pBase,
            (ULONG)40000,
             PAG_READ |
             PAG_WRITE);

/* . */

 cbItems = WinQuerySwitchList(hab, NULL, 0);   /* gets num. of items */
 cbBuf = (cbItems * sizeof(SWENTRY)) + sizeof(HSWITCH);
 DosSubAllocMem((PVOID)pBase,
                (PPVOID)pswblk,
                (ULONG)cbBuf);
 WinQuerySwitchList(hab, pswblk, cbBuf);       /* gets struct. array */
```

WinQuerySysColor — Query System Color

```
#define INCL_WINSYS   /* Or use INCL_WIN or INCL_PM */
```

LONG WinQuerySysColor (HWND hwndDeskTop, LONG lColor, LONG lReserved)

This function returns the system color.

Parameters

hwndDeskTop (HWND) — input
Desktop-window handle:

HWND_DESKTOP The desktop-window handle

Other Specified desktop-window handle.

lColor (LONG) — input
System color-index value.

Must be one of the SYSCLR_* index values defined under the WinSetSysColors function.

lReserved (LONG) — input
Reserved.

0 Reserved value; must be 0.

Returns

RGB value.

RGB value corresponding to the *lColor* parameter.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE	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.

Related Functions

- WinSetSysColors

Example Code

This example uses the WinQuerySysColor to find the RGB index of the system pushbutton, SYSCLR_BUTTONDEFAULT

```
#define INCL_WINSYS
#define INCL_WINDESKTOP
#include <OS2.H>
HAB hab;
LONG lRgbColor;


lRgbColor = WinQuerySysColor(HWND_DESKTOP,
                             SYSCLR_BUTTONDEFAULT,
                             0L);
```

WinQuerySysModalWindow — Query System Modal Window

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

HWND WinQuerySysModalWindow (HWND hwndDeskTop)

This function returns the current system modal window.

Parameters

hwndDeskTop (HWND) — input
Desktop-window handle:

HWND_DESKTOP	The desktop-window handle
Other	Specified desktop-window handle.

Returns

Handle of system modal window:

NULLHANDLE	No system modal window
Other	Handle of system modal window.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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 — Query System Pointer

```
#define INCL_WINPOINTERS   /* Or use INCL_WIN or INCL_PM */
```

HPOINTER WinQuerySysPointer (HWND hwndDeskTop, LONG lIdentifier, BOOL fCopy)

This function returns the system-pointer handle.

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 system pointer and return its handle. Specify this value if the system pointer is to be modified.
FALSE	Return the handle of the system pointer.

WinQuerySysPointer — Query System Pointer

Returns

Pointer handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE	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
- WinSetPointer
- WinSetPointerPos
- 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);
```

WinQuerySysValue — Query System Value

```
#define INCL_WINSYS  /* Or use INCL_WIN or INCL_PM */
```

```
LONG WinQuerySysValue (HWND hwndDeskTop, LONG lValueId)
```

This function returns a system value.

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.

lValueId (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_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.

WinQuerySysValue – Query System Value

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_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_*))

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	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE	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);
```

WinQuerySystemAtomTable — Query System Atom Table

#define INCL_WINATOM /* Or use INCL_WIN or INCL_PM */

HATOMTBL WinQuerySystemAtomTable ()

This function returns the handle of the system atom table.

Parameters

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);
```

WinQueryTaskSizePos — Query Task Window Size and Position

```
#define INCL_WINSWITCHLIST   /* Or use INCL_WIN or INCL_PM */
```

ULONG WinQueryTaskSizePos (HAB hab, ULONG ulID, PSWP pswp)

This function obtains the recommended size, position and status for the first window of a newly started application (typically the main window).

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

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

WinQueryTaskSizePos – Query Task Window Size and Position

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 — Query Task Title

```
#define INCL_WINSWITCHLIST   /* Or use INCL_WIN or INCL_PM */
```

ULONG WinQueryTaskTitle (ULONG ulSession, PSZ pszTitle, ULONG ulTitlelen)

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.)

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

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

WinQueryTaskTitle – Query Task Title

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 — Query Update Rectangle

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinQueryUpdateRect (HWND hwnd, PRECTL prclPrc)

This function returns the rectangle that bounds the update region of a specified window.

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

Success indicator:

TRUE Successful completion

FALSE Window has no update region; it is wholly valid, therefore *prclPrc* is NULL.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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

WinQueryUpdateRect — Query Update Rectangle

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 */
```

WinQueryUpdateRegion — Query Update Region

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

LONG WinQueryUpdateRegion (HWND hwnd, HRGN hrgn)

This call obtains an update region of a window.

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

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	An invalid window handle was specified.
PMERR_HRGN_BUSY	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.

WinQueryUpdateRegion — Query 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  */
               .
               .
               .

}
```

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

ULONG WinQueryVersion (HAB hab)

This function returns the version, the revision level and the environment of PM.

Parameters

hab (HAB) – input
Anchor-block handle.

Returns

System information within which the application is operating:

SYSINF_ENV
Environment:

QV_OS2 OS/2.

SYSINF_MAJVER
Major version number of PM.

10 OS/2 Presentation Manager Version 1.

SYSINF_MINVER
Minor version (revision) number of PM.

10 Revision 1

20 Revision 2.

Example Code

This example gets the version of PM that is running.

```
#define INCL_GPIREGIONS
#include <OS2.H>
#include <stdio.h>
HAB hab;
ULONG lVer;
lVer = WinQueryVersion(hab);
printf("PM revision is %d ",lVer);
```

WinQueryWindow — Query Window

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */

HWND WinQueryWindow (HWND hwnd, LONG lCode)
```

This function returns the handle of a window that has a specified relationship to a specified window.

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

Window handle.

Handle of window related to *hwnd*.

Possible returns from WinGetLastError

PMERR_INVALID_HWND An invalid window handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE 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 – Query Window Device Context

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

HDC WinQueryWindowDC (HWND hwnd)

This function returns the device context for a given window.

Parameters

hwnd (HWND) – input
Window handle.

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	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))
{
     /*  ...    */
}
```

```
#define INCL_WINTHUNKAPI  /* Or use INCL_WIN or INCL_PM */
```

ULONG WinQueryWindowModel (HWND hwnd)

This function queries the memory model associated with a window.

Parameters

hwnd (HWND) — input
Window handle.

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 – Query Window Position

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */

BOOL WinQueryWindowPos (HWND hwnd, PSWP pswp)
```

This function queries the window size and position of a visible window.

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 *flOptions* 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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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 — Query Window Process

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinQueryWindowProcess (HWND hwnd, PPID pidpid, PTID pidtid)

This function obtains the process identity and thread identity of the thread that created a window.

Parameters

hwnd (HWND) — input
Window handle.

pidpid (PPID) — output
Process identity of the thread that created the window.

pidtid (PTID) — output
Thread identity of the thread that created the window.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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 — Query Window Pointer

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

```
PVOID WinQueryWindowPtr (HWND hwnd, LONG lb)
```

This function retrieves a pointer value from the memory of the reserved window word.

Parameters

hwnd (HWND) — input
Window handle which has the pointer to retrieve.

lb (LONG) — input
Index.

Zero-based index of the pointer value to retrieve. The units of **b** are bytes. Valid values are zero through (*usExtra* —4), where *usExtra* 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

Pointer value.

NULL Error occurred.

Other Pointer value.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE	The value of a parameter was not within the defined valid range for that parameter.

Remarks

The *lb* 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 – Query Window Rectangle

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinQueryWindowRect (HWND hwnd, PRECTL prclRect)
```

This function returns a window rectangle.

Parameters

hwnd (HWND) – input
Window handle whose rectangle is retrieved.

prclRect (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 –32 768 through 32 767. The data type WRECT may also be used, if supported by the language.

Returns

Rectangle-returned indicator:

TRUE Rectangle successfully returned

FALSE Rectangle not successfully returned.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 — Query Window Text

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

LONG WinQueryWindowText (HWND hwnd, LONG lLength, PCH pchBuffer)

This function copies window text into a buffer.

Parameters

hwnd (HWND) — input
Window handle.

If *hwnd* is a frame-window handle, the title-bar window text is copied.

lLength (LONG) — input
Length.

Length of *pchBuffer*.

pchBuffer (PCH) — output
Window text.

Returns

Length of returned text.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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.

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 — Query Window Text Length

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

LONG WinQueryWindowTextLength (HWND hwnd)

This call returns the length of the window text, excluding any null termination character.

Parameters

hwnd (HWND) — input
Window handle.

Returns

Length of the window text.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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 — Query Window Pointer-Conversion Procedure

```
#define INCL_WINTHUNKAPI  /* Or use INCL_WIN or INCL_PM */
```

PFN WinQueryWindowThunkProc (HWND hwnd)

This function queries the pointer-conversion procedure associated with a window.

Parameters

hwnd (HWND) — input
Window handle.

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 — Query Window Long

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

ULONG WinQueryWindowULong (HWND hwnd, LONG lb)

This function obtains the unsigned long integer value, at a specified offset, from the memory of a reserved window word, of a given window.

Parameters

hwnd (HWND) — input
Handle of window to be queried.

lb (LONG) — input
Index.

Zero-based index into the window words of the value to be queried. The units of **b** are bytes. Valid values are zero through (*usExtra* —4), where *usExtra* 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 pushbutton for a dialog. The default pushbutton is the one that sends its WM_COMMAND message when the enter key is pressed.
Other	Zero-based index.

Returns

Value contained in the window word.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE	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 *lb* 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 — Query Window Short

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

ULONG WinQueryWindowUShort (HWND hwnd, LONG lb)

This function obtains the unsigned short integer value at a specified offset from the reserved window word's memory of a given window.

Parameters

hwnd (HWND) — input
Handle of window to be queried.

lb (LONG) — input
Index.

Zero-based index into the window words of the value to be queried. The units of **b** are bytes. Valid values are zero through (*usExtra* –2), where *usExtra* 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_FLASHWINDOW	Frame window is flashing.
FF_ACTIVE	Frame window is displayed in the active state.
FF_SELECTED	Frame window is selected.
FF_FLASHHILITE	Window is currently flashed. This indicator toggles with each flash.
FF_OWNERHIDDEN	Frame window is hidden as a result of its owner being hidden or minimized. This indicator is set only if the window and its owner are siblings.
FF_DLGDISMISSED	Dialog has been dismissed by the WinDismissDlg function.
FF_OWNERDISABLED	Window's owner is disabled. This indicator is only set if the window and its owner are siblings.

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

Value contained in the indicated window word.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE	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. */
}
```

WinQueryObject – Query Workplace Object Handle

#define INCL_WINWORKPLACE

HOBJECT WinQueryObject (PSZ pszObjectID)

The WinQueryObject function returns a handle to the given object

Parameters

pszObjectID (PSZ) – input

The ObjectID of an existing object, for example <WP_DESKTOP>, or alternatively the fully qualified filename of any file or directory.

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.

Related Functions

- WinCreateObject
- WinDestroyObject
- WinSetObjectData

```
#define INCL_WIN  /* Or use INCL_PM */

LONG WinRealizePalette (HWND hwnd, HPS hps, PULONG pcclr)
```

This function indicates that drawing is about to take place after a palette has been selected.

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

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	An attempt to realize a palette failed because no palette was previously selected into the Presentation Space.
PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INV_HDC	An invalid device-context handle or (micro presentation space) presentation-space handle was specified.
PMERR_HDC_BUSY	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	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 a GpiSelectPalette function or in response to a WM_REALIZEPALETTE message. It causes the system to ensure that the palette is appropriately realized for all drawing operations.

WinRealizePalette – Realize Palette

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 — Register Window Class

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinRegisterClass (HAB hab, PSZ pszClassName, PFNWP pWndProc,
                       ULONG flClassStyle, USHORT usExtra)
```

This function registers a window class.

Parameters

hab (HAB) — input
Anchor-block handle.

pszClassName (PSZ) — input
Window-class name.

An application-specified class name.

pWndProc (PFNWP) — input
Window-procedure identifier.

Can be NULL if the application does not provide its own window procedure.

flClassStyle (ULONG) — input
Default-window style.

This can be any of the standard class styles (CS_*) (see page 12-1) in addition to any class-specific styles that are defined. These styles can be augmented when a window of this class is created.

A public window class is created if the CS_PUBLIC style is specified, otherwise a private class is created. The CS_PUBLIC style must only be specified for the shell process.

Public classes are available for creating windows from any process. Private classes are only available to the registering process.

usExtra (USHORT) — input
Reserved storage.

This is the number of bytes of storage reserved per window created of this class for application use.

Returns

Window-class-registration indicator:

TRUE Window class successfully registered

FALSE Window class not successfully registered.

Possible returns from WinGetLastError

PMERR_INVALID_FLAG	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	The specified atom is not a valid integer atom.
PMERR_INVALID_HATOMTBL	An invalid atom-table handle was specified.
PMERR_INVALID_ATOM_NAME	An invalid atom name string was passed.
PMERR_ATOM_NAME_NOT_FOUND	The specified atom name is not in the atom table.

WinRegisterClass — Register Window Class

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 – Register Workplace Object Class

#define INCL_WINWORKPLACE

BOOL WinRegisterObjectClass (PSZ pszClassName, PSZ pszModname)

The WinRegisterObjectClass function registers a workplace object class.

Parameters

pszClassName (PSZ) – input

A pointer to a zero-terminated string which contains the name of the object class being registered in the workplace.

pszModname (PSZ) – input

A pointer to a zero-terminated string which contains the name of the DLL which holds the object definition.

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 – Register User Data Type

```
#define INCL_WINMESSAGEMGR  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

BOOL WinRegisterUserDatatype (HAB hab, LONG lDatatype, LONG lCount, PSHORT asTypes)

This function registers a data type and defines its structure.

Parameters

hab (HAB) – input
Anchor-block handle.

lDatatype (LONG) – input
Data type code to be defined.

This must not be less than DTYP_USER, and must not have been defined previously.

lCount (LONG) – input
Number of elements.

Must not be less than one.

asTypes (PSHORT) – 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 *asTypes* 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 data type.
DTYP_BIT16	See USHORT data type.
DTYP_BIT32	See ULONG data type.
DTYP_BIT8	See UCHAR data type.
DTYP_BOOL	See BOOL data type.
DTYP_COUNT2	See USHORT data type.
DTYP_COUNT2B	See USHORT data type.
DTYP_COUNT2CH	See USHORT data type.
DTYP_COUNT4B	See ULONG data type.
DTYP_CPID	See USHORT data type.
DTYP_ERRORID	See ERRORID data type.
DTYP_IDENTITY	See USHORT data type.
DTYP_IDENTITY4	See ULONG data type.
DTYP_INDEX2	See USHORT data type.
DTYP_IPT	See IPT data type.
DTYP_LENGTH2	See USHORT data type.
DTYP_LENGTH4	See ULONG data type.
DTYP_LONG	See LONG data type.

DTYP_OFFSET2B	See USHORT data type.
DTYP_PID	See PID data type.
DTYP_PIX	See PIX data type.
DTYP_PROGCATEGORY	See PROGCATEGORY data type.
DTYP_PROPERTY2	See USHORT data type.
DTYP_PROPERTY4	See LONG data type.
DTYP_RESID	See HMODULE data type.
DTYP_SEGOFF	See NPBYTE data type.
DTYP_SHORT	See SHORT data type.
DTYP_TID	See TID data type.
DTYP_TIME	See LONG data type.
DTYP_UCHAR	See UCHAR data type.
DTYP_ULONG	See ULONG data type.
DTYP_USHORT	See USHORT data type.
DTYP_WIDTH4	See LONG data type.
DTYP_WNDPROC	See PFNWP data type.
Handle Data Types:	
DTYP_HAB	See HAB data type.
DTYP_HACCEL	See HACCEL data type.
DTYP_HAPP	See HAPP data type.
DTYP_HATOMTBL	See HATOMTBL data type.
DTYP_HBITMAP	See HBITMAP data type.
DTYP_HDC	See HDC data type.
DTYP_HENUM	See HENUM data type.
DTYP_HINI	See HINI data type.
DTYP_HLIB	See HLIB data type.
DTYP_HMF	See HMF data type.
DTYP_HMQ	See HMQ data type.
DTYP_HPOINTER	See HPOINTER data type.
DTYP_HPROGRAM	See HPROGRAM data type.
DTYP_HPS	See HPS data type.
DTYP_HRGN	See HRGN data type.
DTYP_HSEM	See HSEM data type.
DTYP_HSPL	See HSPL data type.
DTYP_HSWITCH	See HSWITCH data type.
DTYP_HWND	See HWND data type.
Character/String/Buffer Data Types:	
DTYP_BYTE	See BYTE data type.
DTYP_CHAR	See CHAR data type.

WinRegisterUserDatatype – Register User Data Type

DTYP_STRL	See PSZ data type.
DTYP_STR16	See STR16 data type.
DTYP_STR32	See STR32 data type.
DTYP_STR64	See STR64 data type.
DTYP_STR8	See STR8 data type.
Structure Data Types:	
DTYP_ACCEL	See ACCEL data type.
DTYP_ACCELTABLE	See ACCELTABLE data type.
DTYP_ARCPARAMS	See ARCPARAMS data type.
DTYP_AREABUNDLE	See AREABUNDLE data type.
DTYP_BITMAPINFO	See BITMAPINFO data type.
DTYP_BITMAPINFOHEADER	See BITMAPINFOHEADER data type.
DTYP_BTNCDATA	See BTNCDATA data type.
DTYP_CATCHBUF	See CATCHBUF data type.
DTYP_CHARBUNDLE	See CHARBUNDLE data type.
DTYP_CLASSINFO	See CLASSINFO data type.
DTYP_CREATESTRUCT	See CREATESTRUCT data type.
DTYP_CURSORINFO	See CURSORINFO data type.
DTYP_DEVOPENSTRUC	See DEVOPENSTRUC data type.
DTYP_DLGTEMPLATE	See DLGTEMPLATE data type.
DTYP_DLGTITEM	See DLGTITEM data type.
DTYP_ENTRYFDATA	See ENTRYFDATA data type.
DTYP_FATTRS	See FATTRS data type.
DTYP_FFDESCS	See FFDESCS data type.
DTYP_FIXED	See FIXED data type.
DTYP_FONTMETRICS	See FONTMETRICS data type.
DTYP_FRAMECDATA	See FRAMECDATA data type.
DTYP_GRADIENTL	See GRADIENTL data type.
DTYP_HCINFO	See HCINFO data type.
DTYP_IMAGEBUNDLE	See IMAGEBUNDLE data type.
DTYP_KERNINGPAIRS	See KERNINGPAIRS data type.
DTYP_LINEBUNDLE	See LINEBUNDLE data type.
DTYP_MARGSTRUCT	See MLEMARGSTRUCT data type.
DTYP_MARKERBUNDLE	See MARKERBUNDLE data type.
DTYP_MATRIXLF	See MATRIXLF data type.
DTYP_MLECTLDATA	See MLECTLDATA data type.
DTYP_OVERFLOW	See MLEOVERFLOW data type.
DTYP_OWNERITEM	See OWNERITEM data type.
DTYP_POINTERINFO	See POINTERINFO data type.

DTYP_POINTL	See POINTL data type.
DTYP_PROGRAMENTRY	See PROGRAMENTRY data type.
DTYP_PROGTYPE	See PROGTYPE data type.
DTYP_QMSG	See QMSG data type.
DTYP_RECTL	See RECTL data type.
DTYP_RGB	See RGB data type.
DTYP_RGNRECT	See RGNRECT data type.
DTYP_SBCDATA	See SBCDATA data type.
DTYP_SIZEF	See SIZEF data type.
DTYP_SIZEL	See SIZEF data type.
DTYP_SWBLOCK	See SWBLOCK data type.
DTYP_SWCNTRL	See SWCNTRL data type.
DTYP_SWENTRY	See SWENTRY data type.
DTYP_SWP	See SWP data type.
DTYP_TRACKINFO	See TRACKINFO data type.
DTYP_USERBUTTON	See USERBUTTON data type.
DTYP_WNDPARAMS	See WNDPARAMS data type.
DTYP_WPOINT	See WPOINT data type.
DTYP_WRECT	See WRECT data type.
DTYP_XYWINSIZE	See XYWINSIZE data type.
Pointer Data Types:	
DTYP_Pxxxx	Pointer to an item of data type DTYP_xxxx, where DTYP_xxxx is one of the data types in the preceding lists. The value of a pointer data type is the value of the corresponding non-pointer data type prefixed with minus to make it negative.
Minimum Application Data Type:	
DTYP_USER	Minimum value for application-defined non-pointer data type.
Control Data Types:	
DTYP_CTL_ARRAY	This starts a sequence of three array elements 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.

Element	Value
n	DTYP_CTL_ARRAY
n+1	data type of array data.
n+2	minus the number of elements in the array (for an array of fixed size), or the index of the element in *asTypes* corresponding to the structure component which contains the number of elements in the array being defined; this component must have a suitable numeric data type; the array-size element must precede element 'n' in *asTypes*. The index is zero-based.

DTYP_CTL_LENGTH

This starts a sequence of four array elements that define a structure component containing the length of part or all of the structure. The length component resides at this point in the structure.

Element	Value
n	DTYP_CTL_LENGTH
n+1	data type of structure component that contains the length (must be a suitable numeric data type).
n+2	the index of the element in *asTypes* corresponding to the first structure component that is included in the length; a value of −1 denotes the start of the structure. The index is zero-based. The element specified must not be one that is the second or subsequent element in a DTYP_CTL_* sequence of elements.
n+3	the index of the element in *asTypes* 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.

Element	Value
n	DTYP_CTL_OFFSET
n+1	data type of the structure component that contains the offset (must be a suitable unsigned numeric datatype).
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 *asTypes* corresponding to the structure component that contains the number of elements in the array being defined; this component must have a suitable numeric data type; the array-size element may occur earlier or later in the structure. The index is zero-based. A value of −1 indicates that the data is not an array.

DTYP_CTL_PARRAY This starts a sequence of three array elements that define a pointer to an array; the pointer resides at this point in the structure, the array resides elsewhere. The array may have a fixed or variable number of elements.

Element	Value
n	DTYP_CTL_PARRAY
n+1	data type of array data.
n+2	minus the number of elements in the array (for an array of fixed size), or the index of the element in *asTypes* 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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_DATATYPE_TOO_SMALL	The datatype specified was too small.
PMERR_DATATYPE_NOT_UNIQUE	An attempt to register a datatype failed because it is not unique.
PMERR_ARRAY_TOO_SMALL	The array specified was too small.
PMERR_DATATYPE_ENTRY_NOT_NUM	The datatype entry specified was not numerical.
PMERR_DATATYPE_ENTRY_NOT_OFF	The datatype entry specified was not an offset.
PMERR_DATATYPE_ENTRY_BAD_INDEX	An invalid datatype entry index was specified.
PMERR_DATATYPE_ENTRY_CTL_MISS	The datatype entry control was missing.
PMERR_DATATYPE_ENTRY_CTL_BAD	An invalid datatype entry control was specified.

Remarks

This function has no effect unless the RegisterUserMsg hook, which is invoked by this function, 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.

WinRegisterUserDatatype – Register User Data Type

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserMsg
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

Example Code

This example calls WinRegisterClass 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 — Register User Message

```
#define INCL_WINMESSAGEMGR  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinRegisterUserMsg (HAB hab, ULONG ulMsgId, LONG lType1, LONG lDir1,
                         LONG lType2, LONG lDir2, LONG lTyper)
```

This function registers a user message and defines its parameters.

Parameters

hab (HAB) — input
Anchor-block handle.

ulMsgid (ULONG) — input
Message identifier.

This must not be less than WM_USER, and must not have been defined previously.

lType1 (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 USHORT data type.
DTYP_BIT32	See ULONG data type.
DTYP_BIT8	See UCHAR 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.

lDir1 (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).

lType2 (LONG) — input
Data type of message parameter 2.

See the description of *lType1*.

WinRegisterUserMsg — Register User Message

lDir2 (LONG) — input
Direction of message parameter 2.

See the description of *lDir1*.

lTyper (LONG) — input
Data type of message reply.

See the description of *lType1*. The message reply is always an output parameter.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_MSGID_TOO_SMALL	The message identifier specified is too small.
PMERR_DATATYPE_INVALID	An invalid datatype was specified.
PMERR_DATATYPE_TOO_LONG	The datatype specified was too long.

Remarks

This function has no effect unless the RegisterUserMsg 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
- WinRegisterUserDatatype
- 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 – Release Hook

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinReleaseHook (HAB hab, HMQ hmq, LONG lHook, PFN pAddress,
                     HMODULE Module)
```

This function releases an application hook from a hook chain.

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 must be one of the HK_* values; see WinSetHook.

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 DosGetModHandle call.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HMQ An invalid message-queue handle was specified.

PMERR_PARAMETER_OUT_OF_RANGE 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 –
Release Presentation Space

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
BOOL WinReleasePS (HPS hps)
```

This function releases a cache presentation space obtained using the WinGetPS or the WinGetScreenPS call.

Parameters

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

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 — Remove Presentation Parameter

```
#define INCL_WINSYS  /* Or use INCL_WIN or INCL_PM */

BOOL WinRemovePresParam (HWND hwnd, ULONG idAttrType)
```

This function removes a presentation parameter associated with the window *hwnd*.

Parameters

hwnd (HWND) — input
Window handle.

idAttrType (ULONG) — input
Attribute type identity.

The type of the presentation parameter attribute that is to be removed.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 – Remove Switch Entry

```
#define INCL_WINSWITCHLIST   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
USHORT WinRemoveSwitchEntry (HSWITCH hswitchSwitch)
```

This function removes a specified entry from the Window List.

Parameters

hswitchSwitch (HSWITCH) – input
Switch-list (Window List) entry handle.

Returns

Success indicator:

0	Successful completion
Other	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_SWITCH_HANDLE	An invalid Window List entry handle was specified.
PMERR_INVALID_WINDOW	The window specified with a Window List function 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 — Replace Workplace Object Class

#define INCL_WINWORKPLACE

```
BOOL WinReplaceObjectClass (PSZ pszOldClassName, PSZ pszNewClassName,
                            BOOL fReplace)
```

The WinReplaceObjectClass function replaces a registered class with another registered class. If *fReplace* is FALSE, *pszOldClassName* will revert back to its original definition.

Parameters

pszOldClassName (PSZ) — input

A pointer to a zero-terminated string which contains the name of the object class being replaced by *pszNewClassName* in the workplace.

pszNewClassName (PSZ) — input

A pointer to a zero-terminated string which contains the name of the object class replacing the *pszOldClassName* class.

fReplace (BOOL) — input

TRUE Replace the function of class *pszOldClassName* with the function of the class *pszNewClassName*.

FALSE Undo the replacement of the *pszOldClassName* with *pszNewClassName* by restoring the *pszOldClassName* back to its original functionality.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

The class specified by *pszNewClassName* must be a descendant of the class specified by *pszOldClassName*, 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 *pszOldClassName* and which are not aware of the class *pszNewClassName*.

Related Functions

- WinCreateObject
- WinDeregisterObjectClass
- WinRegisterObjectClass

WinRequestMutexSem – Request Mutex Semaphore

#define INCL_WINMESSAGEMGR

ULONG WinRequestMutexSem (HMTX hmtx, ULONG ulTimeout)

WinRequestMutexSem requests ownership of a mutex semaphore or waits for a Presentation Manager message.

Parameters

hmtx (HMTX) – input

The handle of the mutex semaphore to request.

ulTimeout (ULONG) – input

The 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:

Value	Definition
0	(SEM_IMMEDIATE_RETURN) WinRequestMutexSem returns immediately without blocking the calling thread.
-1	(SEM_INDEFINITE_WAIT) WinRequestMutexSem blocks the calling thread indefinitely.

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.

WinRequestMutexSem – Request Mutex Semaphore

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;
     }
```

WinRestoreWindowPos — Restore Window Position

#define INCL_WINWORKPLACE

BOOL WinRestoreWindowPos (PSZ pszAppName, PSZ pszKeyName, HWND hwnd)

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 *pszAppName* and *pszKeyName*.

Parameters

pszAppName (PSZ) — input

A pointer to a zero-terminated string which contains the application name.

pszKeyName (PSZ) — input

A pointer to a zero-terminated string which contains the key name.

hwnd (HWND) — input

Window handle of the window to restore.

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

WinSaveWindowPos — Save Window Position

```
#define INCL_WINFRAMEMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSaveWindowPos (HSVWP hsvwp, PSWP aswpaswp, ULONG ccswp)

This function associates an array of SWP structures with the process of repositioning a frame window.

Parameters

hsvwp (HSVWP) — input
Identifier of the frame window repositioning process.

This handle is provided in the second parameter of the WM_ADJUSTFRAMEPOS message.

aswpaswp (PSWP) — input
Array of SWP structures.

ccswp (ULONG) — input
Count of SWP structures.

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 — Scroll Window

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
LONG WinScrollWindow (HWND hwnd, LONG lDx, LONG lDy, PRECTL prclScroll,
                      PRECTL prclClip, HRGN hrgnUpdateRgn, PRECTL prclUpdate,
                      ULONG flOptions)
```

This function scrolls the contents of a window rectangle.

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 –32 768 through 32 767. 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) — input/output
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

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	An invalid window handle was specified.
PMERR_INVALID_FLAG	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	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

WinScrollWindow — Scroll Window

Related Messages

- WM_ERASEWINDOW
- 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 — Send Message to Dialog Item

```
#define INCL_WINDIALOGS   /* Or use INCL_WIN or INCL_PM */
```

```
MRESULT WinSendDlgItemMsg (HWND hwndDlg, ULONG idItem, ULONG ulMsgId,
                           MPARAM mpParam1, MPARAM mpParam2)
```

This function sends a message to the dialog item defined by *idItem* in the dialog window specified by *hwndDlg*.

Parameters

hwndDlg (HWND) — input
Parent-window handle.

idItem (ULONG) — input
Identity of the child window.

ulMsgId (ULONG) — input
Message identity.

mpParam1 (MPARAM) — input
Message parameter 1.

mpParam2 (MPARAM) — input
Message parameter 2.

Returns

Message-return data.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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

WinSendDlgItemMsg – Send Message to Dialog Item

- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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;
}
```

#define INCL_WINMESSAGEMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

```
MRESULT WInSendMsg (HWND hwnd, ULONG ulMsgId, MPARAM mpParam1,
                    MPARAM mpParam2)
```

This function sends a message with identity *ulMsgid* to *hwnd*, passing *mpParam1* and *mpParam2* as the parameters to the window.

Parameters

hwnd (HWND) – input
Window handle.

ulMsgId (ULONG) – input
Message identity.

mpParam1 (MPARAM) – input
Parameter 1.

mpParam2 (MPARAM) – input
Parameter 2.

Returns

Message-return data.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_WINDOW_NOT_LOCKED	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.

WinSendMsg – Send Message

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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 — Set Accelerator Table

```
#define INCL_WINACCELERATORS  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSetAccelTable (HAB hab, HACCEL haccelAccel, HWND hwndFrame)

This function sets the window-accelerator, or queue-accelerator table.

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

Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_HACCEL	An invalid accelerator-table handle was specified.

Related Functions

- WinCopyAccelTable
- WinCreateAccelTable
- WinDestroyAccelTable
- WinLoadAccelTable
- WinQueryAccelTable
- WinTranslateAccel

WinSetAccelTable — Set Accelerator Table

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 — Set Active Window

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinSetActiveWindow (HWND hwndDeskTop, HWND hwnd)

This function makes the frame window the active window.

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

Active-window-set indicator:

TRUE Active window is set

FALSE Active window is not set.

Possible returns from WinGetLastError

PMERR_INVALID_HWND An invalid window handle was specified.

Remarks

This function is equivalent to the WinFocusChange function in which the *flFocusChange* parameter is set to FC_SETACTIVEFOCUS.

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 — Set Capture

```
#define INCL_WININPUT   /* Or use INCL_WIN or INCL_PM */

BOOL WinSetCapture (HWND hwndDesktop, HWND hwnd)
```

This function captures all pointing device messages.

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

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 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 — Set Class Message Interest

#define INCL_WINMESSAGEMGR /* Or use INCL_WIN or INCL_PM */

```
BOOL WinSetClassMsgInterest (HAB hab, PSZ pszClassName, ULONG ulMsgClass,
                             LONG lControl)
```

This function sets the message interest of a window class.

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.

WinSetClassMsgInterest – Set Class Message Interest

Returns

Interest-changed indicator:

TRUE Interest successfully changed

FALSE Interest not successfully changed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND An invalid window handle was specified.

Remarks

This function has no effect unless the MsgCtlHook 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
- WinRegisterUserDatatype
- WinRegisterUserMsg
- WinSendDlgItemMsg
- WinSendMsg
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode
- WinWaitMsg

WinSetClassMsgInterest — Set Class Message Interest

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 — Set Class Pointer-Conversion Procedure

```
#define INCL_WINTHUNKAPI  /* Or use INCL_WIN or INCL_PM */

BOOL WinSetClassThunkProc (PSZ pszClassName, PFN pthunkpr)
```

This function associates a pointer-conversion procedure with a window class.

Parameters

pszClassName (PSZ) — input
Window-class name.

pthunkpr (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

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

WinSetClassThunkProc –
Set Class Pointer-Conversion Procedure

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 — Set Clipboard Data

```
#define INCL_WINCLIPBOARD   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSetClipbrdData (HAB hab, ULONG ulh, ULONG ulfmt, ULONG flFmtInfo)

This call puts data into the clipboard.

Parameters

hab (HAB) — input
Anchor-block handle.

ulh (ULONG) — input
Handle.

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
Format.

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:

WinSetClipbrdData – Set Clipboard Data

- 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.

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 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 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 — Set Clipboard Owner

#define INCL_WINCLIPBOARD /* Or use INCL_WIN or INCL_PM */

```
BOOL WinSetClipbrdOwner (HAB hab, HWND hwnd)
```

This function sets the current clipboard-owner window.

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

Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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_RENDERFMT
WM_RENDERALLFMTS
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 — Set Clipboard Viewer

```
#define INCL_WINCLIPBOARD   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSetClipbrdViewer (HAB hab, HWND hwndNewClipViewer)

This function sets the current clipboard-viewer window to a specified window.

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

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	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 –
Set Code Page

```
#define INCL_WINCOUNTRY  /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinSetCp (HMQ hmq, ULONG ulCodePage)
```

This function sets the code page for a queue.

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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HMQ	An invalid message-queue handle was specified.
PMERR_RESOURCE_NOT_FOUND	The specified resource identity could not be found.

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 — Set Desktop Background

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM */

HBITMAP WinSetDesktopBkgnd (HWND hwndDeskTop, PDESKTOP pDeskTopState)

This function sets the desktop window state.

Parameters

hwndDeskTop (HWND) — input
Desktop-window handle:

pDeskTopState (PDESKTOP) — input
Desktop-state structure.

If the *fl* parameter has the SDT_LOADFILE flag set then the *szFile*[*MAX_FILENAME*] 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

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	An invalid pointer handle was specified.
PMERR_INVALID_HWND	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.

WinSetDesktopBkgnd — Set Desktop Background

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 — Set Dialog Item Short

#define INCL_WINDIALOGS /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinSetDlgItemShort (HWND hwndDlg, ULONG idItem, USHORT usValue, BOOL fSigned)

This function converts an integer value into the text of a dialog item.

Parameters

hwndDlg (HWND) — input
Parent-window handle.

idItem (ULONG) — input
Identity of the child window whose text is to be changed.

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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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

WinSetDlgItemShort – Set Dialog Item Short

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 – Set Dialog Item Text

#define INCL_WINDIALOGS /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinSetDlgItemText (HWND hwndDlg, ULONG idItem, PSZ pszText)

This function sets a text string in a dialog item.

Parameters

hwndDlg (HWND) – input
Parent-window handle.

idItem (ULONG) – input
Identity of the child window whose text is to be set.

pszText (PSZ) – input
Source string.

This is the text string that is to be set into the dialog item.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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

WinSetDlgItemText — Set Dialog Item Text

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;

    WinQuerySetDlgItemText(hwndDlg,
                           ID_DLG_CALENDAR,
                           "CALENDAR");
```

WinSetFileIcon — Set File Icon

#define INCL_WINWORKPLACE

BOOL WinSetFileIcon (PSZ pszFileName, PICONINFO picon)

The WinSetFileIcon function will set the icon on the file specified by *pszFileName* to be that specified by *picon*.

Parameters

pszFileName (PSZ) — input

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

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 — Set Focus

```
#define INCL_WININPUT   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinSetFocus  (HWND  hwndDeskTop, HWND  hwndNewFocus)
```

This function sets the focus window.

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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 */
                                           /* window.        */

}
```

WinSetHook – Set Hook

```
#define INCL_WINHOOKS   /* Or use INCL_WIN or INCL_PM */

BOOL WinSetHook (HAB hab, HMQ hmq, LONG lHookType, PFN pHookProc,
                 HMODULE Module)
```

This function installs an application procedure into a specified hook chain.

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 CodePageChangeHook.
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 MsgCtlHook.
HK_MSGFILTER	See MsgFilterHook.
HK_REGISTERUSERMSG	See RegisterUserMsg.
HK_SENDMSG	See SendMsgHook.

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 DosGetModHandle 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

Success indicator:

TRUE Successful completion

FALSE An error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HMQ	An invalid message-queue handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE	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 — Set Keyboard State Table

```
#define INCL_WININPUT   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinSetKeyboardStateTable (HWND hwndDeskTop, PBYTE abKeyStateTable, BOOL fSet)
```

This function gets or sets the keyboard state.

Parameters

hwndDeskTop (HWND) — input
Desktop-window handle:

HWND_DESKTOP The desktop-window handle

Other Specified desktop-window handle.

abKeyStateTable (PBYTE) — input/output
Key state table.

This is a 256-byte table indexed by virtual key value.

For any virtual key, the 0x80 bit is set if the key is down, and zero if it is up. The 0x01 bit is set if the key is toggled, (pressed an odd number of times), otherwise it is zero.

fSet (BOOL) — input
Set indicator:

TRUE The keyboard state is set from *abKeyStateTable*

FALSE The keyboard state is copied to *abKeyStateTable*.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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[257]; /* 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 — Set Listbox Item Text

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSetLboxItemText (HWND hwndLbox, SHORT sLboxIndx, PSZ pszText)

This macro sets the text of the list box indexed item to buffer.

Parameters

hwndLbox (HWND) — input
List box handle.

sLboxIndx (SHORT) — input
Index of the list box item.

pszText (PSZ) — input
Pointer to a null terminated string.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This macro expands to:

```
#define WinSetLboxItemText(hwndLbox, sLboxIndx, pszText)
  ((BOOL)WinSendMsg(hwndLbox,
                    LM_SETITEMTEXT,
                    MPFROMSHORT(sLboxIndx),
                    MPFROMP(pszText)))
```

This function 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;
SHORT 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++)
{
    WinQuerySetDlgItemText(hwndLbox,
                           i,
                           months[i]);
} /* endfor */
```

WinSetMenuItemText — Set Menu Item Text

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSetMenuItemText (HWND hwndMenu, USHORT usId, PSZ pszText)

This macro sets the text for Menu indexed item to buffer.

Parameters

hwndMenu (HWND) — input
Menu window handle.

usId (USHORT) — input
Identity of the menu item.

pszText (PSZ) — input
Text for the menu item.

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 — Set Message Interest

```
#define INCL_WINMESSAGEMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSetMsgInterest (HWND hwnd, ULONG ulMsgClass, LONG lControl)

This function sets a window's message interest.

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

Interest-changed indicator:

TRUE	Interest successfully changed
FALSE	Interest not successfully changed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.

Remarks

This function has no effect unless the MsgCtlHook 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.

WinSetMsgInterest — Set Message Interest

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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 – Set Message Mode

```
#define INCL_WINMESSAGEMGR   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
BOOL WinSetMsgMode (HAB hab, PSZ pszClassName, LONG lControl)
```

This function indicates the mode for the generation and processing of messages for the private window class of an application.

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

Message delay indicator:

TRUE Message mode successfully set

FALSE Message mode not successfully set.

Possible returns from WinGetLastError

PMERR_INVALID_HWND An invalid window handle was specified.

Remarks

This function has no effect unless the MsgCtlHook 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
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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 — Set Multiple Window Positions

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinSetMultWindowPos (HAB hab, PSWP aSwp, ULONG cCount)
```

This function performs the WinSetWindowPos function for *cCount* windows, using *aSwp*, an array of structures whose elements correspond to the input parameters of WinSetWindowPos.

Parameters

hab (HAB) — input
Anchor-block handle.

aSwp (PSWP) — input
Array.

An array of set window position (SWP) structures. The elements of each correspond to the input parameters of WinSetWindowPos.

cCount (ULONG) — input
Window count.

Returns

Positioning success indicator:

TRUE Positioning succeeded

FALSE Positioning failed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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 – Set Object Data

```
#define INCL_WINWORKPLACE
```

BOOL WinSetObjectData (HOBJECT object, PSZ pszSetupString)

The WinSetObjectData function is called to set data on a workplace object.

Parameters

object (HOBJECT) – input

Handle to a workplace object.

pszSetupString (PSZ) – input

A pointer to a zero-terminated string which contains the object-specific parameters to the new object.

The *pszSetupString* string is extracted when the wpSetup method is called.

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.

Related Functions

- WinCreateObject
- WinDestroyObject

WinSetOwner – Set Owner

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM */

BOOL WinSetOwner (HWND hwnd, HWND hwndNewOwner)

This function changes the owner window of a specified window.

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

Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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 – Set Parent

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSetParent (HWND hwnd, HWND hwndNewParent, BOOL fRedraw)

This function sets the parent for *hwnd* to *hwndNewParent*.

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

Parent-changed indicator:

TRUE Parent successfully changed

FALSE Parent not successfully changed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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_ERASEWINDOW
- 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 — Set Pointer

```
#define INCL_WINPOINTERS  /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinSetPointer (HWND hwndDeskTop, HPOINTER hptrNewPointer)
```

This call sets the desktop-pointer handle.

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

Pointer-updated indicator:

TRUE Pointer successfully updated

FALSE Pointer not successfully updated.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_HPTR	An invalid pointer handle was specified.
PMERR_INV_CURSOR_BITMAP	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
- WinSetPointerPos
- 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);

}
```

WinSetPointerPos – Set Pointer Position

#define INCL_WINPOINTERS /* Or use INCL_WIN or INCL_PM */

BOOL WinSetPointerPos (HWND hwndDeskTop, LONG lx, LONG ly)

This function sets the pointer position.

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

Pointer position updated indicator:

TRUE Pointer position successfully updated

FALSE Pointer position not successfully updated.

Possible returns from WinGetLastError

PMERR_INVALID_HWND An invalid window handle was specified.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinSetPointer
- 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 – Set Presentation Parameter

```
#define INCL_WINSYS   /* Or use INCL_WIN or INCL_PM */

BOOL WinSetPresParam (HWND hwnd, ULONG idAttrType, ULONG cbAttrValueLen,
                      PVOID pAttrValue)
```

This function sets a presentation parameter for a window.

Parameters

hwnd (HWND) – input
Window handle.

idAttrType (ULONG) – input
Attribute type identity.

This is either one of the system-defined presentation parameter attribute types (see the *id* parameter of the PARAM data type), or an application-defined type.

cbAttrValueLen (ULONG) – input
Byte count of the data passed in the *pAttrValue* parameter.

pAttrValue (PVOID) – input
Attribute value.

See the *abab*[*1*] parameter of the PARAM data type for the values of system-defined attributes.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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.

Related Functions

- WinDrawBitmap
- WinDrawBorder
- WinDrawPointer
- WinDrawText
- WinFillRect
- WinGetSysBitmap
- WinInvertRect
- WinQueryPresParam
- WinRemovePresParam
- WinScrollWindow

WinSetPresParam — Set Presentation Parameter

Example Code

This example changes the border color to blue.

```
#define INCL_WINSYS
#define INCL_GPIBITMAPS  /* for RGB structure definition. */
#include <OS2.H>
HWND hwnd;
RGB rgb;   /* red, green, and blue color index. */
rgb.bBlue = 200;
rgb.bGreen = 10;
rgb.bRed = 5;


WinSetPresParam(hwnd,
                PP_BORDERCOLOR,
               (ULONG)sizeof(RGB),
               (PVOID)&rgb);
```

WinSetRect – Set Rectangle

#define INCL_WINRECTANGLES /* Or use INCL_WIN or INCL_PM */

BOOL WinSetRect (HAB hab, PRECTL prcIrect, LONG lLeft, LONG lBottom, LONG lRight, LONG lTop)

This function sets rectangle coordinates.

Parameters

hab (HAB) – input
Anchor-block handle.

prcIrect (PRECTL) – input/output
Rectangle to be updated.

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.

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

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

WinSetRect — Set Rectangle

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 — Set Rectangle Empty

```
#define INCL_WINRECTANGLES  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSetRectEmpty (HAB hab, PRECTL prclrect)

This function sets a rectangle empty.

Parameters

hab (HAB) — input
Anchor-block handle.

prclrect (PRECTL) — input/output
Rectangle to be set empty.

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

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 – Set Synchronization Mode

```
#define INCL_WINMESSAGEMGR   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
BOOL WinSetSynchroMode (HAB hab, LONG lMode)
```

This function is intended for use in a distributed application.

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

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 MsgCtlHook hook which is invoked by this function.

Related Functions

- WinBroadcastMsg
- WinCreateMsgQueue
- WinDestroyMsgQueue
- WinDispatchMsg
- WinGetDlgMsg
- WinGetMsg
- WinInSendMsg
- WinPeekMsg
- WinPostMsg
- WinPostQueueMsg
- WinQueryMsgPos
- WinQueryMsgTime
- WinQueryQueueInfo
- WinQueryQueueStatus
- WinRegisterUserDatatype
- WinRegisterUserMsg
- 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 – Set System Colors

```
#define INCL_WINSYS  /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinSetSysColors (HWND hwndDeskTop, ULONG flOptions, ULONG ulFormat,
                      LONG lStart, ULONG ulTablen, PLONG alTable)
```

This function sets system color values.

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 pushbutton (3D effect).

SYSCLR_BUTTONMIDDLE
Middle pushbutton (3D effect).

SYSCLR_BUTTONDARK
Dark pushbutton (3D effect).

SYSCLR_BUTTONDEFAULT
Pushbutton.

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.

WinSetSysColors — Set System Colors

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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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	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 Appendix G.

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

WinSetSysColors – Set System Colors

System Color Index	Default Color	Default RGB Values
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
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].index = (R * 65536L) + (G * 256L) + B;
R = 5; G = 200; B = 5;
alTable[1].index = (R * 65536L) + (G * 256L) + B;
alTable[0].color = SYSCLR_OUTPUTTEXT; /* output text. */
alTable[1].color = 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 — Set System Modal Window

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSetSysModalWindow (HWND hwndDesktop, HWND hwnd)

This function makes a window become the system-modal window, or ends the system-modal state.

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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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);
```

WinSetSysValue — Set System Value

```
#define INCL_WINSYS  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSetSysValue (HWND hwndDeskTop, LONG lValueId, LONG lValue)

This function sets a system value.

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.

lValueId (LONG) — input
System-value identity.

This must be a valid SV_* value (see WinQuerySysValue). The following values can be set:

SV_CXSIZEBORDER
Width of the sizing border

SV_CYSIZEBORDER
Height of the sizing border

SV_SWAPBUTTON
TRUE when the mouse buttons are set for left-handed use

SV_CURSORRATE
Cursor blink rate, in milliseconds

SV_DBLCLKTIME
Mouse double-click time, in milliseconds

SV_CXDBLCLK
Width of the mouse double-click sensitive area

SV_CYDBLCLK
Height of the mouse double-click sensitive area

SV_ALARM
TRUE enables the alarm sound generated by WinAlarm; FALSE disables the alarm sound

SV_WARNINGFREQ
Frequency for warning alarms generated by WinAlarm

SV_WARNINGDURATION
Duration of 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
Delay (in milliseconds) before autoscrolling starts, when using a scroll bar

SV_SCROLLRATE
Delay (in milliseconds) between scroll operations, when using a scroll bar

SV_SETLIGHTS
When TRUE, the appropriate light is set when the keyboard state table is set.

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_PRINTSCREEN
TRUE when the Print Screen function is enabled; FALSE when the Print Screen function is disabled.

lValue (LONG) – input
The system value.

Dimensions are in pels and times are in milliseconds.

Returns

Value-set indicator:

TRUE System value set

FALSE An error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE	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);
```

WinSetWindowBits — Set Window Word Bits

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinSetWindowBits (HWND hwnd, LONG lb, ULONG flData, ULONG flMask)
```

This function sets a number of bits into the memory of the reserved window words.

Parameters

hwnd (HWND) — input
Window handle.

lb (LONG) — input
Zero-based index of the value to be set.

The units of **b** are bytes. Valid values are zero through (*usExtra* —4), where *usExtra* is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage. Any of the QWL_* values specified for the WinQueryWindowULong function can also be used.

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

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 — Set Window Position

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinSetWindowPos (HWND hwnd, HWND hwndBehind, LONG lx, LONG ly, LONG lcx,
                      LONG lcy, ULONG flOptions)
```

This function allows the general positioning of a window.

Parameters

hwnd (HWND) — input
Window handle.

hwndBehind (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.

lx (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.

ly (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.

lcx (LONG) — input
Window size.

This specifies the width of *hwnd* in device units, but is ignored if SWP_SIZE is not selected.

lcy (LONG) — input
Window size.

This specifies the depth of *hwnd* in device units, but is ignored if SWP_SIZE is not selected.

flOptions (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 *hwndBehind* 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 *hwndBehind* 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

Repositioning indicator:

TRUE Window successfully repositioned

FALSE Window not successfully repositioned.

Possible returns from WinGetLastError

PMERR_INVALID_HWND An invalid window handle was specified.

PMERR_INVALID_FLAG An invalid bit was set for a parameter. Use constants defined by PM for options, and do not set any reserved bits.

Remarks

Note: Messages may be received from other processes or threads during the processing of this function.

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.

WinSetWindowPos – Set Window Position

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_CALCVALIDRECTS	Sent to determine the area of a window that may be possible to preserve as the window is sized.
WM_SIZE	Sent if the size of the window has changed, after the change has been made.
WM_MOVE	Sent when a window with CS_MOVENOTIFY class style moves its absolute position.
WM_ACTIVATE	Sent if a different window becomes the active window. See also WinSetActiveWindow.
WM_ADJUSTWINDOWPOS	Not sent if SWP_NOADJUST is specified. The message contains an SWP structure that has been filled in by this 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.

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 — Set Window Words Pointer

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSetWindowPtr (HWND hwnd, LONG lb, PVOID pp)

This function sets a pointer value into the memory of the reserved window words.

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 (*usExtra* —4), where *usExtra* 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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE	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 — Set Window Text

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */

BOOL WinSetWindowText (HWND hwnd, PSZ pszString)
```

This function sets the window text for a specified window.

Parameters

hwnd (HWND) — input
Window handle.

pszString (PSZ) — input
Window text.

Returns

Success indicator:

TRUE Text updated

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 Chapter 13, "Button Control Window Processing" on page 13-1 and Chapter 17, "Menu Control Window Processing" on page 17-1.

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 — Set Window Pointer-Conversion Procedure

---

```
#define INCL_WINTHUNKAPI  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSetWindowThunkProc (HWND hwnd, PFN pthunkpr)

This function associates a pointer-conversion procedure with a window.

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

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 — Set Window Word Long

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM */

BOOL WinSetWindowULong (HWND hwnd, LONG lb, ULONG ulData)

This function sets an unsigned, long integer value into the memory of the reserved window words.

Parameters

hwnd (HWND) — input
Window handle.

lb (LONG) — input
Zero-based index of the value to be set.

The units of *lb* are bytes. Valid values are zero through (*usExtra* —4), where *usExtra* is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage. So too are any of the QWL_* values, as specified for the WinQueryWindowULong function.

QWS_* values cannot be used.

ulData (ULONG) — input
Unsigned, long integer value to store in the window words.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_PARAMETER_OUT_OF_RANGE	The value of a parameter was not within the defined valid range for that parameter.

Remarks

The specified *lb* is valid only if all of the bytes referenced are within the reserved memory.

Related Functions

- WinQueryWindowPtr
- WinQueryWindowULong
- WinQueryWindowUShort
- WinSetWindowBits
- WinSetWindowPtr
- WinSetWindowUShort

WinSetWindowULong — Set Window Word Long

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 — Set Window Word Short

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM */

BOOL WinSetWindowUShort (HWND hwnd, LONG lb, USHORT usData)

This function sets an unsigned, short integer value into the memory of the reserved window words.

Parameters

hwnd (HWND) — input
Window handle.

lb (LONG) — input
Zero-based index of the value to be set.

The units of *lb* are bytes. Valid values are zero through (*usExtra* –2), where *usExtra* is the parameter in WinRegisterClass that specifies the number of bytes available for application-defined storage. So too are any of the QWS_* values, as specified for the WinQueryWindowUShort function.

QWL_* values cannot be used.

usData (USHORT) — input
Unsigned, short integer value to store in the window words.

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 — Show Cursor

```
#define INCL_WINCURSORS  /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
BOOL WinShowCursor (HWND hwnd, BOOL fShow)
```

This function shows or hides the cursor that is associated with a specified window.

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

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 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 — Show Pointer

```
#define INCL_WINPOINTERS   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinShowPointer (HWND hwndDeskTop, BOOL fShow)
```

This function adjusts the pointer display level to show or hide a pointer.

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

Display-level-updated indicator:

TRUE Pointer display level not successfully updated.

FALSE Pointer display level successfully updated

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 *lValueid* set to SV_POINTERLEVEL.

Related Functions

- WinCreatePointer
- WinCreatePointerIndirect
- WinDestroyPointer
- WinDrawPointer
- WinLoadPointer
- WinQueryPointer
- WinQueryPointerInfo
- WinQueryPointerPos
- WinQuerySysPointer
- WinSetPointer
- WinSetPointerPos

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 — Show Tracking Rectangle

```
#define INCL_WINTRACKRECT  /* Or use INCL_WIN or INCL_PM */

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

This function hides or shows the tracking rectangle.

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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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 — Show Window

#define INCL_WINWINDOWMGR /* Or use INCL_WIN or INCL_PM. Also in COMMON section */

BOOL WinShowWindow (HWND hwnd, BOOL fNewVisibility)

This function sets the visibility state of a window.

Parameters

hwnd (HWND) — input
Window handle.

fNewVisibility (BOOL) — input
New visibility state:

TRUE Set window state visible

FALSE Set window state invisible.

Returns

Visibility changed indicator:

TRUE Window visibility successfully changed

FALSE Window visibility not successfully changed.

Possible returns from WinGetLastError

PMERR_INVALID_HWND 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

WinShowWindow — Show Window

- 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 );
}
```

#define INCL_WINWORKPLACE

BOOL WinShutdownSystem (HAB hab, HMQ hmq)

The WinShutdownSystem function will close down the system.

Parameters

hab (HAB) – input

Anchor-block handle.

hmq (HMQ) – input

Message-queue handle.

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.

WinStartApp — Window Start Application

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

```
HAPP WinStartApp (HWND hwndNotify, PPROGDETAILS pDetails, PSZ pszParams,
                  PVOID pReserved, ULONG ulOptions)
```

This function starts an application.

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.

pszParams (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 *pszParams* 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

Application handle.

NULL Application not started

Other Application handle.

Possible returns from WinGetLastError

PMERR_INVALID_PARAMETERS	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	Attempted to start an application whose type is not recognized by OS/2.
PMERR_INVALID_WINDOW	The window specified with a Window List function is not a valid frame window.
PMERR_STARTED_IN_BACKGROUND	The application started a new session in the background.
PMERR_DOS_ERROR	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

WinStartApp –
Window Start Application

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 — Start Timer

```
#define INCL_WINTIMER   /* Or use INCL_WIN or INCL_PM */
```

ULONG WinStartTimer (HAB hab, HWND hwnd, ULONG idTimer, ULONG ulTimeout)

This function starts a timer.

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.

ulTimeout (ULONG) — input
Delay time in milliseconds.

Returns

Return code.

When *hwnd* is set to NULLHANDLE:

0 Error occurred

Other Timer identity.

Possible returns from WinGetLastError

PMERR_INVALID_HWND An invalid window handle was specified.

Remarks

This function creates a timer identified by *hwnd* and *idTimer*, set to time out every *ulTimeout* milliseconds. When a timer times out, a WM_TIMER message is posted.

A *ulTimeout* 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.

WinStartTimer — Start 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 — Stop Timer

```
#define INCL_WINTIMER   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinStopTimer (HAB hab, HWND hwnd, ULONG ulTimer)

This function stops a timer.

Parameters

hab (HAB) — input
Anchor-block handle.

hwnd (HWND) — input
Window handle.

ulTimer (ULONG) — input
Timer identifier.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred, or timer did not exist.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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

WinStopTimer — Stop Timer

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 — Store Window Position

#define INCL_WINWORKPLACE

BOOL WinStoreWindowPos (PSZ pszAppName, PSZ pszKeyName, HWND hwnd)

The WinStoreWindowPos function will save the current size and position of the window specified by *hwnd*.

Parameters

pszAppName (PSZ) — input

A pointer to a zero-terminated string which contains the application name.

pszKeyName (PSZ) — input

A pointer to a zero-terminated string which contains the key name.

hwnd (HWND) — input

Window handle for the window to be stored.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This function will also save the presentation parameters.

Related Functions

- WinRestoreWindowPos

WinSubclassWindow — Subclass Window

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

PFNWP WinSubclassWindow (HWND hwnd, PFNWP pNewWindowProc)

This function subclasses the indicated window by replacing its window procedure with another window procedure, specified by *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

Old window procedure.

Previous window procedure belonging to *hwnd*.

If this function fails, **0L** is returned.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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 — Substitute Strings

```
#define INCL_WINDIALOGS  /* Or use INCL_WIN or INCL_PM */
```

LONG WinSubstituteStrings (HWND hwnd, PSZ pszSrc, LONG lDestMax, PSZ pszDest)

This function performs a substitution process on a text string, replacing specific marker characters with text supplied by the application.

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*.

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

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	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 — Subtract Rectangle

```
#define INCL_WINRECTANGLES   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinSubtractRect (HAB hab, PRECTL prclDest, PRECTL prclSrc1, PRECTL prclSrc2)

This function subtracts one rectangle from another.

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 –32 768 through 32 767. 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 –32 768 through 32 767. 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 –32 768 through 32 767. The data type WRECT may also be used, if supported by the language.

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.

The GpiCombineRegion function 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 — Switch To Program

```
#define INCL_WINSWITCHLIST   /* Or use INCL_WIN or INCL_PM */

ULONG WinSwitchToProgram (HSWITCH hswitchSwHandle)
```

This function makes a specific program the active program.

Parameters

hswitchSwHandle (HSWITCH) — input
Window List entry handle of program to be activated.

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	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 — Terminate

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM. Also in COMMON section */
```

```
BOOL WinTerminate (HAB hab)
```

This function terminates an application thread's use of the Presentation Manager and releases all of its associated resources.

Parameters

hab (HAB) — input
Anchor-block handle.

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 – Terminate Application

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinTerminateApp (HAPP happ)
```

This function terminates an application previously started with the WinStartApp function.

Parameters

happ (HAPP) – input
Anchor-block handle.

Identifies the application to terminate.

Returns

Termination indicator:

TRUE Application successfully terminated

NULL Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HAPP	The application handle passed to WinTerminateApp does not correspond to a valid session.
PMERR_CANNOT_STOP	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 — Draw Tracking Rectangle

```
#define INCL_WINTRACKRECT  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinTrackRect (HWND hwnd, HPS hps, PTRACKINFO ptiTrackinfo)

This function draws a tracking rectangle.

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) — input/output
Track information.

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 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.

WinTrackRect – Draw Tracking Rectangle

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 — Translate Accelerator

```
#define INCL_WINACCELERATORS  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinTranslateAccel (HAB hab, HWND hwnd, HACCEL haccelAccel, PQMSG pQmsg)

This function translates a WM_CHAR message.

Parameters

hab (HAB) — input
Anchor-block handle.

hwnd (HWND) — input
Destination window.

haccelAccel (HACCEL) — input
Accelerator-table handle.

pQmsg (PQMSG) — input/output
Message to be translated.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_HACCEL	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 — Union Rectangle

```
#define INCL_WINRECTANGLES  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinUnionRect (HAB hab, PRECTL prclDest, PRECTL prclSrc1, PRECTL prclSrc2)

This function calculates a rectangle that bounds the two source rectangles.

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 —32 768 through 32 767. 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 —32 768 through 32 767. 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 —32 768 through 32 767. The data type WRECT may also be used, if supported by the language.

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);
```

WinUpdateWindow — Update Window

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

```
BOOL WinUpdateWindow (HWND hwnd)
```

This function forces the update of a window and its associated child windows.

Parameters

hwnd (HWND) — input
Window handle.

Returns

Window-updated indicator:

TRUE Window successfully updated

FALSE Window not successfully updated.

Possible returns from WinGetLastError

PMERR_INVALID_HWND An invalid window handle was specified.

Remarks

If *hwnd* is an asynchronous window, 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 and not synchronously.

If *hwnd* is a child of a nonclip-children parent, the update region of *hwnd* is subtracted from the update region of the parent, if the parent has one. This is so that any parent-window drawing after *hwnd* does not draw over whatever is drawn by *hwnd*.

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 — Uppercase String

```
#define INCL_WINCOUNTRY   /* Or use INCL_WIN or INCL_PM */
```

```
ULONG WinUpper (HAB hab, ULONG ulCodepage, ULONG ulCountry, PSZ pszString)
```

This function converts a string to uppercase.

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) — input/output
String to be converted to uppercase.

Returns

Length of converted string.

Possible returns from WinGetLastError

PMERR_INVALID_STRING_PARM The specified string parameter is invalid.

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 — Uppercase Character

```
#define INCL_WINCOUNTRY   /* Or use INCL_WIN or INCL_PM */
```

ULONG WinUpperChar (HAB hab, ULONG ulCodepage, ULONG ulCountry, ULONG ulInchar)

This function translates a character to uppercase.

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.

ulInchar (ULONG) — input
Character to be translated to uppercase.

Returns

Translated character:

0 Error occurred

Other The translated character.

Possible returns from WinGetLastError

PMERR_INVALID_STRING_PARM 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 — Validate Rectangle

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

BOOL WinValidateRect (HWND hwnd, PRECTL prclRect, BOOL fIncludeClippedChildren)

This function subtracts a rectangle from the update region of an asynchronous paint window, marking that part of the window as visually valid.

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 –32 768 through 32 767. 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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_INVALID_FLAG	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 — Validate Region

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

BOOL WinValidateRegion (HWND hwnd, HRGN hrgn, BOOL fIncludeClippedChildren)

This function subtracts a region from the update region of an asynchronous paint window, marking that part of the window as visually valid.

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

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	An invalid window handle was specified.
PMERR_HRGN_BUSY	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	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

WinValidateRegion — Validate Region

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 – Wait Event Semaphore

#define INCL_WINMESSAGEMGR

ULONG WinWaitEventSem (HEV hev, ULONG ulTimeout)

WinWaitEventSem waits for an event semaphore to be posted or for a Presentation Manager message.

Parameters

hev (HEV) – input

The handle of the event semaphore to wait for.

ulTimeout (ULONG) – input

The 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:

Value	Definition
0	(SEM_IMMEDIATE_RETURN) WinWaitEventSem returns without blocking the calling thread.
–1	(SEM_INDEFINITE_WAIT) WinWaitEventSem blocks the calling thread indefinitely.

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 *Timeout* parameter, this function may not return within the time specified by that value.

Related Functions

- WinSendMsg
- WinPostMsg

WinWaitEventSem – Wait Event Semaphore

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 — Wait Message

#define INCL_WINMESSAGEMGR /* Or use INCL_WIN or INCL_PM */

BOOL WinWaitMsg (HAB hab, ULONG ulFirst, ULONG ulLast)

This function waits for a filtered message.

Parameters

hab (HAB) — input
Anchor-block handle.

ulFirst (ULONG) — input
First message identity.

ulLast (ULONG) — input
Last message identity.

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 *fSuccess* 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
- WinRegisterUserDatatype
- WinRegisterUserMsg
- WinSendDlgItemMsg
- WinSendMsg
- WinSetClassMsgInterest
- WinSetMsgInterest
- WinSetMsgMode
- WinSetSynchroMode

WinWaitMsg — Wait Message

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 – Wait MuxWait Semaphore or Message

#define INCL_WINMESSAGEMGR

ULONG WinWaitMuxWaitSem (HMUX hmux, ULONG ulTimeout, PULONG pUser)

WinWaitMuxWaitSem waits for a muxwait semaphore to clear or for a Presentation Manager message.

Parameters

hmux (HMUX) – input

The handle of the muxwait semaphore to wait for.

ulTimeout (ULONG) – input

The 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:

Value	Definition
0	(SEM_IMMEDIATE_RETURN) WinWaitMuxWaitSem returns without blocking the calling thread.
–1	(SEM_INDEFINITE_WAIT) WinWaitMuxWaitSem blocks the calling thread indefinitely.

pUser (PULONG) – output

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

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

WinWaitMuxWaitSem – Wait MuxWait Semaphore or Message

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

- WinSendMsg
- WinPostMsg

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 –
Query Window Handle From Device Context

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

HWND WinWindowFromDC (HDC hdc)

This function returns the handle of the window corresponding to a particular device context.

Parameters

hdc (HDC) – input
Device-context handle.

The device context must first be opened by the WinOpenWindowDC function.

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 An invalid presentation-space handle was specified.

PMERR_INV_HDC 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 –
Query Window Handle From Identifier

```
#define INCL_WINWINDOWMGR   /* Or use INCL_WIN or INCL_PM */
```

HWND WinWindowFromID (HWND hwndParent, ULONG ulIdentifier)

This function returns the handle of the child window with the specified identity.

Parameters

hwndParent (HWND) – input
Parent-window handle.

ulIdentifier (ULONG) – input
Identity of the child window.

Returns

Window handle:

NULLHANDLE	No child window of the specified identity exists
Other	Child-window handle.

Possible returns from WinGetLastError

PMERR_INVALID_HWND	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 *ulIdentifier* 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 — Window From Point

```
#define INCL_WINWINDOWMGR  /* Or use INCL_WIN or INCL_PM */
```

HWND WinWindowFromPoint (HWND hwndParent, PPOINTL pptlPoint, BOOL fEnumChildren)

This function finds the window below a specified point, that is a descendant of a specified window.

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

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 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 9. Workplace Classes, Instance Methods, and Class Methods

The following sections contain Workplace classes, instance methods, and class methods. Each section is in alphabetical order.

These sections contain technical reference information. See the workplace chapter in the Application Design Guide for workplace guide information. For information on the System Object Model (SOM), see *System Object Model Guide and Reference.*

The following is some terminology used in these sections:

class — A way of categorizing objects based on their behavior and shape. A class is, in effect, a definition of a generic object. In SOM, a class is a special kind of object that can manufacture other objects that all have a common shape and exhibit similar behavior (more precisely, all of the objects manufactured by a class have the same memory layout and share a common set of methods). New classes can be defined in terms of existing classes through a technique known as *inheritance.*

class method — A class method of class <X> is a method provided by the metaclass of class <X>. Class methods are executed without requiring any instances of class <X> to exist, and are frequently used to create instances.

inheritance — The technique of specifying the shape and behavior of one class (called a *subclass*) as incremental differences from another class (called the *parent class* or *superclass*). The subclass inherits the superclass's state representation and methods, and can provide additional data elements and methods. The subclass also can provide new functions with the same method names used by the superclass. Such a subclass method is said to override the superclass method, and will be selected automatically by method resolution on subclass instances. An overriding method can elect to call upon the superclass's method as part of its own implementation.

instance — (Or object instance). A specific object, as distinguished from the abstract definition of an object referred to as its class.

instance method — A method valid for a particular object.

in-use emphasis — A visual queue around an object's icon to indicate that one or more view windows are currently open.

in-use list — A linked list of items representing resources allocated on an object's behalf.

metaclass — A class whose instances are all classes. In SOM, any class descended from SOMClass is a metaclass. The methods of a metaclass are sometimes called "class" methods.

method — One of the units that makes up the behavior of an object. A method is a combination of a function and a name, such that many different functions can have the same name. Which function the name refers to at any point in time depends on the object that is to execute the method and is the subject of method resolution.

object — The elements of data and function that programs create, manipulate, pass as arguments, and so forth. An object is a way of associating specific data values with a specific set of named functions (called *methods*) for a period of time (referred to as the *lifetime* of the object). The data values of an object are referred to as its *state*. In SOM, objects are created by other objects called *classes*. The specification of what comprises the set of functions and data elements that make up an object is referred to as the *definition* of a class.

SOM objects offer a high degree of *encapsulation*. This property permits many aspects of the implementation of an object to change without affecting client programs that depend on the object's behavior.

object definition — See *class*.

object instance — See *instance*.

subclass — A class that inherits from another class. See *inheritance*.

superclass — A class from which another class inherits. See *inheritance*.

Workplace object classes

Workplace objects are icons representative of physical objects which users deal with in the workplace. A Workplace object is implemented as an instance of a Workplace object class. Workplace object classes are System Object Model (SOM) object classes which are descendants of the predefined Workplace object class, **WPObject**. For more information about SOM object classes, see *System Object Model Guide and Reference*.

All Workplace objects are descendants of a Workplace object storage class. Workplace storage object classes are responsible primarily for storing and retrieving object class data for instances of objects which are descendants (subclasses) of that storage class.

The three predefined Workplace object storage classes are:

"WPAbstract" on page 9-4 Storage class for objects stored in the user profile (OS2.INI)
"WPFileSystem" on page 9-14 Storage class for objects stored as files in the file system
"WPTransient" on page 9-50 Storage class for non-persistent objects.

Workplace Object Class Hierarchy

The following diagram lists the predefined Workplace object classes in a hierarchical order. Each branch in the tree represents an immediate descendant (subclass) of a Workplace object class. The predefined SOM object class, **SOMObject**, is the root class for all SOM object classes, including all Workplace object classes.

```
CLASS NAME                                          CLASS DEFINITION FILE

SOMObject                                           somobj.sc
   ├── SOMClass                                     somcls.sc
   ├── SOMClassMgr                                  somcm.sc
   └── "WPObject" on page 9-24                      wpobject.sc
       ├── "WPAbstract" on page 9-4                 wpabs.sc
       │   ├── "WPClock" on page 9-5                wpclock.sc
       │   ├── "WPCountry" on page 9-8              wpctry.sc
       │   ├── "WPDisk" on page 9-11                wpdisk.sc
       │   ├── "WPKeyboard" on page 9-22            wpkeybd.sc
       │   ├── "WPMouse" on page 9-23               wpmouse.sc
       │   ├── "WPPalette" on page 9-27             wppalet.sc
       │   │   ├── "WPColorPalette" on page 9-6     wpclrpal.sc
       │   │   ├── "WPFontPalette" on page 9-19     wpfntpal.sc
       │   │   └── "WPSchemePalette" on page 9-40   wpscheme.sc
       │   ├── "WPPrinter" on page 9-31             wpprint.sc
       │   ├── "WPProgram" on page 9-34             wppgm.sc
       │   ├── "WPShadow" on page 9-41              wpshadow.sc
       │   ├── "WPShredder" on page 9-43            wpshred.sc
       │   ├── "WPSound" on page 9-44               wpsound.sc
       │   ├── "WPSpecialNeeds" on page 9-45        wpspneed.sc
       │   ├── "WPSpooler" on page 9-46             wpspool.sc
       │   └── "WPSystem" on page 9-48              wpsystem.sc
       ├── "WPFileSystem" on page 9-14              wpfsys.sc
       │   ├── "WPDataFile" on page 9-9             wpdataf.sc
       │   ├── "WPFolder" on page 9-16              wpfolder.sc
       │   │   ├── "WPDesktop" on page 9-10         wpdesk.sc
       │   │   ├── "WPDrives" on page 9-13          wpdrives.sc
       │   │   ├── "WPStartup" on page 9-47         wpstart.sc
       │   │   └── "WPTemplateFolder" on page 9-49  wptemps.sc
       │   └── "WPProgramFile" on page 9-37         wppgmf.sc
       └── "WPTransient" on page 9-50               wptrans.sc
           ├── "WPJob" on page 9-21                 wpjob.sc
           ├── "WPPort" on page 9-30                wpport.sc
           ├── "WPPrinterDriver" on page 9-33       wppdr.sc
           └── "WPQueueDriver" on page 9-51         wpqdr.sc
```

Figure 9-1. Class Hierarchy for Predefined Workplace Objects

Instances of some Workplace object classes cannot be created as a Workplace object. These classes are provided as base classes which provide support for descendant classes that can have instances created. Other classes are SOM classes and are described in better detail in the *System Object Model Guide and Reference*.

These classes include:

SOMObject	This is the SOM root class, all SOM classes must be descended from **SOMObject**. A Workplace object of this class cannot be created.
SOMClass	This is the SOM metaclass, that is, the instances of this class are class objects. A Workplace object of this class cannot be created.
SOMClassMgr	This is the SOM class manager class. A Workplace object of this class cannot be created.

WPAbstract

Class definition file:wpabs.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract

Description

This is the abstract object storage class. The storage medium for objects that are descendants of the **WPAbstract** class is the INI file. In other words, any object class derived from **WPAbstract** will have persistent storage for its instance variables in the INI file. Note that an abstract object does not have a file name, just a numeric handle that can be used to identify it. An instance of this class can be created as a Workplace object. There are no instances of this class initially created by the system.

Instance methods

There are currently no methods defined for the **WPAbstract** Workplace object class.

WPObject instance methods

The following list shows all the methods overridden by the **WPAbstract** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpCopyObject – WPObject instance method" on page 9-105
- "wpCreateFromTemplate – WPObject instance method" on page 9-106
- "wpSaveImmediate – WPObject instance method" on page 9-198
- "wpSaveState – WPObject instance method" on page 9-200
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpMoveObject – WPObject instance method" on page 9-141
- "wpQueryIconData – WPObject instance method" on page 9-170
- "wpSetIconData – WPObject instance method" on page 9-218
- "wpSetTitle – WPObject instance method" on page 9-227

Class methods

The following shows the class methods overridden by the WPAbstract class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryTitle – WPObject class method" on page 9-268

WPClock

Class definition file:wpclock.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPClock

Description

This is the system clock object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system. It has the title, "System Clock" and resides in the "System Setup" folder.

Instance methods

The following list shows all the WPClock instance methods.

- "wpAddClockAlarmPage – WPClock instance method" on page 9-53
- "wpAddClockDateTimePage – WPClock instance method" on page 9-54
- "wpAddClockView1Page – WPClock instance method" on page 9-55
- "wpAddClockView2Page – WPClock instance method" on page 9-56

WPObject instance methods

The following list shows all the methods overridden by the **WPClock** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpQueryDefaultView – WPObject instance method" on page 9-160
- "wpOpen – WPObject instance method" on page 9-142
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpSaveState – WPObject instance method" on page 9-200
- "wpRestoreState – WPObject instance method" on page 9-194

Class methods

The following list shows all the class methods overridden by the WPClock class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPColorPalette

Class definition file:wpclrpal.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPPalette
        WPColorPalette

Description

This is the color palette object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system. It has the title, "Color Palette" and resides in the "System Setup" folder.

Instance methods

There are currently no methods defined for the **WPColorPalette** Workplace object class.

WPObject instance methods

The following list shows all the methods overridden by the WPColorPalette class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpSetup – WPObject instance method" on page 9-229

These are the keyname – value pairs added by WPColorPalette class.

KEYNAME	VALUE	Description
COLORS	RGB values	These are the initial color values of each cell in the color palette. The values for each cell are separated by commas. This is equivalent to calling the wpSetupCell method. The RGB value must be presented as a 6–digit hex value in the format **0xRRGGBB** where RR,GG, and BB are the red, green and blue, respectively, values ranging between 00 and FF (0–255).

Example

```
pszSetupString="COLORS=0x000000,0xFFFFFF,0xFF23B4
                      0x505050,0x00FFFF,0xA0A0A0"
```

WPPalette methods

- "wpDragCell – WPPalette instance method" on page 9-115
- "wpEditCell – WPPalette instance method" on page 9-121
- "wpPaintCell – WPPalette instance method" on page 9-143
- "wpQueryPaletteHelp – WPPalette instance method" on page 9-173

Class methods

The following list shows all the class methods overridden by the WPColorPalette class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPPalette class methods

- "wpclsQueryEditString – WPPalette class method" on page 9-257

WPCountry

Class definition file:wpctry.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPCountry

Description

This is the country object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system. It has the title, "Country" and resides in the "System Setup" folder.

Instance methods

The following list shows all of the **WPCountry** instance methods.

- "wpAddCountryDatePage – WPCountry instance method" on page 9-57
- "wpAddCountryNumbersPage – WPCountry instance method" on page 9-58
- "wpAddCountryPage – WPCountry instance method" on page 9-59
- "wpAddCountryTimePage – WPCountry instance method" on page 9-60

WPObject instance methods

The following list shows all of the methods overridden by the **WPCountry** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpInitData – WPObject instance method" on page 9-133
- "wpQueryDefaultHelp – WPObject instance method" on page 9-159
- "wpQueryDefaultView – WPObject instance method" on page 9-160

Class methods

The following list shows all the class methods overridden by the WPCountry class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPDataFile

Class definition file:wpdataf.sc

Class hierarchy

SOMObject
  WPObject
    WPFileSystem
      WPDataFile

Description

This is the data file object class. An instance of this class can be created as a Workplace object and is created initially by the system in its template form. It has the title, "Data file" and resides in the "Templates" folder.

Instance methods

The following shows all the **WPDataFile** methods.

- "wpAddFileTypePage – WPDataFile instance method" on page 9-66
- "wpPrintMetaFile – WPDataFile instance method" on page 9-146
- "wpPrintPifFile – WPDataFile instance method" on page 9-148
- "wpPrintPlainTextFile – WPFileSystem instance method" on page 9-149
- "wpPrintPrinterSpecificFile – WPDataFile instance method" on page 9-150
- "wpPrintUnknownFile – WPDataFile instance method" on page 9-151

WPObject instance methods

The following list shows all the methods overridden by the **WPDataFile** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpFormatDragItem – WPObject instance method" on page 9-126
- "wpInitData – WPObject instance method" on page 9-133
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpOpen – WPObject instance method" on page 9-142
- "wpPrintObject – WPObject instance method" on page 9-147
- "wpQueryDefaultHelp – WPObject instance method" on page 9-159
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpSaveState – WPObject instance method" on page 9-200
- "wpUnInitData – WPObject instance method" on page 9-238

Class methods

The following list shows all the class methods overridden by the **WPDataFile** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPDesktop

Class definition file:wpdesk.sc

Class hierarchy

SOMObject
  WPObject
    WPFileSystem
      WPFolder
        WPDesktop

Description

This is the Workplace desktop object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system. It has the title, "OS/2 Desktop" and resides in the root directory of the drive containing the user profile.

Instance methods

The following table shows all the **WPDesktop** methods.

- "wpAddDesktopLockup1Page – WPDesktop instance method" on page 9-61
- "wpAddDesktopLockup2Page – WPDesktop instance method" on page 9-62
- "wpAddDesktopLockup3Page – WPDesktop instance method" on page 9-63
- "wpIsCurrentDesktop – WPDesktop instance method" on page 9-137

WPObject instance methods

The following list shows all the methods overridden by the **WPDesktop** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140

Class methods

The following list shows all the class methods overridden by the **WPDesktop** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPDisk

Class definition file:wpdisk.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPDisk

Description

This is the file system device object class. It is used to represent all types of file system devices including:

- CD ROM drives
- Fixed disk drives
- Floppy diskette drives
- Magnetic tape drives
- Network drives
- Ram drives

An instance of this class can be created as a Workplace object. Shadow instances of this class are created initially by the system for each file system device available. Each instance has a title corresponding to a file system device and resides in the Drives folder. If a primary local removable media file system device such as a floppy drive exists, an instance of this class will also be created on the desktop, which will represent this drive. These instances actually appear as **WPShadow** objects which are shadows of **WPDisk** objects.

Instance methods

The following shows the **WPDisk** methods.

- "wpAddDiskDetailsPage – WPDisk instance method" on page 9-64
- "wpQueryLogicalDrive – WPDisk instance method" on page 9-171
- "wpQueryRootFolder – WPDisk instance method" on page 9-179

WPObject instance methods

The following list shows all the methods overridden by the **WPDisk** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpDragOver – WPObject instance method" on page 9-118
- "wpDrop – WPObject instance method" on page 9-119
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpOpen – WPObject instance method" on page 9-142
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpSaveState – WPObject instance method" on page 9-200
- "wpSetTitle – WPObject instance method" on page 9-227

Class methods

The following list shows all the class methods overridden by the **WPDisk class**. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267

WPDrives

Class definition file:wpdrives.sc

Class hierarchy

SOMObject
  WPObject
    WPFileSystem
      WPFolder
        WPDrives

Description

This is the file system device folder object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system. It has the title, "Drives" and resides in the "OS/2 System" folder.

Methods

There are currently no methods defined for the **WPDrives** Workplace object class.

WPObject methods

The following list shows all the methods overridden by the **WPDrives** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpPopulate – WPFolder instance method" on page 9-144

Class methods

The following list shows all the class methods overridden by the **WPDrives** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPFileSystem

Class definition file:wpfsys.sc

Class hierarchy

SOMObject
  WPObject
    WPFileSystem

Description

This is the file system object storage class. **WPFileSystem** is the storage class that represents all file system objects including directory (folder), data file, executable file, and root directory (drive) objects. This class also provides persistent storage of instance variables for all classes derived from it. A Workplace object of this class cannot be created.

Instance methods

The following list shows all the **WPFileSystem** class methods.

- "wpAddFileMenuPage – WPFileSystem instance method" on page 9-65
- "wpAddFile1Page – WPFileSystem instance method" on page 9-67
- "wpAddFile2Page – WPFileSystem instance method" on page 9-68
- "wpAddFile3Page – WPFileSystem instance method" on page 9-69
- "wpQueryHandle – WPObject instance method" on page 9-168
- "wpQueryRealName – WPFileSystem instance method" on page 9-178
- "wpQueryType – WPFileSystem instance method" on page 9-183
- "wpSetRealName – WPFileSystem instance method" on page 9-224
- "wpSetType – WPFileSystem instance method" on page 9-228

WPObject instance methods

The following list shows all the methods overridden by the **WPFileSystem** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpCopyObject – WPObject instance method" on page 9-105
- "wpCreateFromTemplate – WPObject instance method" on page 9-106
- "wpDoesObjectMatch – WPObject instance method" on page 9-114
- "wpDraggedOverObject – WPObject instance method" on page 9-116
- "wpDroppedOnObject – WPObject instance method" on page 9-120
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpFormatDragItem – WPObject instance method" on page 9-126
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpMoveObject – WPObject instance method" on page 9-141
- "wpOpen – WPObject instance method" on page 9-142
- "wpQueryDetailsData – WPObject instance method" on page 9-161
- "wpQueryIconData – WPObject instance method" on page 9-170
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpSaveImmediate – WPObject instance method" on page 9-198
- "wpSaveState – WPObject instance method" on page 9-200
- "wpSetIconData – WPObject instance method" on page 9-218
- "wpSetTitle – WPObject instance method" on page 9-227
- "wpUnInitData – WPObject instance method" on page 9-238

Class methods

The following list shows the **wpFileSystem** class methods.

- "wpclsQueryInstanceFilter – WPFileSystem class method" on page 9-262
- "wpclsQueryInstanceType – WPFileSystem class method" on page 9-263

WPObject class methods

The following list shows all the class methods overridden by the **WPFileSystem class**. These methods are overridden to modify the behavior defined by an ancestor class.

- "wpclsInitData – WPObject class method" on page 9-246
- "wpclsQueryDetailsInfo – WPObject class method" on page 9-254
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPFolder

Class definition file:wpfolder.sc

Class hierarchy

SOMObject
  WPObject
    WPFileSystem
      WPFolder

Description

This is the folder object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system in its template form. It has the title "Folder" and resides in the "Templates" folder.

Instance methods

The following list shows how all the **WPFolder** methods are related within functional areas.

Settings notebook methods

- "wpAddFolderBackgroundPage – WPFolder instance method" on page 9-70
- "wpAddFolderIncludePage – WPFolder instance method" on page 9-71
- "wpAddFolderSortPage – WPFolder instance method" on page 9-72
- "wpAddFolderView1Page – WPFolder instance method" on page 9-73
- "wpAddFolderView2Page – WPFolder instance method" on page 9-74
- "wpAddFolderView3Page – WPFolder instance method" on page 9-75

Object position methods

- "wpQueryNextIconPos – WPFolder instance method" on page 9-172
- "wpSetNextIconPos – WPFolder instance method" on page 9-219

Folder attributes methods

- "wpQueryFldrAttr – WPFolder instance method" on page 9-164
- "wpQueryFldrDetailsClass – WPFolder instance method" on page 9-165
- "wpQueryFldrFlags – WPFolder instance method" on page 9-166
- "wpQueryFldrFont – WPFolder instance method" on page 9-167
- "wpSetFldrAttr – WPFolder instance method" on page 9-213
- "wpSetFldrDetailsClass – WPFolder instance method" on page 9-214
- "wpSetFldrFlags – WPFolder instance method" on page 9-215
- "wpSetFldrFont – WPFolder instance method" on page 9-216

Folder state methods

- "wpHideFldrRunObjs – WPFolder instance method" on page 9-130

Folder content methods

- "wpDeleteContents – WPFolder instance method" on page 9-110
- "wpPopulate – WPFolder instance method" on page 9-144
- "wpQueryContent – WPFolder instance method" on page 9-158
- "wpRefresh – WPFileSystem instance method" on page 9-185

WPObject instance methods

The following list shows all the methods overridden by the **WPFolder** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddFile2Page – WPFileSystem instance method" on page 9-68
- "wpAddFile3Page – WPFileSystem instance method" on page 9-69
- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpConfirmDelete – WPObject instance method" on page 9-103
- "wpDelete – WPObject instance method" on page 9-108
- "wpDragOver – WPObject instance method" on page 9-118
- "wpDrop – WPObject instance method" on page 9-119
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpFormatDragItem – WPObject instance method" on page 9-126
- "wpFree – WPObject instance method" on page 9-127
- "wpInitData – WPObject instance method" on page 9-133
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpMoveObject – WPObject instance method" on page 9-141
- "wpOpen – WPObject instance method" on page 9-142
- "wpRender – WPObject instance method" on page 9-189
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpSaveState – WPObject instance method" on page 9-200
- "wpSetup – WPObject instance method" on page 9-229
- "wpUnInitData – WPObject instance method" on page 9-238

The following table shows the keyname-value pairs supported by the **WPObject** class.

KEYNAME	VALUE	Description
OPEN	ICON	Open icon view when object is created or when WinSetObjectData is called.
	TREE	Open tree view when object is created or when WinSetObjectData is called.
	DETAILS	Open details view when object is created or when WinSetObjectData is called.
ICONVIEW	s1[,s2,...sn]	Set icon view to the specified styles.
TREEVIEW	s1[,s2,...sn]	Set tree view to the specified styles.
DETAILSVIEW	s1[,s2,...sn]	Set details view to the specified styles.
view styles	FLOWED	flowed list items.
	NONFLOWED	non-flowed list items.
	NONGRID	non-gridded icon view.
	NORMAL	normal size icons.
	MINI	small icons.
	INVISIBLE	no icons.
	LINES	lines in tree view.
	NOLINES	no lines in tree view.
BACKGROUND	filename	This sets the folder background. *filename* is the name of a file in the \OS2\BITMAP directory of the boot drive.

Class methods

The following list shows all the **WPFolder** class methods.

- "wpclsQueryOpenFolders – WPFolder class method" on page 9-265

WPObject class methods

The following list shows all the class methods overridden by the **WPFolder class**. These methods are overridden to modify the behavior defined by an ancestor class.

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPFontPalette

Class definition file:wpfntpal.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPPalette
        WPFontPalette

Description

This is the font palette object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system. It has the title, "Font Palette" and resides in the "System Setup" folder.

Instance methods

There are currently no methods defined for the **WPFontPalette** Workplace object class.

WPObject instance methods

The following list shows all the methods overridden by the **WPFontPalette** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpSetup – WPObject instance method" on page 9-229

The following are the keyname – value pairs added by the **WPFontPalette** class.

KEYNAME	VALUE	Description
FONTS	fonttype	These are the initial fonts for each cell in the font palette. The values for each cell are separated by commas. This is equivalent to calling the wpSetupCell method. The fonttype value is presented as the point size followed by a period which is then followed by the face name.

Example

```
pszSetupString="FONTS=10.Helvetica,8.Helvetica,10.Courier,
                    12.Courier,24.Courier,10.System"
```

WPPalette instance methods

- "wpDragCell – WPPalette instance method" on page 9-115
- "wpEditCell – WPPalette instance method" on page 9-121
- "wpPaintCell – WPPalette instance method" on page 9-143
- "wpQueryPaletteHelp – WPPalette instance method" on page 9-173
- "wpSetupCell – WPPalette instance method" on page 9-233

Class methods

The following list shows all the class methods overridden by the **WPFontPalette** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPPalette class methods

- "wpclsQueryEditString – WPPalette class method" on page 9-257

WPJob

Class definition file:wpjob.sc

Class hierarchy

SOMObject
  WPObject
    WPTransient
      WPJob

Description

This is the job object class. An instance of this class is created by the print object in its icon or detail view.

Instance methods

The following table shows all the **WPJob** methods.

- "wpDeleteJob – WPJob instance method" on page 9-112
- "wpHoldJob – WPJob instance method" on page 9-131
- "wpPrintJobNext – WPJob instance method" on page 9-145
- "wpReleaseJob – WPJob instance method" on page 9-187
- "wpStartJobAgain – WPJob instance method" on page 9-235

WPObject instance methods

The following list shows all the methods overridden by the **WPJob** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpCopyObject – WPObject instance method" on page 9-105
- "wpDragOver – WPObject instance method" on page 9-118
- "wpDrop – WPObject instance method" on page 9-119
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpFree – WPObject instance method" on page 9-127
- "wpInitData – WPObject instance method" on page 9-133
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpOpen – WPObject instance method" on page 9-142
- "wpSetTitle – WPObject instance method" on page 9-227
- "wpUnInitData – WPObject instance method" on page 9-238

Class methods

The following list shows all the class methods overridden by the **WPJob** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryDetails – WPObject class method" on page 9-253
- "wpclsQueryDetailsInfo – WPObject class method" on page 9-254
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPKeyboard

Class definition file:wpkeybd.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPKeyboard

Description

This is the keyboard object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system. It has the title, "Keyboard" and resides in the "System Setup" folder.

Instance methods

The following list shows all the **WPKeyboard** methods.

- "wpAddKeyboardMappingsPage – WPKeyboard instance method" on page 9-76
- "wpAddKeyboardSpecialNeedsPage – WPKeyboard instance method" on page 9-77
- "wpAddKeyboardTimingPage – WPKeyboard instance method" on page 9-78

WPObject instance methods

The following list shows all the methods overridden by the **WPKeyboard** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpFilterPopupMenu – WPObject instance method" on page 9-123

Class methods

The following list shows all the class methods overridden by the WPKeyboard class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPMouse

Class definition file:wpmouse.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPMouse

Description

This is the mouse object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system. It has the title, "Mouse" and resides in the "System Setup" folder.

Instance methods

The following table shows all the **WPMouse** methods.

- "wpAddMouseMappingsPage – WPMouse instance method" on page 9-79
- "wpAddMouseTimingPage – WPMouse instance method" on page 9-80
- "wpAddMouseTypePage – WPMouse instance method" on page 9-81

WPObject instance methods

The following list shows all the methods overridden by the **WPMouse** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpFilterPopupMenu – WPObject instance method" on page 9-123

Class methods

The following list shows all the class methods overridden by the **WPMouse** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPObject

Class definition file:wpobject.sc

Class hierarchy

SOMObject
  WPObject

Description

This is the root Workplace object class. This is the fundamental class from which all workplace objects are derived, irrespective of where they are actually stored. Immediate descendant classes of **WPObject** are called *storage classes*, since they take responsibility for storing the object information, typically in a persistent form. Predefined workplace object storage classes are:

- "WPAbstract" on page 9-4
- "WPFileSystem" on page 9-14
- "WPTransient" on page 9-50

A Workplace object of this class cannot be created.

Instance methods

The following lists show how all the **WPObject** instance methods are related within functional areas.

Settings notebook methods

- "wpAddObjectGeneralPage – WPObject instance method" on page 9-82
- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpInsertSettingsPage – WPObject instance method" on page 9-136

Save/restore state methods

- "wpRestoreData – WPObject instance method" on page 9-192
- "wpRestoreLong – WPObject instance method" on page 9-193
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpRestoreString – WPObject instance method" on page 9-195
- "wpSaveData – WPObject instance method" on page 9-196
- "wpSaveImmediate – WPObject instance method" on page 9-198
- "wpSaveLong – WPObject instance method" on page 9-199
- "wpSaveState – WPObject instance method" on page 9-200
- "wpSaveString – WPObject instance method" on page 9-201

Object usage methods

- "wpAddToObjUseList – WPObject instance method" on page 9-95
- "wpDeleteFromObjUseList – WPObject instance method" on page 9-111
- "wpFindUseItem – WPObject instance method" on page 9-125
- "wpUnlockObject – WPObject instance method" on page 9-237

Popup menu methods

- "wpClose – WPObject instance method" on page 9-98
- "wpCopyObject – WPObject instance method" on page 9-105
- "wpCreateFromTemplate – WPObject instance method" on page 9-106
- "wpCreateShadowObject – WPObject instance method" on page 9-107
- "wpDelete – WPObject instance method" on page 9-108
- "wpDisplayHelp – WPObject instance method" on page 9-113
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpHide – WPObject instance method" on page 9-129
- "wpInsertPopupMenuItems – WPObject instance method" on page 9-134
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpMoveObject – WPObject instance method" on page 9-141
- "wpOpen – WPObject instance method" on page 9-142
- "wpPrintObject – WPObject instance method" on page 9-147
- "wpRestore – WPObject instance method" on page 9-191

Query and Set object information methods

- "wpQueryConfirmations – WPObject instance method" on page 9-157
- "wpQueryDefaultHelp – WPObject instance method" on page 9-159
- "wpQueryDefaultView – WPObject instance method" on page 9-160
- "wpQueryDetailsData – WPObject instance method" on page 9-161
- "wpQueryIcon – WPObject instance method" on page 9-169
- "wpQueryIconData – WPObject instance method" on page 9-170
- "wpQueryStyle – WPObject instance method" on page 9-181
- "wpQueryTitle – WPObject instance method" on page 9-182
- "wpSetDefaultHelp – WPObject instance method" on page 9-209
- "wpSetDefaultView – WPObject instance method" on page 9-211
- "wpSetIcon – WPObject instance method" on page 9-217
- "wpSetIconData – WPObject instance method" on page 9-218
- "wpSetStyle – WPObject instance method" on page 9-226
- "wpSetTitle – WPObject instance method" on page 9-227

Error handling methods

- "wpQueryError – WPObject instance method" on page 9-163
- "wpSetError – WPObject instance method" on page 9-212

Memory management methods

- "wpAllocMem – WPObject instance method" on page 9-97
- "wpFreeMem – WPObject instance method" on page 9-128

Setup/cleanup methods

- "wpFree – WPObject instance method" on page 9-127
- "wpInitData – WPObject instance method" on page 9-133
- "wpScanSetupString – WPObject instance method" on page 9-202
- "wpSetup – WPObject instance method" on page 9-229
- "wpUnInitData – WPObject instance method" on page 9-238

Direct manipulation methods

- "wpDragOver – WPObject instance method" on page 9-118
- "wpDraggedOverObject – WPObject instance method" on page 9-116
- "wpDrop – WPObject instance method" on page 9-119
- "wpDroppedOnObject – WPObject instance method" on page 9-120
- "wpEndConversation – WPObject instance method" on page 9-122
- "wpFormatDragItem – WPObject instance method" on page 9-126
- "wpRender – WPObject instance method" on page 9-189
- "wpRenderComplete – WPObject instance method" on page 9-190

Miscellaneous methods

- "wpCnrInsertObject – WPObject instance method" on page 9-99
- "wpCnrRemoveObject – WPObject instance method" on page 9-101
- "wpCnrSetEmphasis – WPObject instance method" on page 9-102
- "wpConfirmDelete – WPObject instance method" on page 9-103
- "wpCopiedFromTemplate – WPObject instance method" on page 9-104
- "wpDoesObjectMatch – WPObject instance method" on page 9-114
- "wpRegisterView – WPObject instance method" on page 9-186
- "wpSwitchTo – WPObject instance method" on page 9-236

SOMObject methods

The following lists show all the instance methods overridden by the **WPObject** class. These instance methods are overridden in order to modify the behavior defined by an ancestor class.

- somFree
- somInit
- somUninit

Class Methods

The following list shows all the WPObject class methods.

- "wpclsCreateDefaultTemplates – WPObject class method" on page 9-240
- "wpclsFindObjectEnd – WPObject class method" on page 9-241
- "wpclsFindObjectFirst – WPObject class method" on page 9-242
- "wpclsFindObjectNext – WPObject class method" on page 9-244
- "wpclsInitData – WPObject class method" on page 9-246
- "wpclsMakeAwake – WPObject class method" on page 9-247
- "wpclsNew – WPObject class method" on page 9-249
- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryDetails – WPObject class method" on page 9-253
- "wpclsQueryDetailsInfo – WPObject class method" on page 9-254
- "wpclsQueryError – WPObject class method" on page 9-258
- "wpclsQueryFolder – WPObject class method" on page 9-259
- "wpclsQueryIcon – WPObject class method" on page 9-260
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryObject – WPObject class method" on page 9-264
- "wpclsSetError – WPObject class method" on page 9-269
- "wpclsQuerySettingsPageSize – WPObject class method" on page 9-266
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268
- "wpclsUnInitData – WPObject class method" on page 9-270

SOMObject methods

The following lists show all the class methods overridden by the **WPObject** class. These methods are overridden in order to modify the behavior defined by ancestor class.

- somUnInit

SOMClass methods

- somInitClass
- somNew
- somRenew

WPPalette

Class definition file:wppalet.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPPalette

Description

This is the palette object class. There are no instances of this class initially created by the system.

Instance methods

The following list shows all the **WPPalette** instance methods.

- "wpDragCell – WPPalette instance method" on page 9-115
- "wpEditCell – WPPalette instance method" on page 9-121
- "wpPaintCell – WPPalette instance method" on page 9-143
- "wpQueryPaletteHelp – WPPalette instance method" on page 9-173
- "wpQueryPaletteInfo – WPPalette instance method" on page 9-174
- "wpRedrawCell – WPPalette instance method" on page 9-184
- "wpSetupCell – WPPalette instance method" on page 9-233
- "wpSetPaletteInfo – WPPalette instance method" on page 9-220

WPObject instance methods

The following list shows all the methods overridden by the **WPPalette** class. These methods are overridden to modify the behavior defined by an ancestor class.

- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpInitData – WPObject instance method" on page 9-133
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpOpen – WPObject instance method" on page 9-142
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpSaveState – WPObject instance method" on page 9-200
- "wpSetup – WPObject instance method" on page 9-229
- "wpUnInitData – WPObject instance method" on page 9-238

wpSetup override by WPPalette

The following are the keyname – value pairs added by the **WPFontPalette** class.

KEYNAME	VALUE	Description
XCELLCOUNT	columns	Number of columns of cells.
YCELLCOUNT	rows	Number of rows of cells.
XCELLWIDTH	width	Width in dialog units of each cell.
XCELLHEIGHT	height	Height in dialog units of each cell.
XCELLGAP	gap	X separation in dialog units between each cell.
YCELLGAP	gap	Y separation in dialog units between each cell.

Example

```
pszSetupString="XCELLCOUNT=3,YCELLCOUNT=4"
```

Class methods

The following list shows all the class methods overridden by the **WPPalette** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPPalette class methods

- "wpclsQueryEditString – WPPalette class method" on page 9-257

WPPort

Class definition file:wpport.sc

Class hierarchy

SOMObject
  WPObject
    WPTransient
      WPPort

Description

This is the port object class. An instance of this class is created by the print object in its settings view.

Instance methods

The following list shows all the methods overridden by the **WPPort** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

WPObject instance methods

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpDragOver – WPObject instance method" on page 9-118
- "wpDrop – WPObject instance method" on page 9-119
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpInitData – WPObject instance method" on page 9-133
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpOpen – WPObject instance method" on page 9-142
- "wpSetTitle – WPObject instance method" on page 9-227
- "wpUnInitData – WPObject instance method" on page 9-238

Class methods

The following list shows all the class methods overridden by the **WPPort** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPPrinter

Class definition file:wpprint.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPPrinter

Description

This is the print object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system in its template form. It has the title, "Create print destination" and resides in the "Templates" folder. Instances of this class are also created initially by the system for each print configured. Each instance will have a title corresponding to the description of the configured queue and printer and will reside on the desktop.

Instance methods

The following list shows all the **WPPrinter** methods.

- "wpDeleteAllJobs – WPPrinter instance method" on page 9-109
- "wpHoldPrinter – WPPrinter instance method" on page 9-132
- "wpQueryComputerName – WPPrinter instance method" on page 9-156
- "wpQueryPrinterName – WPPrinter instance method" on page 9-175
- "wpReleasePrinter – WPPrinter instance method" on page 9-188
- "wpSetComputerName – WPPrinter instance method" on page 9-208
- "wpSetDefaultPrinter – WPPrinter instance method" on page 9-210
- "wpSetPrinterName – WPPrinter instance method" on page 9-221

WPObject instance methods

The following list shows all the methods overridden by the **WPPrinter** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpCopiedFromTemplate – WPObject instance method" on page 9-104
- "wpCopyObject – WPObject instance method" on page 9-105
- "wpDragOver – WPObject instance method" on page 9-118
- "wpDrop – WPObject instance method" on page 9-119
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpFree – WPObject instance method" on page 9-127
- "wpInitData – WPObject instance method" on page 9-133
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpOpen – WPObject instance method" on page 9-142
- "wpQueryDefaultHelp – WPObject instance method" on page 9-159
- "wpQueryDefaultView – WPObject instance method" on page 9-160
- "wpQueryIcon – WPObject instance method" on page 9-169
- "wpQueryStyle – WPObject instance method" on page 9-181
- "wpQueryTitle – WPObject instance method" on page 9-182
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpSaveState – WPObject instance method" on page 9-200
- "wpSetTitle – WPObject instance method" on page 9-227
- "wpUnInitData – WPObject instance method" on page 9-238

Class methods

The following list shows all the class methods overridden by the **WPPrinter** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPPrinterDriver

Class definition file:wppdr.sc

Class hierarchy

SOMObject
  WPObject
    WPTransient
      WPPrinterDriver

Description

This is the printer driver object class. An instance of this class is created by the print object in its settings view.

Instance methods

The following list shows all the methods overridden by the **WPPrinterDriver** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

WPObject instance methods

- "wpDragOver – WPObject instance method" on page 9-118
- "wpDrop – WPObject instance method" on page 9-119
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpInitData – WPObject instance method" on page 9-133
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpOpen – WPObject instance method" on page 9-142
- "wpSetTitle – WPObject instance method" on page 9-227
- "wpUnInitData – WPObject instance method" on page 9-238

Class methods

The following list shows all the class methods overridden by the **WPPrinterDrive** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPProgram

Class definition file:wppgm.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPProgram

Description

This is the program object class. This class provides an object that points at executable programs, and allows the user to run that program by simply double-clicking on the program object. The program can also contain a variety of useful additional parameters, such as the environment for the program and the parameters that are passed to it. An instance of this class can be created as a Workplace object and is created initially by the system in its template form. It has the title "Program" and resides in the "Templates" folder.

Other instances of this class initially created by the system include:

- "DOS Full Screen" in the "Command Prompts" folder
- "DOS Window" in the "Command Prompts" folder
- "OS/2 Full Screen" in the "Command Prompts" folder
- "OS/2 Window" in the "Command Prompts" folder
- Every object in the "Games" folder
- Some objects in the "Information" folder
- Every object in the "Productivity" folder

Instance methods

The following list shows all the **WPProgram** methods.

- "wpAddProgramAssociationPage – WPProgram instance method" on page 9-84
- "wpAddProgramPage – WPProgram instance method" on page 9-85
- "wpAddProgramSessionPage – WPProgram instance method" on page 9-87
- "wpQueryAssociationFilter – WPProgram instance method" on page 9-152
- "wpQueryAssociationType – WPProgram instance method" on page 9-154
- "wpQueryProgDetails – WPProgram instance method" on page 9-176
- "wpSetAssociationFilter – WPProgram instance method" on page 9-204
- "wpSetAssociationType – WPProgram instance method" on page 9-206
- "wpSetProgDetails – WPProgram instance method" on page 9-222

WPObject instance methods

The following list shows all the methods overridden by the **WPProgram** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpCopiedFromTemplate – WPObject instance method" on page 9-104
- "wpCopyObject – WPObject instance method" on page 9-105
- "wpDragOver – WPObject instance method" on page 9-118
- "wpDrop – WPObject instance method" on page 9-119
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpMoveObject – WPObject instance method" on page 9-141
- "wpOpen – WPObject instance method" on page 9-142
- "wpQueryDefaultHelp – WPObject instance method" on page 9-159
- "wpQueryIcon – WPObject instance method" on page 9-169
- "wpQueryIconData – WPObject instance method" on page 9-170
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpSetTitle – WPObject instance method" on page 9-227

- "wpSetup – WPObject instance method" on page 9-229
- "wpSaveState – WPObject instance method" on page 9-200
- "wpUnInitData – WPObject instance method" on page 9-238

wpSetup override by WPProgram

The following table shows the keyname – value pairs added by the **WPProgram** class.

KEYNAME	VALUE	Description
ASSOCFILTER	filters	Sets the filename filter for files associated to this program. Multiple filters are separated by commas.
ASSOCTYPE	type	Sets the type of files associated to this program. Multiple filters are separated by commas.
EXENAME	filename	Sets the name of the program.
PARAMETERS	params	Sets the parameters list, which may include substitution characters.
PROGTYPE	FULLSCREEN	Sets the session type to OS/2 full screen
	PM	Sets the session type to PM.
	SEPARATEWIN	Sets the session type to WIN-OS2 window running in a separate VDM.
	VDM	Sets the session type to DOS full screen.
	WIN	Sets the session type to WIN-OS2 full screen.
	WINDOWABLEVIO	Sets the session type to OS/2 windowed.
	WINDOWEDVDM	Sets the session type to DOS windowed.
	WINDOWEDWIN	Sets the session type to WIN-OS2 window.
NOAUTOCLOSE	YES	Leaves the window open upon program termination.
	NO	Closes the window when the program terminates.
STARTUPDIR	pathname	Sets the working directory.
VIEWBUTTON	not supported	This keyname is not supported by this class.
WINDOWPOS	not supported	This keyname is not supported by this class.

```
pszSetupString="PROGTYPE=VDM;EXENAME=C=\EDIT.EXE;
                ASSOCFILTER=*.DOC,*.TXT;ASSOCTYPE=Plain Text"
```

Class methods

The following list shows all the class methods overridden by the **WPProgram** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryTitle – WPObject class method" on page 9-268
- "wpclsQueryStyle – WPObject class method" on page 9-267

WPProgramFile

Class definition file:wppgmf.sc

Class hierarchy

SOMObject
  WPObject
    WPFileSystem
      WPProgramFile

Description

This is the program file object class. All executable files are of this class. From the settings notebook pages for objects of this class, it is possible to set up associations to various data file types (files with .TYPE EAs or file extensions). It is also possible to set up a default working directory and specify parameters to the executable to be used when it is opened from the shell. An instance of this class can be created as a Workplace object. Instances of this class are created initially by the system for each program file in the system. Each instance has a title corresponding to the file name of the program file it represents and resides in a folder corresponding to the physical directory in which the program file resides.

Instance methods

The following list shows all the **WPProgramFile** methods.

- "wpAddProgramAssociationPage – WPProgramFile instance method" on page 9-83
- "wpAddProgramPage – WPProgramFile instance method" on page 9-86
- "wpAddProgramSessionPage – WPProgramFile instance method" on page 9-88
- "wpQueryAssociationFilter – WPProgramFile instance method" on page 9-153
- "wpQueryAssociationType – WPProgramFile instance method" on page 9-155
- "wpQueryProgDetails – WPProgramFile instance method" on page 9-177
- "wpSetAssociationFilter – WPProgramFile instance method" on page 9-205
- "wpSetAssociationType – WPProgramFile instance method" on page 9-207
- "wpSetProgDetails – WPProgramFile instance method" on page 9-223

WPObject instance methods

The following list shows all the methods overridden by the **WPProgramFile** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpCopyObject – WPObject instance method" on page 9-105
- "wpDragOver – WPObject instance method" on page 9-118
- "wpDrop – WPObject instance method" on page 9-119
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpMoveObject – WPObject instance method" on page 9-141
- "wpOpen – WPObject instance method" on page 9-142
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpSaveState – WPObject instance method" on page 9-200

wpSetup override by **WPProgramFile**

The following table shows the keyname-value pairs added by the **WPProgramFile** class.

KEYNAME	VALUE	Description
ASSOCFILTER	filters	Sets the filename filter for files associated to this program. Multiple filters are separated by commas.
ASSOCTYPE	type	Sets the type of files associated to this program. Multiple filters are separated by commas.
EXENAME	filename	Sets the name of the program.
PARAMETERS	params	Sets the parameters list, which may include substitution characters.
PROGTYPE	FULLSCREEN	Sets the session type to OS/2 full screen
	PM	Sets the session type to PM.
	SEPARATEWIN	Sets the session type to WIN-OS2 window running in a separate VDM.
	VDM	Sets the session type to DOS full screen.
	WIN	Sets the session type to WIN-OS2 full screen.
	WINDOWABLEVIO	Sets the session type to OS/2 windowed.
	WINDOWEDVDM	Sets the session type to DOS windowed.
	WINDOWEDWIN	Sets the session type to WIN-OS2 window.
NOAUTOCLOSE	YES	Leaves the window open upon program termination.
	NO	Closes the window when the program terminates.
STARTUPDIR	pathname	Sets the working directory.
VIEWBUTTON	not supported	This keyname is not supported by this class.
WINDOWPOS	not supported	This keyname is not supported by this class.

```
pszSetupString="PROGTYPE=VDM;EXENAME=C=\EDIT.EXE;
                ASSOCFILTER=*.DOC,*.TXT;ASSOCTYPE=Plain Text"
```

Class methods

The following list shows all the class methods overridden by the **WPProgramFile** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPSchemePalette

Class definition file:wpscheme.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPPalette
        WPSchemePalette

Description

This is the scheme palette object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system. It has the title, "Window Schemes" and resides in the "System Setup" folder. There are currently no methods defined for the WPSchemePalette workplace object class.

Instance methods

The following list shows all the methods overridden by the **WPSchemePalette** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

WPObject instance methods

- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpSetup – WPObject instance method" on page 9-229

WPPalette methods

- "wpDragCell – WPPalette instance method" on page 9-115
- "wpEditCell – WPPalette instance method" on page 9-121
- "wpPaintCell – WPPalette instance method" on page 9-143
- "wpQueryPaletteHelp – WPPalette instance method" on page 9-173
- "wpSetupCell – WPPalette instance method" on page 9-233

Class methods

The following list shows all the class methods overridden by the **WPSchemePalette** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPPalette class methods

- "wpclsQueryEditString – WPPalette class method" on page 9-257
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267

WPShadow

Class definition file:wpshadow.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPShadow

Description

This is the shadow object class. This class provides a persistent link or reference to any other object class. This is achieved by storing away the location and identity of the object that it is linked to and then rerouting all requests for help, context menus, and open views on to the object that it is linked to. Delete, Copy, and Move are the only action requests that are handled by the **WPShadow** object and are not rerouted to the linked object. An instance of this class can be created as a Workplace object. There are no instances of this class initially created by the system.

Instance methods

The following list shows all the **WPShadow** methods.

- "wpQueryShadowedObject – WPShadow instance method" on page 9-180
- "wpSetShadowTitle – WPShadow instance method" on page 9-225

WPObject instance methods

The following list shows all the methods overridden by the **WPShadow** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpConfirmDelete – WPObject instance method" on page 9-103
- "wpDragOver – WPObject instance method" on page 9-118
- "wpDraggedOverObject – WPObject instance method" on page 9-116
- "wpDrop – WPObject instance method" on page 9-119
- "wpDroppedOnObject – WPObject instance method" on page 9-120
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpFormatDragItem – WPObject instance method" on page 9-126
- "wpInitData – WPObject instance method" on page 9-133
- "wpInsertPopupMenuItems – WPObject instance method" on page 9-134
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpOpen – WPObject instance method" on page 9-142
- "wpPrintObject – WPObject instance method" on page 9-147
- "wpQueryDefaultHelp – WPObject instance method" on page 9-159
- "wpQueryStyle – WPObject instance method" on page 9-181
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpSaveState – WPObject instance method" on page 9-200
- "wpSetTitle – WPObject instance method" on page 9-227
- "wpUnInitData – WPObject instance method" on page 9-238

These are the keyname-value pairs supported by the **WPObject** class.

KEYNAME	VALUE	Description
SHADOWID	<name> or filename	This specifies the object for which this object is a shadow of. The value for this keyname is an object's id (OBJECTID) or a fully qualified pathname of a directory, program file, or data file.

Class methods

The following list shows all the class methods overridden by the **WPShadow** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPShredder

Class definition file:wpshred.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPShredder

Description

This is the shredder device object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system. It has the title "shredder" and resides on the desktop. There are currently no methods defined for the **WPShredder** Workplace object class.

Instance methods

The following list shows all the methods overridden by the **WPShredder** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

WPObject instance methods

- "wpDragOver – WPObject instance method" on page 9-118
- "wpDrop – WPObject instance method" on page 9-119
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpOpen – WPObject instance method" on page 9-142

Class methods

The following list shows all the class methods overridden by the **WPShredder** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPSound

Class definition file:wpsound.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPSound

Description

This is the sound object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system. It has the title "Sound" and resides in the "System Setup" folder.

Instance methods

The following list shows all the **WPSound** methods.

- "wpAddSoundWarningBeepPage – WPSound instance method" on page 9-90

WPObject instance methods

The following list shows all the methods overridden by the **WPSound** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpFilterPopupMenu – WPObject instance method" on page 9-123

Class methods

The following list shows all the class methods overridden by the **WPSound** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPSpecialNeeds

Class definition file:wpspneed.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPSpecialNeeds

Description

This is the special needs object class. An instance of this class can be created as a Workplace object. An instance of this class is created initially by the system when the keyboard special needs mode is activated via the keyboard object settings. It has the title, "Special Needs" and resides on the desktop.

Instance methods

There are currently no methods defined for the **WPSpecialNeeds** Workplace object class.

WPObject instance methods

The following list shows all the methods overridden by the **WPSpecialNeeds** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpFilterPopupMenu – WPObject instance method" on page 9-123

Class methods

The following list shows all the class methods overridden by the **WPSpecialNeeds** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPSpooler

Class definition file:wpspool.sc

Class hierarchy

SOMObject
WPObject
WPAbstract
WPSpooler

Description

This is the spooler object class. An instance of this class is created initially by the system. It has the title, "Spooler" and resides in the "System Configuration" folder.

Instance methods

The following list shows all the methods overridden by the **WPSpooler** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

WPObject instance methods

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpDragOver – WPObject instance method" on page 9-118
- "wpDrop – WPObject instance method" on page 9-119
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpInitData – WPObject instance method" on page 9-133
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpOpen – WPObject instance method" on page 9-142
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpSaveState – WPObject instance method" on page 9-200
- "wpSetTitle – WPObject instance method" on page 9-227
- "wpUnInitData – WPObject instance method" on page 9-238

Class methods

The following list shows all the class methods overridden by the WPSpooler class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPStartup

Class definition file:wpstart.sc

Class hierarchy

SOMObject
  WPObject
    WPFileSystem
      WPFolder
        WPStartup

Description

This is the startup folder object class. Any object in the startup folder will be automatically opened every time the system is restarted. Any object in the system that is open at shutdown time will be automatically reopened when the system is restarted. The startup folder is used to automatically open (start) objects that are not necessarily open (running) when the system is shutdown. This usually includes things such as a batch file that initializes the network and then terminates. An instance of this class can be created as a Workplace object and is created initially by the system. It has the title "Startup" and resides in the "OS/2 System" folder.

Instance methods

There are currently no methods defined for the **WPStartup** Workplace object class.

WPObject instance methods

The following list shows all the methods overridden by the **WPStartup** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpFree – WPObject instance method" on page 9-127
- "wpSetup – WPObject instance method" on page 9-229

Class methods

The following list shows all the class methods overridden by the **WPStartup class**. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPSystem

Class definition file:wpsystem.sc

Class hierarchy

SOMObject
  WPObject
    WPAbstract
      WPSystem

Description

This is the system object class. An instance of this class can be created as a Workplace object and is created initially by the system. It has the title, "System" and resides in the "System Setup" folder.

Instance methods

The following list shows all the **WPSystem** class methods.

- "wpAddSystemConfirmationPage – WPSystem instance method" on page 9-91
- "wpAddSystemLogoPage – WPSystem instance method" on page 9-92
- "wpAddSystemPrintScreenPage – WPSystem instance method" on page 9-93
- "wpAddSystemWindowPage – WPSystem instance method" on page 9-94

WPObject instance methods

The following list shows all the methods overridden by the **WPSystem** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpAddSettingsPages – WPObject instance method" on page 9-89
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpRestoreState – WPObject instance method" on page 9-194
- "wpSaveState – WPObject instance method" on page 9-200

Class methods

The following list shows all the class methods overridden by the **WPSystem** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPTemplateFolder

Class definition file:wptplfdr.sc

Class hierarchy

SOMObject
  WPObject
    WPFileSystem
      WPFolder
        WPTemplateFolder

Description

This is the template folder object class. This class of folder is guaranteed to always contain a template instance of every class of object that the user can create that supports the "Create another" action.

A **WPTemplateFolder** object is a normal folder in all respects other than:

- It always contains a template for every class that supports them.
- The last template for each object class can not be deleted from the folder.

An instance of this class can be created as a Workplace object and is created initially by the system. It has the title, "Templates" and resides on the desktop. A template instance is also created for each data type defined in an application's **ASSOCTABLE** resource.

Instance methods

There are currently no methods defined for the **WPTemplateFolder** Workplace object class.

WPObject instance methods

The following list shows all the methods overridden by the **WPTemplateFolder** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpPopulate – WPFolder instance method" on page 9-144

Class methods

The following shows the class methods overridden by the **WPTemplateFolder** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPTransient

Class definition file:wptrans.sc

Class hierarchy

SOMObject
  WPObject
    WPTransient

Description

This is the non-persistent object storage class. The **WPTransient** class is a storage class with no storage medium. That means that instances of object classes derived from **WPTransient** do not persist across reboots. This class is available for applications which need to utilize a large amount of workplace functionality (such as context menus and settings notebooks) in their object class without having to be a file, directory or a record in the INI file. An instance of this class can be created as a Workplace object. There are no instances of this class initially created by the system.

Instance methods

There are currently no methods defined for the **WPTransient** Workplace object class.

WPObject instance methods

The following list shows all the methods overridden by the **WPTransient** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

- "wpCopyObject – WPObject instance method" on page 9-105

Class methods

The following list shows all the class methods overridden by the **WPTransient** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

WPQueueDriver

Class definition file:wpqdr.sc

Class hierarchy

SOMObject
  WPObject
    WPTransient
      WPQueueDriver

Description

This is the queue driver object class. An instance of this class is created by the print object in its settings view.

Instance methods

The following list shows all the methods overridden by the **WPQueueDriver** class. These methods are overridden in order to modify the behavior defined by an ancestor class.

WPObject instance methods

- "wpDragOver – WPObject instance method" on page 9-118
- "wpDrop – WPObject instance method" on page 9-119
- "wpFilterPopupMenu – WPObject instance method" on page 9-123
- "wpInitData – WPObject instance method" on page 9-133
- "wpMenuItemHelpSelected – WPObject instance method" on page 9-138
- "wpMenuItemSelected – WPObject instance method" on page 9-139
- "wpModifyPopupMenu – WPObject instance method" on page 9-140
- "wpOpen – WPObject instance method" on page 9-142
- "wpSetTitle – WPObject instance method" on page 9-227
- "wpUnInitData – WPObject instance method" on page 9-238

Class methods

The following list shows all the class methods overridden by the **WPQueueDriver** class. These methods are overridden to modify the behavior defined by an ancestor class.

WPObject class methods

- "wpclsQueryDefaultHelp – WPObject class method" on page 9-251
- "wpclsQueryDefaultView – WPObject class method" on page 9-252
- "wpclsQueryIconData – WPObject class method" on page 9-261
- "wpclsQueryStyle – WPObject class method" on page 9-267
- "wpclsQueryTitle – WPObject class method" on page 9-268

Workplace Instance Methods

The following pages contain an alphabetical listing of the Workplace Instance methods.

wpAddClockAlarmPage – WPClock instance method

#define INCL_WINWORKPLACE

ULONG wpAddClockAlarmPage (WPClock * self, HWND hwndNotebook)

The wpAddClockAlarmPage instance method is called to allow the object to add the *Alarm* page to its settings notebook.

Parameters

self (WPClock *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Alarm* page from the object's settings notebook.

To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddClockDateTimePage – WPClock instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpAddClockDateTimePage (WPClock * self, HWND hwndNotebook)

The wpAddClockDateTimePage instance method is called to allow the object to add the *Date/Time* page to its settings notebook.

Parameters

self (WPClock *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Date/Time* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddClockView1Page — WPClock instance method

#define INCL_WINWORKPLACE

ULONG wpAddClockView1Page (WPClock * self, HWND hwndNotebook)

The wpAddClockView1Page instance method is called to allow the object to add the *View 1* page to its settings notebook.

Parameters

self (WPClock *) — input

The pointer to this object.

hwndNotebook (HWND) — input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *View 1* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddClockView2Page — WPClock instance method

#define INCL_WINWORKPLACE

```
ULONG wpAddClockView2Page (WPClock * self, HWND hwndNotebook)
```

The wpAddClockView2Page instance method is called to allow the object to add the *View 2* page to its settings notebook.

Parameters

self (WPClock *) — input

The pointer to this object.

hwndNotebook (HWND) — input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *View 2* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddCountryDatePage — WPCountry instance method

#define INCL_WINWORKPLACE

ULONG wpAddCountryDatePage (WPCountry * self, HWND hwndNotebook)

The wpAddCountryDatePage instance method is called to allow the object to add the *Date* page to its settings notebook.

Parameters

self (WPCountry *) — input

The pointer to this object.

hwndNotebook (HWND) — input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Date* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddCountryNumbersPage – WPCountry instance method

#define INCL_WINWORKPLACE

```
ULONG wpAddCountryNumbersPage (WPCountry * self, HWND hwndNotebook)
```

The wpAddCountryNumbersPage instance method is called to allow the object to add the *Numbers* page to its settings notebook.

Parameters

self (WPCountry *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Numbers* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddCountryPage – WPCountry instance method

#define INCL_WINWORKPLACE

ULONG wpAddCountryPage (WPCountry * self, HWND hwndNotebook)

The wpAddCountryPage instance method is called to allow the object to add the *Country* page to its settings notebook.

Parameters

self (WPCountry *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Country* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddCountryTimePage – WPCountry instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpAddCountryTimePage (WPCountry * self, HWND hwndNotebook)

The wpAddCountryTimePage instance method is called to allow the object to add the *Time* page to its settings notebook.

Parameters

self (WPCountry *) – input

The pointer to this object.

hwndNotebook (HWND) – input

Settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Time* page from the object's settings notebook.

To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddDesktopLockup1Page — WPDesktop instance method

#define INCL_WINWORKPLACE

```
ULONG wpAddDesktopLockup1Page (WPDesktop * self, HWND hwndNotebook)
```

The wpAddDesktopLockup1Page instance method is called to allow the object to add the *Lockup 1* page to its settings notebook.

Parameters

self (WPDesktop *) — input

The pointer to this object.

hwndNotebook (HWND) — input

Settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Lockup 1* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddDesktopLockup2Page – WPDesktop instance method

#define INCL_WINWORKPLACE

ULONG wpAddDesktopLockup2Page (WPDesktop * self, HWND hwndNotebook)

The wpAddDesktopLockup2Page instance method is called to allow the object to add the *Lockup 2* page to its settings notebook.

Parameters

self (WPDesktop *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Lockup 2* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddDesktopLockup3Page — WPDesktop instance method

#define INCL_WINWORKPLACE

ULONG wpAddDesktopLockup3Page (WPDesktop * self, HWND hwndNotebook)

The wpAddDesktopLockup3Page instance method is called to allow the object to add the *Lockup 3* page to its settings notebook.

Parameters

self (WPDesktop *) — input

The pointer to this object.

hwndNotebook (HWND) — input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Lockup 3* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddDiskDetailsPage – WPDisk instance method

```
#define INCL_WINWORKPLACE
```

```
ULONG wpAddDiskDetailsPage (WPDisk * self, HWND hwndNotebook)
```

The wpAddDiskDetailsPage instance method is called to allow the object to add the *Details* page to its settings notebook.

Parameters

self (WPDisk *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Details* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddFileMenuPage – WPFileSystem instance method

#define INCL_WINWORKPLACE

ULONG wpAddFileMenuPage (WPFileSystem * self, HWND hwndNotebook)

The wpAddFileMenuPage instance method is called to allow the object to add the *Menu* page to its settings notebook.

Parameters

self (WPFileSystem *) – input

The pointer to this object.

hwndNotebook (HWND) – input

Settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

Method should always be overridden in order to replace or remove the *Menu* page from the object's settings notebook.

To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddFileTypePage – WPDataFile instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpAddFileTypePage (WPDataFile * self, HWND hwndNotebook)

The wpAddFileTypePage instance method is called to allow the object to add the *Type* page to its settings notebook.

Parameters

self (WPDataFile *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Type* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpQueryType
- wpSetTitle

wpAddFile1Page – WPFileSystem instance method

```
#define INCL_WINWORKPLACE

ULONG wpAddFile1Page (WPFileSystem * self, HWND hwndNotebook)
```

The wpAddFile1Page instance method is called to allow the object to add the *File 1* page to its settings notebook.

Parameters

self (WPFileSystem *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *File 1* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddFile2Page – WPFileSystem instance method

#define INCL_WINWORKPLACE

ULONG wpAddFile2Page (WPFileSystem * self, HWND hwndNotebook)

The wpAddFile2Page instance method is called to allow the object to add the *File 2* page to its settings notebook.

Parameters

self (WPFileSystem *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *File 2* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddFile3Page – WPFileSystem instance method

#define INCL_WINWORKPLACE

ULONG wpAddFile3Page (WPFileSystem * self, HWND hwndNotebook)

The wpAddFile3Page instance method is called to allow the object to add the *File 3* page to its settings notebook.

Parameters

self (WPFileSystem *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *File 3* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddFolderBackgroundPage – WPFolder instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpAddFolderBackgroundPage (WPFolder * self, HWND hwndNotebook)

The wpAddFolderBackgroundPage instance method is called to allow the object to add the *Background* page to its settings notebook.

Parameters

self (WPFolder *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Background* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddFolderIncludePage — WPFolder instance method

#define INCL_WINWORKPLACE

ULONG wpAddFolderIncludePage (WPFolder * self, HWND hwndNotebook)

The wpAddFolderIncludePage instance method is called to allow the object to add the *Include* page to its settings notebook.

Parameters

self (WPFolder *) — input

The pointer to this object.

hwndNotebook (HWND) — input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Include* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddFolderSortPage – WPFolder instance method

```
#define INCL_WINWORKPLACE
```

```
ULONG wpAddFolderSortPage (WPFolder * self, HWND hwndNotebook)
```

The wpAddFolderSortPage instance method is called to allow the object to add the *Sort* page to its settings notebook.

Parameters

self (WPFolder *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Sort* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddFolderView1Page – WPFolder instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpAddFolderView1Page (WPFolder * self, HWND hwndNotebook)

The wpAddFolderView1Page instance method is called to allow the object to add the *View 1* page to its settings notebook.

Parameters

self (WPFolder *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *View 1* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddFolderView2Page – WPFolder instance method

#define INCL_WINWORKPLACE

```
ULONG wpAddFolderView2Page (WPFolder * self, HWND hwndNotebook)
```

The wpAddFolderView2Page instance method is called to allow the object to add the *View 2* page to its settings notebook.

Parameters

self (WPFolder *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *View 2* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddFolderView3Page – WPFolder instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpAddFolderView3Page (WPFolder * self, HWND hwndNotebook)

The wpAddFolderView3Page instance method is called to allow the object to add the *View 3* page to its settings notebook.

Parameters

self (WPFolder *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *View 3* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddKeyboardMappingsPage – WPKeyboard instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpAddKeyboardMappingsPage (WPKeyboard * self, HWND hwndNotebook)

The wpAddKeyboardMappingsPage instance method is called to allow the object to add the *Mappings* page to its settings notebook.

Parameters

self (WPKeyboard *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Mappings* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddKeyboardSpecialNeedsPage — WPKeyboard instance method

#define INCL_WINWORKPLACE

ULONG wpAddKeyboardSpecialNeedsPage (WPKeyboard * self, HWND hwndNotebook)

The wpAddKeyboardSpecialNeedsPage instance method is called to allow the object to add the *Special Needs* page to its settings notebook.

Parameters

self (WPKeyboard *) — input

The pointer to this object.

hwndNotebook (HWND) — input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Special Needs* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddKeyboardTimingPage — WPKeyboard instance method

```
#define INCL_WINWORKPLACE
```

```
ULONG wpAddKeyboardTimingPage (WPKeyboard * self, HWND hwndNotebook)
```

The wpAddKeyboardTimingPage instance method is called to allow the object to add the *Timing* page to its settings notebook.

Parameters

self (WPKeyboard *) — input

The pointer to this object.

hwndNotebook (HWND) — input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Timing* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddMouseMappingsPage – WPMouse instance method

#define INCL_WINWORKPLACE

ULONG wpAddMouseMappingsPage (WPMouse * self, HWND hwndNotebook)

The wpAddMouseMappingsPage instance method is called to allow the object to add the *Mappings* page to its settings notebook.

Parameters

self (WPMouse *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

The method should always be overridden in order to replace or remove the *Mappings* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddMouseTimingPage – WPMouse instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpAddMouseTimingPage (WPMouse * self, HWND hwndNotebook)

The wpAddMouseTimingPage instance method is called to allow the object to add the *Timing* page to its settings notebook.

Parameters

self (WPMouse *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Timing* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddMouseTypePage – WPMouse instance method

#define INCL_WINWORKPLACE

ULONG wpAddMouseTypePage (WPMouse * self, HWND hwndNotebook)

The wpAddMouseTypePage instance method is called to allow the object to add the *Type* page to its settings notebook.

Parameters

self (WPMouse *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Type* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddObjectGeneralPage – WPObject instance method

```
#define INCL_WINWORKPLACE
```

```
ULONG wpAddObjectGeneralPage (WPObject * self, HWND hwndNotebook)
```

The wpAddObjectGeneralPage instance method is called to allow the object to add the *General* page to its settings notebook.

Parameters

self (WPObject *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *General* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddProgramAssociationPage — WPProgramFile instance method

#define INCL_WINWORKPLACE

ULONG wpAddProgramAssociationPage (WPProgramFile * self, HWND hwndNotebook)

The wpAddProgramAssociationPage instance method is called to allow the object to add the *Association* page to its settings notebook.

Parameters

self (WPProgramFile *) — input

The pointer to this object.

hwndNotebook (HWND) — input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Association* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddProgramAssociationPage – WPProgram instance method

#define INCL_WINWORKPLACE

ULONG wpAddProgramAssociationPage (WPProgramFile * self, HWND hwndNotebook)

The wpAddProgramAssociationPage instance method is called to allow the object to add the *Association* page to its settings notebook.

Parameters

self (WPProgramFile *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Association* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddProgramPage – WPProgram instance method

#define INCL_WINWORKPLACE

ULONG wpAddProgramPage (WPProgram * self, HWND hwndNotebook)

The wpAddProgramPage instance method is called to allow the object to add the *Program* page to its settings notebook.

Parameters

self (WPProgram *) – input

The pointer to this object.

hwndNotebook (HWND) – input

Settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Program* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddProgramPage – WPProgramFile instance method

```
#define INCL_WINWORKPLACE
```

```
ULONG wpAddProgramPage (WPProgramFile * self, HWND hwndNotebook)
```

The wpAddProgramPage instance method is called to allow the object to add the *Program* page to its settings notebook.

Parameters

self (WPProgramFile *) – input

The pointer to this object.

hwndNotebook (HWND) – input

Settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Program* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddProgramSessionPage — WPProgram instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpAddProgramSessionPage (WPProgram * self, HWND hwndNotebook)

The wpAddProgramSessionPage instance method is called to allow the object to add the Session page to its settings notebook.

Parameters

self (WPProgram *) — input

The pointer to this object.

hwndNotebook (HWND) — input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the Session page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddProgramSessionPage – WPProgramFile instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpAddProgramSessionPage (WPProgramFile * self, HWND hwndNotebook)

The wpAddProgramSessionPage instance method is called to allow the object to add the *Session* page to its settings notebook.

Parameters

self (WPProgramFile *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Session* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddSettingsPages – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpAddSettingsPages (WPObject * self, HWND hwndNotebook)
```

The wpAddSettingsPages instance method is called to allow the object to add all of its settings pages to its settings notebook.

Parameters

self (WPObject *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Success indicator:

TRUE Successful completion.

FALSE Error occurred.

Usage

This method is generally only called by the system.

How to Override

This method should always be overridden in order to add pages to the settings notebook or to remove them. To add a page to the settings notebook, a call to the wpInsertSettingsPage method is required. To remove a page from the settings notebook, the method that adds the page to the settings notebook should be overridden and return *SETTINGS_PAGE_REMOVED* without calling its parent method.

In most cases, the parent method should be called first. Calling the parent method first will put pages added by this method at the top of the settings notebook, above the pages added by ancestor classes. Calling the parent last will put pages added by this class at the bottom of the settings notebook, below the pages added by ancestor classes.

Related Methods

See Notebook Control Window Messages for related messages.

- wpInsertSettingsPage

wpAddSoundWarningBeepPage – WPSound instance method

#define INCL_WINWORKPLACE

ULONG wpAddSoundWarningBeepPage (WPSound * self, HWND hwndNotebook)

The wpAddSoundWarningBeepPage instance method is called to allow the object to add the *Warning Beep* page to its settings notebook.

Parameters

self (WPSound *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Warning Beep* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddSystemConfirmationPage — WPSystem instance method

#define INCL_WINWORKPLACE

```
ULONG wpAddSystemConfirmationPage (WPSystem * self, HWND hwndNotebook)
```

The wpAddSystemConfirmationPage instance method is called to allow the object to add the *Confirmation* page to its settings notebook.

Parameters

self (WPSystem *) — input

The pointer to this object.

hwndNotebook (HWND) — input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Confirmation* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddSystemLogoPage – WPSystem instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpAddSystemLogoPage (WPSystem * self, HWND hwndNotebook)

The wpAddSystemLogoPage instance method is called to allow the object to add the *Logo* page to its settings notebook.

Parameters

self (WPSystem *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Logo* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddSystemPrintScreenPage – WPSystem instance method

#define INCL_WINWORKPLACE

ULONG wpAddSystemPrintScreenPage (WPSystem * self, HWND hwndNotebook)

The wpAddSystemPrintScreenPage instance method is called to allow the object to add the *Print Screen* page to its settings notebook.

Parameters

self (WPSystem *) – input

The pointer to the object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier.

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Print Screen* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddSystemWindowPage – WPSystem instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpAddSystemWindowPage (WPSystem * self, HWND hwndNotebook)

The wpAddSystemWindowPage instance method is called to allow the object to add the *Window* page to its settings notebook.

Parameters

self (WPSystem *) – input

The pointer to this object.

hwndNotebook (HWND) – input

The settings notebook handle.

Returns

Page identifier:

0 Error occurred.

PageId Identifier for the inserted page.

Usage

This method must only be called from within an override of the wpAddSettingsPages method.

How to Override

This method should always be overridden in order to replace or remove the *Window* page from the object's settings notebook. To remove the page from the settings notebook, the override method should return *SETTINGS_PAGE_REMOVED* without calling the parent method. To replace the page with another page, the override method should call the wpInsertSettingsPage method without calling the parent method.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages
- wpInsertSettingsPage

wpAddToObjUseList – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpAddToObjUseList (WPObject * self, PUSEITEM pUseItem)

The wpAddToObjUseList instance method is called to add a item type to an object's in-use list.

Parameters

self (WPObject *) – input

The pointer to this object.

pUseItem (PUSEITEM) – input

A pointer to a *USEITEM* structure.

Returns

Success indicator:

TRUE Successful completion.

FALSE Error occurred.

Remarks

This method will add a specified item type to an object's in-use (*USEITEM*) list. Every workplace object in the system has an in-use list. The in-use list is a linked list of *USEITEM* structures which provide the object with important information such as the number of container (*WC_CONTAINER*) windows it has been inserted into. It also provides the number of open views (contents, help, and settings) of itself that already exist and how much memory it has allocated. The *USEITEM* structure consists of an item type, a pointer to the next *USEITEM* structure, and is immediately followed by an item type-specific structure.

Usage

The following types of items can be added to the use list:

USAGE_MEMORY This item specifies a block of memory allocated for this object through the use of the wpAllocMem method. Items of this type add a *MEMORYITEM* structure to the end of the *USEITEM* structure.

USAGE_OPENVIEW When a view of an object is opened, one of these items is added to the in-use list. If multiple concurrent views are not enabled for this object, the *USAGE_OPENVIEW* items are used by the system to automatically switch to the open view when the user chooses to open the same view again. This behavior is controlled by the application by calling the wpOpen or the wpSwitchTo method. This item is also used by the system to update the title bar text and switch-entry text when the user changes the object title. If this type is specified, the in-use emphasis bit will be turned on for all inserted records for this object. Items of this type concatenate a *VIEWITEM* structure to the end of the *USEITEM* structure.

USAGE_RECORD For every view (*WC_CONTAINER*) window that the object is inserted into, there is one of these items on its in-use list. This enables an object to refresh its appearance in all views at the same time. Items of this type add a *RECORDITEM* structure to the end of the *USEITEM* structure.

wpAddToObjUseList – WPObject instance method

How to Override

This method is generally not overridden.

Related Methods

- wpDeleteFromObjUseList
- wpFindUseItem
- wpAllocMem
- wpClose

wpAllocMem – WPObject instance method

#define INCL_WINWORKPLACE

PBYTE wpAllocMem (WPObject * self, ULONG cbBytes, BOOL fReportError)

The wpAllocMem instance method is called to allocate memory for use by an object.

Parameters

self (WPObject *) – input

The pointer to this object.

cbBytes (ULONG) – input

Specifies the size of memory required.

fReportError (BOOL) – input

Report error indicator:

TRUE If an error occurs, the system will display an appropriate message.

FALSE The system will not display an error message.

Returns

Success indicator:

NULL Not enough memory available.

Other A pointer to the newly allocated memory.

Remarks

Memory allocated by the wpAllocMem method should be deallocated when it is no longer needed by calling the wpFreeMem method. Allocated memory not cleaned up by an object is deallocated automatically when the object is no longer in use.

Usage

This method can be called at any time to allocate memory to be used for this object only.

How to Override

This method should be overridden if a substitute memory allocation facility is to be used. Object classes overriding this method should also override the wpFreeMem method.

Related Methods

- wpFreeMem

wpClose –
WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpClose (WPObject * self)
```

The wpClose instance method is called to close all open views of an object.

Parameters

self (WPObject *) – input

The pointer to this object.

Returns

Success indicator:

TRUE Successful completion.

FALSE Error occurred.

Remarks

This method will cycle through the object's in-use list that was created by calls to the wpAddToObjUseList method. All open windows specified by *USAGE_OPENVIEW* items will be sent a *WM_CLOSE* message. Running executables specified by *USAGE_OPENVIEW* items will be terminated.

Usage

This method should be called to close all open views of the object and to free all allocated resources.

How to Override

If this function is overridden, it should call the parent last to ensure that allocated resources are properly deallocated.

Related Methods

- wpAddToObjUseList
- wpDeleteFromObjUseList
- wpFindUseItem
- wpOpen

wpCnrInsertObject – WPObject instance method

#define INCL_WINWORKPLACE

```
PMINIRECORDCORE wpCnrInsertObject (WPObject * self, HWND hwndCnr, PPOINTL pIcon,
                                   PMINIRECORDCORE pParent,
                                   PRECORDINSERT pRecInsert)
```

The wpCnrInsertObject instance method is called to insert a record into a container control window.

Parameters

self (WPObject *) – input

The pointer to this object.

hwndCnr (HWND) – input

The handle of container control window.

pIcon (PPOINTL) – input

The initial icon position in the container control window.

pParent (PMINIRECORDCORE) – input

A pointer to the parent record. *pParent* specifies the record of the immediate parent of the record specified by *pIcon*. This parameter should be set to NULL if the record has no parent or if tree view is not supported.

pRecInsert (PRECORDINSERT) – input

Record position:

NULL Insert the record into the next available position.

Other Insert the record into this position specified by *pRecInsert*.

Returns

Success indicator:

NULL Error occurred.

Other A pointer to the inserted record.

Remarks

This method will put an object into a container control window (*WC_CONTAINER*). These container windows can be application created or can be created by the system such as those in folders and settings notebooks.

The OBJECT_FROM_PREC (prec) macro can be used to determine the pointer to the object (WPObject *) that is associated with a given MINIRECORDCORE,prec, that was inserted using the wpCnrInsertObject method.

The USER_FROM_PREC(prec) macro can be used to access the application definable 32-bit field within the MINIRECORDCORE structure that is created when an object is put into a list control using the wpCnrInsertObject method.

A record inserted by this method will be added to the in-use list. See wpAddToObjUseList for more information on an object's in-use list. wpCnrRemoveObject should be called to remove each record before the container window is destroyed.

wpCnrInsertObject – WPObject instance method

Usage

This method is used to give workplace object behavior (such as context menu support) to records inserted directly into a *WC_CONTAINER* container control window. To remove the record from the container, a call to the wpCnrRemoveObject method should be made.

How to Override

This method is generally not overridden.

Related Methods

- wpAddToObjUseList
- wpCnrRemoveObject

wpCnrRemoveObject – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpCnrRemoveObject (WPObject * self, HWND hwndCnr, PMINIRECORDCORE pRecord)

The wpCnrRemoveObject instance method is called to remove a record from a container control window.

Parameters

self (WPObject *) – input

The pointer to this object.

hwndCnr (HWND) – input

The handle of the container control (*WC_CONTAINER*) window.

pRecord (PMINIRECORDCORE) – input

The pointer to the record to be removed.

Returns

Success indicator:

TRUE Successful completion.

FALSE Error occurred.

Remarks

This method causes the item to be removed from the in-use list.

Usage

This method should be called to remove records inserted by a call to the wpCnrInsertObject method.

How to Override

This method is generally not overridden.

Related Methods

- wpCnrInsertObject
- wpDeleteFromObjUseList

wpCnrSetEmphasis – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpCnrSetEmphasis (WPObject * self, ULONG ulEmphasisAttr, BOOL fTurnOn)

The wpCnrSetEmphasis instance method is called to allow the object to change its visual emphasis.

Parameters

self (WPObject *) – input

The pointer to this object.

ulEmphasisAttr (ULONG) – input

The CRA_* flags. For a detailed list, see RECORDCORE on page A-110.

fTurnOn (BOOL) – input

TRUE Set the specified attribute.

FALSE Reset the specified attribute.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This method changes all visual references to this object to show it with the specified emphasis attributes. This method is called automatically to set and reset CRA_INUSE emphasis state during the processing of the wpAddToObjUseList and wpDeleteFromObjUseList methods, respectively. This method is only valid for objects inserted into a container control using the wpCnrInsertObject method.

Usage

This method can be called at any time in order to set the emphasis for an object in an open folder. Some restrictions apply as to which *CRA_** can be used. See RECORDCORE on page A-110. for more information.

How to Override

This method is generally not overridden.

Related Methods

- wpAddToObjUseList
- wpCnrInsertObject
- wpOpen

wpConfirmDelete — WPObject instance method

#define INCL_WINWORKPLACE

```
ULONG wpConfirmDelete  (WPObject * self, ULONG fConfirmations)
```

The wpConfirmDelete instance method is called to allow the object to prompt the user to confirm the deletion of this object.

Parameters

self (WPObject *) — input

The pointer to this object.

fConfirmations (ULONG) — input

The confirmation flags.

CONFIRM_DELETE Confirm for delete of all objects.
CONFIRM_DELETEFOLDER Confirm for delete of folder objects only.

Returns

User-response value.

The default processing will return one of the following.

OK_DELETE	Delete of this object is confirmed.
NO_DELETE	Delete of this object only is cancelled.
CANCEL_DELETE	Deletion process from that point on is cancelled.

Remarks

This method is called during the processing of the wpDelete method only if the system confirm on delete flag is set.

Usage

This method is generally only called by the system.

How to Override

This method should be overridden in order to remove or replace the user prompt to confirm the delete operation.

Related Methods

- wpDelete
- wpFree

wpCopiedFromTemplate – WPObject instance method

#define INCL_WINWORKPLACE

```
VOID wpCopiedFromTemplate (WPObject * self)
```

The wpCopiedFromTemplate instance method is called to allow an object to perform class specific processing when a new object is created from a template.

Parameters

self (WPObject *) – input

The pointer to this object.

Returns

The return value is VOID.

Remarks

This method is called immediately after a new object is created from a template. The system does not do any default processing for this method. This method is typically overridden to perform class specific initialization on a created object. For example, a customer order form class would prefill the date, time, and order number in an instance of itself that was created from a template.

Usage

This method is generally only called by the system.

How to Override

This method should be overridden by object classes which need to initialize the new object after it is created from a template.

wpCopyObject — WPObject instance method

#define INCL_WINWORKPLACE

```
WPObject * wpCopyObject (WPObject * self, WPFolder * Folder, BOOL fLock)
```

The wpCopyObject instance method is called to create a new copy of the object.

Parameters

self (WPObject *) — input

The pointer to this object.

Folder (WPFolder *) — input

A pointer to a Folder object in which to place this new object. This pointer can be determined by issuing a call to the wpclsQueryFolder method.

fLock (BOOL) — input

The lock object flag.

If this flag is false, the newly created object will be made dormant whenever the object and the folder containing the object are closed. If this flag is true, the new object will remain active until the caller issues the *wpUnlockObject* method on it.

Returns

Success indicator:

NULL Error occurred.

Other A pointer to the new object created.

Remarks

Copies of an object can always be deleted and moved by default, even if the original has the OBJSTYLE_NODELETE or OBJSTYLE_NOMOVE style set.

Usage

This method can be called any time in order to create a copy of an existing object.

How to Override

This method can be overridden by classes which need to keep track of where instances are.

wpCreateFromTemplate – WPObject instance method

#define INCL_WINWORKPLACE

```
WPObject * wpCreateFromTemplate (WPObject * self, WPFolder * Folder, BOOL fLock)
```

The wpCreateFromTemplate instance method is called to create an object from a template.

Parameters

self (WPObject *) – input

The pointer to this object.

Folder (WPFolder *) – input

A pointer to a folder object in which to place the new object. This pointer can be determined by issuing a call to the wpclsQueryFolder method.

fLock (BOOL) – input

The lock object flag.

If this flag is false, the newly created object will be made dormant whenever the object and the folder containing the object are closed. If this flag is true, the new flag will remain active until the caller issues the wpUnlockObject method on it.

Returns

Success indicator:

NULL Error occurred.

Other Pointer to the new object.

Remarks

The new object will be an identical copy of the template object with the exception that the OBJSTYLE_TEMPLATE object style will be taken out. *wpCopiedFromTemplate* instance method will be called on the new object.

Usage

This method can be called at any time in order to create a new object from a template object.

How to Override

This method is generally not overridden.

Related Methods

- wpCopyObject
- wpclsNew

wpCreateShadowObject – WPObject instance method

```
#define INCL_WINWORKPLACE
```

```
WPObject * wpCreateShadowObject (WPObject * self, WPFolder * Folder, BOOL fLock)
```

The wpCreateShadowObject instance method is called to create a shadow of an object.

Parameters

self (WPObject *) – input

The pointer to this object.

Folder (WPFolder *) – input

A pointer to a folder object in which to place the new shadow object. This pointer can be determined by issuing a call to the wpclsQueryFolder method.

fLock (BOOL) – input
Lock object flag.

If this flag is false, the new shadow object will be made dormant whenever the object and the folder containing the object are closed.

If this flag is true, the new object will remain awake until the caller issues the wpUnlockObject method on it.

Returns

Success indicator:

NULL Error occurred.

OTHER Pointer to the new shadow object.

Remarks

The new object is created as an instance of class WPShadow.

Usage

This method can be called at any time in order to create a shadow object for this object.

How to Override

This method is generally not overridden.

wpDelete –
WPObject instance method

#define INCL_WINWORKPLACE

ULONG wpDelete (WPObject * self, ULONG ulConfirmations)

The wpDelete instance method is called to delete an object and prompt for confirmation if necessary.

Parameters

self (WPObject *) – input

The pointer to the object.

ulConfirmations (ULONG) – input

The confirmation flags:

CONFIRM_DELETE	Prompt for confirmation for all objects.
CONFIRM_DELETEFOLDER	Prompt for confirmation for just folder objects.
NULL	No confirmations.

Returns

Success indicator:

NO_DELETE	Error occurred.
CANCEL_DELETE	User canceled operation.
OK_DELETE	Object was deleted.

Remarks

The confirmation flags are passed to the wpConfirmDelete method. If wpConfirmDelete returns OK_DELETE, the wpFree method is called on the object.

Usage

This method can be called at any time in order to delete an object.

How to Override

This method is generally not overridden.

Related Methods

- wpFree
- wpConfirmDelete
- wpQueryConfirmations

wpDeleteAllJobs — WPPrinter instance method

#define INCL_WINWORKPLACE

```
BOOL wpDeleteAllJobs (WPPrinter * self)
```

The wpDeleteAllJobs instance method is called to delete all jobs in a printer (print destination) object.

Parameters

self (WPPrinter *) — input

The pointer to the object to delete all jobs.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method is generally not overridden.

Related Methods

- wpDeleteJob

wpDeleteContents – WPFolder instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpDeleteContents (WPFolder * self, ULONG ulConfirmations)

The wpDeleteContents instance method is called to delete the contents of a folder.

Parameters

self (WPFolder *) – input

The pointer to the object.

ulConfirmations (ULONG) – input

The confirmation flags:

CONFIRM_DELETE	Prompt for confirmation for all objects.
CONFIRM_DELETEFOLDER	Prompt for confirmation for just folder objects.
NULL	No confirmations.

Returns

Success indicator.

OK_DELETE	All objects were deleted.
NO_DELETE	Error occurred. At least one object was not deleted.
CANCEL_DELETE	User canceled operation.

Remarks

wpDelete and *wpFree* will automatically call this instance method for folder objects.

Usage

This method can be called at any time to delete the contents of a folder.

How to Override

This method is generally not overridden.

Related Methods

- wpFree
- wpDelete

wpDeleteFromObjUseList – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpDeleteFromObjUseList (WPObject * self, PUSEITEM pUseItem)

The wpDeleteFromObjUseList instance method is called to remove an item type from an object's in-use list.

Parameters

self (WPObject *) – input

The pointer to this object.

pUseItem (PUSEITEM) – input

A pointer to a *USEITEM* structure.

Returns

Success indicator:

TRUE Successful completion.

FALSE Error occurred.

Remarks

This method will remove a specified item type from an object's in-use (*USEITEM*) list.

If the usage item being removed is of type *USAGE_OPENVIEW*, the in-use emphasis bit will be turned off for all inserted records for this object.

Usage

A call to this method should be made when a particular usage item, added to the in-use list via the wpAddToObjUseList method, is no longer needed.

This method must be called before the USEITEM memory is freed.

How to Override

This method is generally not overridden.

Related Methods

- wpAddToObjUseList
- wpFindUseItem

wpDeleteJob – WPJob instance method

#define INCL_WINWORKPLACE

```
BOOL wpDeleteJob (WPJob * self)
```

The wpDeleteJob instance method is called to delete a job object.

Parameters

self (WPJob *) – input

The pointer to the object to be deleted.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method is generally not overridden.

Related Methods

- wpDeleteAllJobs

wpDisplayHelp — WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpDisplayHelp (WPObject * self, ULONG ulHelpPanelId, PSZ pszHelpLibrary)

The wpDisplayHelp instance method is called to allow the object to display a help panel.

Parameters

self (WPObject *) — input

The pointer to this object.

ulHelpPanelId (ULONG) — input

The object's help panel id.

pszHelpLibrary (PSZ) — input

A pointer to a zero terminated string which contains the name of the help library.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This method allows each object class to display a help panel in response to a users request for help. The help panel will be displayed by the shell on a help instance managed by the shell.

Usage

This method can be called at any time in order to display a help panel.

How to Override

This method is generally not overridden.

Related Methods

- wpMenuItemHelpSelected
- wpQueryDefaultHelp
- wpSetDefaultHelp
- wpclsQueryDefaultHelp

wpDoesObjectMatch – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpDoesObjectMatch (WPObject * object, PVOID pExtendedCriteria)

The wpDoesObjectMatch instance method is called to allow the object to determine if it matches the specified criteria.

Parameters

object (WPObject *) – input

The pointer to the object to be examined.

pExtendedCriteria (PVOID) – input

A pointer to a buffer that contains the class-specific extended search criteria.

Returns

Success indicator:

TRUE The object matches the specified criteria.

FALSE The object does not match the specified criteria.

Remarks

If the object class has extended search criteria, this method gets called to determine if the object found, matches the extended search criteria.

Usage

This method can be called at any time in order to determine if an object matches some extended search criteria.

How to Override

This method should be overridden by classes which introduce extended search criteria for use by the Find and Include facilities.

#define INCL_WINWORKPLACE

```
BOOL wpDragCell (WPPalette * self, PCELL pCell, HWND hwndPal, PPOINTL pptlDrag)
```

The wpDragCell instance method is called to drag-apply a value in a cell.

Parameters

self (WPPalette *) – input

The pointer to the object.

pCell (PCELL) – input

A pointer to the CELL structure to be dragged.

hwndPal (HWND) – input

The palette window handle.

pptlDrag (PPOINTL) – input

The point in hwndPal coordinates from which drag was initiated.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

Remarks

The default processing for this method by the WPPalette class is to do nothing other than return FALSE.

Usage

This method is generally called only by the palette window after it receives the WM_BEGINDRAG message.

How to Override

This method should be overridden to handle the drag-apply action.

Override processing should include capturing the mouse and, waiting for and processing the WM_ENDDRAG message

wpDraggedOverObject – WPObject instance method

#define INCL_WINWORKPLACE

MRESULT wpDraggedOverObject (WPObject * somSelf, WPObject * DraggedOverObject)

The wpDraggedOverObject instance method may be called on an object that is currently being dragged with the mouse to tell it what the current target object is. The return code from this method lets the system know whether the object being dragged can be dropped on the specified target.

Parameters

somSelf (WPObject *) – input

The pointer to an object being dragged.

DraggedOverObject (WPObject *) – input

Pointer to the object that the drag cursor is over, the current target object for the drag operation.

Returns

LowWord DropIndicator (USHORT).

This code is one of the DOR_ constants, such as DOR_DROP or DOR_NODROP which indicate whether a drop is allowed on the current target.

HighWord DropOperation (USHORT).

The current drag operation code. Examples are DO_COPY, DO_MOVE or DO_LINK to indicate that the drag action over this target should be a copy, move or a link.

Remarks

When a target object is dragged over by the mouse, it will always receive a wpDragOver instance method call. Many target objects will choose to decide the current drag operation and whether a drop is possible based upon their own rules. For instance, the *WPShredder* object will return DO_DROP,DO_MOVE if it decides that all the source objects can be deleted. However, some targets require the source or sources to participate in the decision over whether they can accept the drop. The way that a target allows a source object to have a say in what the drop action will be is by calling the wpDraggedOverObject on each source object. The wpDraggedOverObject instance method may be invoked on an object that is being dragged (source object) at any time, to see if it can support a drop on the current target. If the object that is being dragged responds favorably to this method, it may later receive a wpDroppedOnObject instance method call so that it can process the drop action.

As an example, consider the case where a program object is dragged onto a data file object. The program would respond DO_DROP to the *wpDraggedOverObject* instance method, so that the data file would be a valid drop target. If the user chose to allow the drop, then the program will receive a *wpDroppedOnObject* instance method at which time it would be able to open itself as a viewer of the data file object.

This method is called as a result of a DM_DRAGOVER message being sent, and for further documentation of the possible return values, see "DM_DRAGOVER" on page 29-4.

wpDraggedOverObject – WPObject instance method

Usage

This method is typically called by objects that require participation from the source object when a drop occurs. The method can be called at any time, however the method would normally only be called by a target object on one of the source objects during a drag or drop operation.

How to Override

Override this method if your object class wishes to allow itself to be used as a source object that can perform a drop operation. A favorable return code from this method may lead to a *wpDroppedOnObject* instance method being invoked on the source object which would be overridden to actually do the drop operation.

Related Methods

- wpDragOver
- wpDrop
- wpDroppedOnObject
- wpFormatDragItem

wpDragOver – WPObject instance method

#define INCL_WINWORKPLACE

MRESULT wpDragOver (WPObject * self, HWND hwndCnr, PDRAGINFO pDraginfo)

The wpDragOver instance method is called to inform the object that other objects are being dragged over it.

Parameters

self (WPObject *) – input

The pointer to this object.

hwndCnr (HWND) – input

The handle to the container control window.

pDraginfo (PDRAGINFO) – input

A pointer to drag information.

Returns

See "DM_DRAGOVER" on page 29-4 for description of the return value.

Remarks

The wpDragOver method is sent for each DM_DRAGOVER message received by the object. See "DM_DRAGOVER" on page 29-4 for more information.

Usage

This method is generally only called by the system as the folder containing the object processes the DM_DRAGOVER message.

How to Override

This method should be overridden to determine if the object or objects being dragged can be dropped on this object.

Related Methods

- wpDrop
- wpFormatDragItem

#define INCL_WINWORKPLACE

```
MRESULT wpDrop  (WPObject * self, HWND hwndCnr, PDRAGINFO pDraginfo,
                 PDRAGITEM pDragItem)
```

The wpDrop instance method is called to inform an object that another object has been dropped on it.

Parameters

self (WPObject *) – input

The pointer to this object.

hwndCnr (HWND) – input

The handle to the container control window.

pDragInfo (PDRAGINFO) – input

A pointer to a PDRAGINFO structure.

pDragItem (PDRAGITEM) – input

A pointer to a PDRAGITEM structure.

Returns

See "DM_DROP" on page 29-6 for description of the return value.

Remarks

The wpDrop method is called when a DM_DROP message is received by the object. See "DM_DROP" on page 29-6 for more information.

Usage

This method is generally only called by the system as a folder containing the object processed the DM_DROP message.

How to Override

This method should be overridden to process the action of the dragged object or objects being dropped on it.

Related Methods

- wpDragOver
- wpFormatDragItem

wpDroppedOnObject – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpDroppedOnObject (WPObject * somSelf, WPObject * DroppedOnObject)
```

The wpDroppedOnObject instance method is called on an object that has just been dragged (a source object) when the target object that it was dropped on does not know what action to perform. This instance method is only called on a source object when that source object has previously responded favorably to a wpDraggedOverObject instance method call.

Parameters

somSelf (WPObject *) – input

The pointer to an object being dragged.

DroppedOnObject (WPObject *) – input

Pointer to the object that was dropped on. The current target object for the drag operation.

Returns

TRUE The drop action was successful.

FALSE An error occurred.

Remarks

When you drop on an a target object and the source object has said it knows how to handle the drop operation, this method will be invoked on the source object. For example, the program object class supports being dropped on certain other classes of object where the drop action will be taken by the program itself. The drop action provided in *WPProgram's* override of the *wpDroppedOnObject* method would be to execute itself as a viewer of the target object. This method is called as a result of a DM_DROP message being sent.

Usage

Target objects that do not know how to handle the current drop operation can call this method on the source object that was dropped on it. This method would not normally be called outside the scope of a drag or drop operation.

How to Override

Object classes that override the *wpDraggedOverObject* instance method would normally be expected to also override this method.

Related Methods

- wpDragOver
- wpDraggedOverObject
- wpDrop
- wpFormatDragItem

#define INCL_WINWORKPLACE

```
BOOL wpEditCell (WPPalette * self, PCELL pCell, HWND hwndPal)
```

The wpEditCell instance method is called to edit a value in a cell.

Parameters

self (WPPalette *) – input

The pointer to the object.

pCell (PCELL) – input

A pointer to the CELL structure to be edited.

hwndPal (HWND) – input

The palette window handle.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

Remarks

The default processing for this method by the WPPalette class is to do nothing other than return FALSE.

Usage

This method is generally called only by the palette window when the user requests to edit the value in the cell. This request is made by selecting a cell and pressing the Enter key or the edit pushbutton, or by double-clicking on the cell.

How to Override

This method should be overridden to handle the edit action.

wpEndConversation – WPObject instance method

#define INCL_WINWORKPLACE

MRESULT wpEndConversation (WPObject * self, ULONG ulItemId, ULONG ulResult)

The wpEndConversation instance method is called to notify the object that the drag or drop operation is complete.

Parameters

self (WPObject *) – input

The pointer to the object.

ulItemId (ULONG) – input

This is the ulItemID from the DRAGITEM that was contained within the DRAGINFO structure when the object was dropped.

ulResult (ULONG) – input

Flag indicating whether the operation was performed successfully. See "DM_ENDCONVERSATION" on page 29-8 for more information about this parameter.

Returns

Refer to the DM_ENDCONVERSATION message for a description of the return value.

Remarks

The wpEndConversation method is called when the object receives a DM_ENDCONVERSATION message. Refer to the DM_ENDCONVERSATION message for more information.

Usage

This method is generally called only by the system as the folder containing the object processed by the DM_ENDCONVERSATION message.

How to Override

This method should be overridden, if the object needs to do any actions once the drag or drop operation is complete.

Related Methods

- wpDragOver
- wpDrop
- wpFormatDragItem
- wpRender
- wpRenderComplete

wpFilterPopupMenu — WPObject instance method

#define INCL_WINWORKPLACE

```
ULONG wpFilterPopupMenu  (WPObject * self, ULONG ulFlags, HWND hwndcnr,
                          BOOL fMultiSelect)
```

The wpFilterPopupMenu instance method is called to allow the object to modify its context menu.

Parameters

self (WPObject *) — input

The pointer to this object.

ulFlags (ULONG) — input

If the flag is set, the corresponding pop-up menu item will be available. These flags are *ored* together with the flags already defined by ancestor classes, to specify the standard pop-up menu items which apply to this object.

CTXT_NEW	Create another
CTXT_OPEN	Open
CTXT_SWITCHTO	Switch to
CTXT_CLOSE	Close
CTXT_SETTINGS	Open settings
CTXT_PRINT	Print
CTXT_HELP	Help
CTXT_DELETE	Delete
CTXT_COPY	Copy
CTXT_MOVE	Move
CTXT_LINK	Create shadow
CTXT_WINDOW	Window

hwndcnr (HWND) — input

The handle to container control window.

fMultiSelect (BOOL) — input

The multiple menu items flag.

Returns

New pop-up menu flags for this object.

wpFilterPopupMenu – WPObject instance method

Usage

This method is generally only called by the system when a request is made to display the object's pop-up window.

How to Override

This method should be overridden to remove undesired pop-up menu actions that were added by ancestor classes. The parent method should be called prior to any override processing.

Related Methods

- wpMenuItemSelected
- wpMenuItemHelpSelected
- wpModifyPopupMenu
- wpInsertPopupMenuItems

wpFindUseItem – WPObject instance method

#define INCL_WINWORKPLACE

PUSEITEM wpFindUseItem (WPObject * self, ULONG ultype, PUSEITEM pCurrentUseItem)

The wpFindUseItem instance method is called to retrieve an item type from the object's in-use list.

Parameters

self (WPObject *) – input

The pointer to this object.

ultype (ULONG) – input

Specify the usage type of the item to be located.

pCurrentUseItem (PUSEITEM) – input
A pointer to a *USEITEM* structure:

NULL Retrieve the first item in the in-use list that has a usage type of *ultype*.

Other Retrieve the next item in the in-use list, following the item specified by *pCurrentUseItem*, that has a usage type of *ultype*.

Returns

Success indicator:

NULL No items matching the specified find criteria were found in the in-use list for this object.

Other A pointer to a *USEITEM* structure that matches the specified find criteria.

Remarks

This method will search the object's in-use list for all items that were added by previous calls to the wpAddToObjUseList method.

Usage

This method should be called to determine how the object is currently being used, for example, which views are currently open and what container window it is inserted into.

How to Override

This method is generally not overridden.

Related Methods

- wpAddToObjUseList
- wpDeleteFromObjUseList

wpFormatDragItem – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpFormatDragItem (WPObject * self, PDRAGITEM pDragItem)

The wpFormatDragItem instance method is called to allow the object to format its drag information when the user starts to drag it.

Parameters

self (WPObject *) – input

The pointer to this object.

pDragItem (PDRAGITEM) – input

The address of the drag item.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This method will enable the direct manipulation of this object by initializing the DRAGITEM structure.

Usage

This method is generally only called by the system when the user first starts to drag the object.

How to Override

This method is generally overridden by classes which require special processing to initiate a drag or drop operation.

Related Methods

- wpDragOver
- wpDrop

wpFree – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpFree (WPObject * self)
```

The wpFree instance method is called to destroy the object and deallocate its associated resources.

Parameters

self (WPObject *) – input

The pointer to this object.

Returns

Success indicator:

True Successful completion

False Error occurred.

Remarks

This method destroys the persistent form of the object and then frees the memory that represented that object. If confirmations are on, wpDelete will prompt the user before calling wpFree.

Usage

General destruction of an object should be done with the wpDelete method. This method is generally only called by the system.

How to Override

This method is generally overridden by storage classes which permanently remove this object and its associated data, or any objects that need to do special processing before deletion.

Related Methods

- wpDelete
- wpUnInitData

wpFreeMem – WPObject instance method

```
#define INCL_WINWORKPLACE
```

```
BOOL wpFreeMem (WPObject * self, PBYTE pbMemory)
```

The wpFreeMem instance method is called to deallocate memory allocated by a call to the wpAllocMem method.

Parameters

self (WPObject *) – input

The pointer to this object.

pbMemory (PBYTE) – input

The pointer to the memory to be deallocated.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This method deallocates memory for an object. wpFreeMem should always be called when the memory allocated by wpAllocMem is no longer needed.

Usage

This method should be called when the memory allocated by a call to the wpAllocMem method is no longer needed.

How to Override

This method should only be overriden to provide the deallocation of memory allocated by an override method of wpAllocMem.

Related Methods

- wpAllocMem

wpHide – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpHide  (WPObject * self)
```

The wpHide instance method is called to hide or minimize open views of this object.

Parameters

self (WPObject *) – input

The pointer to this object.

Returns

Success indicator:

True Successful completion

False Error occurred.

Remarks

This method will turn all windows owned by this object invisible or if the system setting is set to provide minimized windows instead of hidden windows, wpHide will minimize all windows owned by this object.

Usage

This method can be called to hide an object's window.

How to Override

This method is not generally overridden.

Related Methods

- wpRestore

wpHideFldrRunObjs – WPFolder instance method

#define INCL_WINWORKPLACE

```
BOOL wpHideFldrRunObjs (WPFolder * self, BOOL fHide)
```

The wpHideFldrRunObjs instance method is called to allow the folder to hide or minimize its open objects.

Parameters

self (WPFolder *) – input

The pointer to this object.

fHide (BOOL) – input

Flag to indicate hide or show of open objects.

True Hide all objects opened from this folder.

False Unhide all objects opened from this folder.

Returns

Success indicator:

True Successful completion

False Error occurred.

Remarks

Open objects in this folder will either be hidden or minimized, depending on the current system setting. This method is called automatically on folders with the work area flag, FOI_WORKAREA, set when it is hidden or minimized.

Usage

This method can be called at any time in order to hide or minimize all objects in this folder which are currently open.

How to Override

This method is generally not overridden.

wpHoldJob – WPJob instance method

#define INCL_WINWORKPLACE

```
BOOL wpHoldJob (WPJob * self)
```

The wpHoldJob instance method is called to hold a job object.

Parameters

self (WPJob *) – input

The pointer to the object to be held.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method is generally not overridden.

Related Methods

- wpReleaseJob

wpHoldPrinter – WPPrinter instance method

```
#define INCL_WINWORKPLACE
```

```
BOOL wpHoldPrinter (WPPrinter * self)
```

The wpHoldPrinter instance method is called to hold a print object.

Parameters

self (WPPrinter *) – input

The pointer to the object to be held.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method is generally not overridden.

Related Methods

- wpReleasePrinter

#define INCL_WINWORKPLACE

```
VOID wpInitData (WPObject * self)
```

The wpInitData instance method is called to allow the object to initialize its instance data.

Parameters

self (WPObject *) – input

The pointer to this object.

Returns

The return value is VOID.

Remarks

This routine is called when the object is created or when it is awakened from the dormant state so that it can initialize all of its instance variables to a known state. Note that this method is called before the object's state is known, so it is very important that the object does not try to process any other method while processing this method. Should an object require extra initialization that requires it to invoke other methods, this should be done from the wpRestoreState instance method. When the object is first created, the wpSetup instance method should be overridden to perform initialization that is only required once.

Usage

This method is generally only called by the system when the object is awake.

How to Override

Any class that has instance variables should override this method so that those variables are all initially in a known state. It is essential to pass this method onto the parent class. If this method is overridden, the wpUnInitData method should also be overridden in order to deallocate resources that were allocated by the override processing of wpInitData.

Related Methods

- wpSetup
- wpUnInitData

wpInsertPopupMenuItems – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpInsertPopupMenuItems (WPObject * self, HWND hwndMenu, ULONG ulPostition,
                             HMODULE hmod, ULONG ulMenuID, ULONG ulSubMenuId)
```

The wpInsertPopupMenuItems instance method is called to allow an object to insert items into its popup menu.

Parameters

self (WPObject *) – input

The pointer to this object.

hwndMenu (HWND) – input

A handle to the popup menu.

ulPostition (ULONG) – input

Position at which to start inserting items.

hmod (HMODULE) – input

Module handle where ulMenuID can be found.

ulMenuID (ULONG) – input

Id of menu to put into popup menu.

ulSubMenuId (ULONG) – input

Id of submenu to put into popup menu.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This instance method will insert all menu items in *ulMenuID* into the pop-up menu.

Menu item ids in open cascade must match corresponding openview.

Class specific menu IDs should be above *WPMENUID_USER*.

Usage

This method can be called only during the processing of wpModifyPopupMenu.

How to Override

This method is generally not overridden.

Related Methods

- wpMenuItemSelected
- wpMenuItemHelpSelected
- wpModifyPopupMenu
- wpFilterPopupMenu

wpInsertSettingsPage – WPObject instance method

#define INCL_WINWORKPLACE

ULONG wpInsertSettingsPage (WPObject * self, HWND hwndNotebook, PPAGEINFO ppageinfo)

The wpInsertSettingsPage instance method is called to insert a page into the object's settings notebook.

Parameters

self (WPObject *) – input

The pointer to this object.

hwndNotebook (HWND) – input

A handle to the setting notebook.

ppageinfo (PPAGEINFO) – input

A pointer to the notebook page information.

Returns

Page identifier:

0 Error occurred.

PageId PageId identifier for the inserted page.

Usage

This method can be called only during the processing of the wpAddSettingsPages method.

How to Override

This method is generally not overridden.

Related Methods

See Notebook Control Window Messages for related messages.

- wpAddSettingsPages

wpIsCurrentDesktop – WPDesktop instance method

#define INCL_WINWORKPLACE

```
BOOL wpIsCurrentDesktop (WPDesktop * self)
```

The wpIsCurrentDesktop instance method allows the desktop to specify whether it is the active desktop folder on the system.

Parameters

self (WPDesktop *) – input

The pointer to this object.

Returns

Active desktop indicator:

True This object is the active desktop.

False This object is not the active desktop and should behave as a normal folder.

Remarks

The active desktop is set by the system every time the user profile is reset by a call to the PrfReset function. Since there can only be one desktop and objects cannot change their class, desktop objects and descendents must call wpIsCurrentDesktop to determine if it is the current desktop. Desktop folders which are not active take on the behavior of standard folder objects.

Usage

This method is called at the beginning of every overridden method. If the return is false, the override method should call its parent without doing any override processing. If the return is true, override processing can be made.

How to Override

This method is generally not overridden.

wpMenuItemHelpSelected – WPObject instance method

```
#define INCL_WINWORKPLACE
```

BOOL wpMenuItemHelpSelected (WPObject * self, ULONG ulMenuId)

The wpMenuItemHelpSelected instance method is called to allow the object to display the requested help panel.

Parameters

self (WPObject *) – input

The pointer to this object.

ulMenuId (ULONG) – input

An unsigned short containing the object's menu id.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

The default WPObject class doesn't process this method at all other than to return false.

Usage

This method is generally only called by the system when help on a popup menu item is requested.

How to Override

This method should be overridden in order to display an appropriate help panel for the specified menu item. This is typically accomplished by issuing a call to the wpDisplayHelp method.

Related Methods

- wpDisplayHelp
- wpQueryDefaultHelp
- wpSetDefaultHelp
- wpclsQueryDefaultHelp
- wpMenuItemSelected
- wpModifyPopupMenu
- wpFilterPopupMenu
- wpInsertPopupMenuItems

wpMenuItemSelected — WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpMenuItemSelected (WPObject * self, HWND hwndFrame, ULONG ulMenuId)

The wpMenuItemSelected instance method is called to allow an object to process a pop-up menu selection.

Parameters

self (WPObject *) — input

The pointer to this object.

hwndFrame (HWND) — input

Handle to the frame window.

ulMenuId (ULONG) — input

ID of selected pop-up menu selected.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

Class specific Menu Ids should be above WPMENUID_USER.

Usage

This method is generally only called by the system when a new pop-up menu on all item is selected.

How to Override

This method should be overridden to process class-specific menu item actions or to modify the behavior of a menu item action provided by an ancestor class.

Related Methods

- wpMenuItemHelpSelected
- wpModifyPopupMenu
- wpFilterPopupMenu
- wpInsertPopupMenuItems

wpModifyPopupMenu – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpModifyPopupMenu (WPObject * self, HWND hwndMenu, HWND hwndCnr,
                        ULONG ulPosition)
```

The wpModifyPopupMenu instance method is called to allow an object to add additional items to its pop-up menu.

Parameters

self (WPObject *) – input

The pointer to this object.

hwndMenu (HWND) – input

The menu handle.

hwndCnr (HWND) – input

The handle to container control window.

ulPosition (ULONG) – input

The position to insert menu items.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

Class specific Menu Ids should be above WPMENUID_USER. This method will only be called if the current pop-up menu applies to objects of the same class.

Usage

This method is generally called by the system when a request to display the object's pop-up menu is made. This method is called following a call to the wpFilterPopupMenu method.

How to Override

This method should be overridden in order to add class-specific actions to the object's pop-up menu. Descendent classes can remove these actions by processing the wpFilterPopupMenu method.

Related Methods

- wpMenuItemSelected
- wpMenuItemHelpSelected
- wpFilterPopupMenu
- wpInsertPopupMenuItems

wpMoveObject – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpMoveObject (WPObject * self, WPFolder * Folder)

The wpMoveObject instance method is called to move the object to a different location.

Parameters

self (WPObject *) – input

The pointer to this object.

Folder (WPFolder *) – input

A pointer to a folder object in which to move this object into. This pointer can be determined by issuing a call to wpclsQueryFolder method.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Usage

This method can be called at any time in order to move an object to a new location.

How to Override

This method is generally not overridden except by storage classes. The parent should be called last unless special actions need to take place when an object is moved.

Related Methods

- wpCopyObject
- wpCreateShadowObject

wpOpen –
WPObject instance method

```
#define INCL_WINWORKPLACE
```

HWND wpOpen (WPObject * self, HWND hwndCnr, ULONG ulView, ULONG ulparam)

The wpOpen instance method is called to open a view to the object.

Parameters

self (WPObject *) – input

The pointer to this object.

hwndCnr (HWND) – input

Handle of the container window which the object is opened from. This value may be set to NULLHANDLE.

ulView (ULONG) – input

Specifies which view to open.

OPEN_CONTENTS	Open content view.
OPEN_DEFAULT	Open default view (same as double-click).
OPEN_DETAILS	Open details view.
OPEN_HELP	Display HelpPanel.
OPEN_RUNNING	Execute object.
OPEN_SETTINGS	Open settings notebook.
OPEN_TREE	Open tree view.
OPEN_USER	Class specific views have a greater value than this.

ulparam (ULONG) – input

Open view parameter.

This value is (reserved = NULL) for views supported by the WPObject class.

Returns

Success indicator:

NULLHANDLE	Error occurred.
Other	Handle to either window created or program executed.

Usage

This method can be called at any time in order to open a view of an object.

How to Override

This method should be overridden in order to process class-specific open views. This method can also be overridden in order to modify the behavior defined an ancestor class.

Related Methods

- wpClose

wpPaintCell – WPPalette instance method

#define INCL_WINWORKPLACE

BOOL wpPaintCell (WPPalette * self, PCELL pCell, HPS hps, PRECTL pprcl, BOOL fHilite)

The wpPaintCell instance method is called to paint a cell.

Parameters

self (WPPalette *) – input

The pointer to the object.

pCell (PCELL) – input

A pointer to the CELL structure to be painted.

hps (HPS) – input

A presentation space handle for the cell.

pprcl (PRECTL) – input

A pointer to the RECTL structure for the area to be painted.

fHilite (BOOL) – input

A flag to indicate selected state.

TRUE Cell is currently selected.

FALSE Cell is not currently selected.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

Remarks

The default processing for this method by the WPPalette class is to paint a SYSCLR_WINDOW background. If the cell is highlighted, a SYSCLR_HIGHLITEBACKGROUND background is painted.

Usage

This method is can be called at any time in order to paint a cell.

How to Override

This method should be overriden by all subclasses that want to display visual information in the cell window. It is recommended that the parent method be called first.

wpPopulate – WPFolder instance method

#define INCL_WINWORKPLACE

```
BOOL wpPopulate  (WPFolder * self, ULONG ulReserved, WPFolder * Folder,
                  BOOL fFoldersOnly)
```

The wpPopulate instance method is called to allow the folder to populate itself.

Parameters

self (WPFolder *) – input

The pointer to this object.

ulReserved (ULONG) – input

Reserved value must be 0.

Folder (WPFolder *) – input

The real name of the folder to populate.

fFoldersOnly (BOOL) – input

The *ored* flag indicating type of contents with which to populate folder:

TRUE Populate with folder objects only.

FALSE Populate with entire contents of folder.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

The wpPopulate method sets the folder flags depending upon the value of fFoldersOnly.

The folder flags indicate what the current population state of the folder is:

FOI_POPULATEDWITHALL folder is completely populated.
FOI_POPULATEDWITHFOLDERS folder is populated only with subfolders.

If the folder is re-populated when it has already been populated (as determined by inspecting the folder flags), no action is taken on this message apart from sending the notification message back.

Usage

This method is generally called only by the system when the folder is opened.

How to Override

This method can be overridden to alter the contents of a folder. To filter contents added by ancestor classes, it is important to call the parent method first.

wpPrintJobNext – WPJob instance method

```
#define INCL_WINWORKPLACE
```

```
BOOL wpPrintJobNext  (WPJob  *  self)
```

The wpPrintJobNext instance method is called to print a job next.

Parameters

self (WPJob *) – input

The pointer to the object to be printed next.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method is generally not overridden.

wpPrintMetaFile – WPDataFile instance method

#define INCL_WINWORKPLACE

```
BOOL wpPrintMetaFile (WPDataFile * self, PPRINTDEST pPrintDest)
```

The wpPrintMetaFile instance method is called to print an object of type "MetaFile."

Parameters

self (WPDataFile *) – input

The pointer to this object.

pPrintDest (PPRINTDEST) – input

The pointer to the print data.

It contains all the parameters required to issue a DevPostDeviceModes and DevOpenDC function calls.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method can be overridden by any object class that wants to replace the system supplied MetaFile print method.

Related Methods

- wpPrintObject

wpPrintObject — WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpPrintObject (WPObject * self, PPRINTDEST pPrintDest, ULONG ulReserved)

The wpPrintObject instance method is called to print a view of the object.

Parameters

self (WPObject *) — input

The pointer to this object.

pPrintDest (PPRINTDEST) — input

The pointer to print data.

It contains all the parameters required to issue a DevPostDeviceModes and DevOpenDC function calls.

ulReserved (ULONG) — input

Reserved value must be 0.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Usage

This method can be called at any time in order to print a view of an object.

How to Override

This method should be overridden in order to modify the print behavior supported by an ancestor class.

Related Methods

- wpPrintMetaFile
- wpPrintPifFile
- wpPrintPlainTextFile
- wpPrintPrinterSpecificFile
- wpPrintUnknownFile

wpPrintPifFile – WPDataFile instance method

```
#define INCL_WINWORKPLACE
```

```
BOOL wpPrintPifFile (WPDataFile * self, PPRINTDEST pPrintDest)
```

The wpPrintPifFile instance method is called to print an object of type "Pif."

Parameters

self (WPDataFile *) – input

The pointer to this object.

pPrintDest (PPRINTDEST) – input

The pointer to the print data.

It contains all the parameters required to issue a DevPostDeviceModes and DevOpenDC function calls.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method can be overridden by any object class that wants to replace the system supplied Pif file print method.

Related Methods

- wpPrintObject

wpPrintPlainTextFile – WPFileSystem instance method

#define INCL_WINWORKPLACE

BOOL wpPrintPlainTextFile (WPFileSystem * self, PPRINTDEST pPrintDest)

The wpPrintPlainTextFile instance method is called to print an object of type "Plain Text."

Parameters

self (WPFileSystem *) – input

The pointer to this object.

pPrintDest (PPRINTDEST) – input

The pointer to the print data.

It contains all the parameters required to issue a DevPostDeviceModes and DevOpenDC function calls.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method can be overridden by any object class that wants to replace the system supplied Plain Text File print method.

Related Methods

- wpPrintObject

wpPrintPrinterSpecificFile – WPDataFile instance method

#define INCL_WINWORKPLACE

```
BOOL wpPrintPrinterSpecificFile (WPDataFile * self, PPRINTDEST pPrintDest)
```

The wpPrintPrinterSpecificFile instance method is called to print an object of type "Printer-specific Data."

Parameters

self (WPDataFile *) – input

The pointer to this object.

pPrintDest (PPRINTDEST) – input

The pointer to the print data.

It contains all the parameters required to issue a DevPostDeviceModes and DevOpenDC function calls.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method can be overridden by any object class that wants to replace the system supplied Printer-specific File print method.

Related Methods

- wpPrintObject

wpPrintUnknownFile – WPDataFile instance method

#define INCL_WINWORKPLACE

```
BOOL wpPrintUnknownFile (WPDataFile * self, PPRINTDEST pPrintDest)
```

The wpPrintUnknownFile instance method is called to print an object of "unknown" type.

Parameters

self (WPDataFile *) – input

The pointer to this object.

pPrintDest (PPRINTDEST) – input

The pointer to the print data.

It contains all the parameters required to issue a DevPostDeviceModes and DevOpenDC function calls.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method can be overridden by any object class that wants to replace the system supplied Unknown File print method.

Related Methods

- wpPrintObject

wpQueryAssociationFilter – WPProgram instance method

#define INCL_WINWORKPLACE

PSZ wpQueryAssociationFilter (WPProgram * self)

The wpQueryAssociationFilter instance method is called to determine which file title filters are used to associate data file objects to this program object.

Parameters

self (WPProgram *) – input

The pointer to the object.

Returns

Success indicator:

NULL Error occurred.

Other A pointer to a string containing file title filter(s). This string can contain several file title filters separated by a comma.

Example: "*.TXT, *.DOC"

Remarks

The association filter is used to designate this program as an available open view for data file objects which have a title that matches one of the association filters which are set. If a data file object matches a filter in a program object or program file object, the title of the program object or program file object will appear in the data file object's Open cascade of its pop-up menu. The wpQueryAssociationFilter method will return the filter string set by the last call to the wpSetAssociationFilter method. The wpQueryAssociationType method can be called to determine which file types are used to associate data file objects to this program object.

Usage

This method can be called at any time in order to determine which file title filters are used to associate data file objects to this program object.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryAssociationType
- wpSetAssociationFilter
- wpSetAssociationType

wpQueryAssociationFilter – WPProgramFile instance method

#define INCL_WINWORKPLACE

```
PSZ wpQueryAssociationFilter (WPProgramFile * self)
```

The wpQueryAssociationFilter instance method is called to determine which file title filters are used to associate data file objects to this program object.

Parameters

self (WPProgramFile *) – input

The pointer to the object.

Returns

Success indicator:

NULL Error occurred.

Other A pointer to a string containing file title filters. This string can contain several file title filters separated by a comma.

Example: "*.TXT, *.DOC"

Remarks

The association filter is used to designate this program as an available open view for data file objects which have a title that matches one of the association filters which are set. If a data file object matches a filter in a program object or program file object, the title of the program object or program file object will appear in the data file object's Open cascade of its pop-up menu. The wpQueryAssociationFilter method will return the filter string set by the last call to the wpSetAssociationFilter method. The wpQueryAssociationType method can be called to determine which file types are used to associate data file objects to this program object.

Usage

This method can be called at any time in order to determine which file title filters are used to associate data file objects to this program object.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryAssociationType
- wpSetAssociationFilter
- wpSetAssociationType

wpQueryAssociationType – WPProgram instance method

```
#define INCL_WINWORKPLACE
```

```
PSZ wpQueryAssociationType (WPProgram * self)
```

The wpQueryAssociationType instance method is called to determine which file types are used to associate data file objects to this program object.

Parameters

self (WPProgram *) – input

The pointer to the object.

Returns

Success indicator:

NULL Error occurred.

Other A pointer to a string containing file type(s). This string can contain several file title filters separated by a comma.

Example: "Plain Text,C Code"

Remarks

The association type is used to designate this program as an available open view for data file objects which have a type that matches one of the association types which are set. If a data file object matches a type in a program object or program file object, the title of the program object or program file object will appear in the data file object's Open cascade of its popup menu.

The wpQueryAssociationType method will return the type string set by the last call to the wpSetAssociationType method.

The wpQueryAssociationFilter method can be called to determine which file title filters are used to associate data file objects to this program object.

Usage

This method can be called at any time in order to determine which file types are used to associate data file objects to this program object.

How to Override

This method is generally not overridden.

Related Methods

- wpSetAssociationType
- wpSetType
- wpQueryType

wpQueryAssociationType – WPProgramFile instance method

#define INCL_WINWORKPLACE

```
PSZ wpQueryAssociationType  (WPProgramFile * self)
```

The wpQueryAssociationType instance method is called to determine which file types are used to associate data file objects to this program object.

Parameters

self (WPProgramFile *) – input

The pointer to the object.

Returns

Success indicator:

NULL Error occurred.

Other A pointer to a string containing file types. This string can contain several file title filters separated by a comma.

Example: "Plain Text,C Code"

Remarks

The association type is used to designate this program as an available open view for data file objects which have a type that matches one of the association types which are set. If a data file object matches a type in a program object or program file object, the title of the program object or program file object will appear in the data file object's Open cascade of its pop-up menu. The wpQueryAssociationType method will return the type string set by the last call to the wpSetAssociationType method. The wpQueryAssociationFilter method can be called to determine which file title filters are used to associate data file objects to this program object.

Usage

This method can be called at any time in order to determine which file types are used to associate data file objects to this program object.

How to Override

This method is generally not overridden.

Related Methods

- wpSetAssociationType
- wpSetType
- wpQueryType

wpQueryComputerName – WPPrinter instance method

#define INCL_WINWORKPLACE

```
ULONG wpQueryComputerName (WPPrinter * self, PSZ pszComputerName)
```

The wpQueryComputerName instance method is called to query the name of the computer that the print object exists on.

Parameters

self (WPPrinter *) – input

The pointer to the object to be queried.

pszComputerName (PSZ) – output

The returned computer name of the object queried.

Returns

0 An error occurred.

1 Successful - ComputerName is local (NULL).

2 Successful - ComputerName is on network. See *pszComputerName* for value.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryPrinterName
- wpSetComputerName

wpQueryConfirmations — WPObject instance method

#define INCL_WINWORKPLACE

```
ULONG wpQueryConfirmations  (WPObject * self)
```

The wpQueryConfirmations instance method is called to determine which confirmations are set on this object.

Parameters

self (WPObject *) — input

The pointer to the object.

Returns

Confirmation flags.

NULL	No confirmations set.
CONFIRM_DELETE	Prompt for confirmation for all objects.
CONFIRM_DELETEFOLDER	Prompt for confirmation for just folder objects.

Usage

This method can be called at any time in order to determine the confirmations set on an object.

How to Override

This method is generally not overridden.

Related Methods

- wpDelete
- wpFree
- wpConfirmDelete

wpQueryContent – WPFolder instance method

#define INCL_WINWORKPLACE

WPObject * wpQueryContent (WPFolder * self, WPObject * object, ULONG ulOption)

The wpQueryContent instance method is called to allow the folder to specify its contents.

Parameters

self (WPFolder *) – input

The pointer to this object.

object (WPObject *) – input

A pointer to a workplace object. This field is ignored unless QC_NEXT is specified in ulOption.

ulOption (ULONG) – input

A flag indicating the object to query (QC_FIRST, QC_NEXT,QC_LAST).

Returns

A pointer to the correct item in the folder's content list.

Remarks

This method allows the user to query the folder's content in various ways using the ulOption flag. QC_FIRST returns the first item in the content list, QC_LAST returns the last and QC_NEXT will return the next item after "Object" in the list.

Usage

This method is generally called to look for a specific object or to query the contents in a specific folder.

How to Override

This method is generally not overridden.

wpQueryDefaultHelp — WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpQueryDefaultHelp (WPObject * self, PULONG pHelpPanelId, PSZ pszHelpLibrary)

The wpQueryDefaultHelp instance method is called to allow the object to specify its default help panel.

Parameters

self (WPObject *) — input

The pointer to the object.

pHelpPanelId (PULONG) — output

The pointer to the help panel id.

pszHelpLibrary (PSZ) — output

The pointer to a buffer in which to place the name of help library. This buffer should be the length of CCHMAXPATH bytes.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

The default help panel for this class can be determined by calling the wpclsQueryDefaultHelp method.

Usage

This method can be called at any time in order to determine the default panel for this object.

How to Override

This method is generally not overridden.

Related Methods

- wpDisplayHelp
- wpMenuItemHelpSelected
- wpSetDefaultHelp
- wpclsQueryDefaultHelp

wpQueryDefaultView – WPObject instance method

#define INCL_WINWORKPLACE

```
ULONG wpQueryDefaultView (WPObject * self)
```

The wpQueryDefaultView instance method is called to allow the object to query its current default open view.

Parameters

self (WPObject *) – input

The pointer to this object.

Returns

A flag indicating the default open view.

OPEN_CONTENTS	Open content view.
OPEN_DEFAULT	Open default view (same as double-click).
OPEN_DETAILS	Open details view.
OPEN_HELP	Display HelpPanel.
OPEN_RUNNING	Execute object.
OPEN_SETTINGS	Open settings notebook.
OPEN_TREE	Open tree view.
OPEN_UNKNOWN	Unknown view.
OPEN_USER	Class specific views have a greater value than this.

Remarks

This method returns the default open view for this instance. The default open view is displayed when a user double-clicks on the object or when the user selects *Open* without selecting an item in the open cascade.

Usage

This method can be called at any time in order to determine its default open view.

How to Override

This method is generally not overridden.

Related Methods

- wpclsQueryDefaultView
- wpSetDefaultView

wpQueryDetailsData – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpQueryDetailsData (WPObject * self, PVOID *ppDetailsData, PULONG pcp)

The wpQueryDetailsData instance method is called to allow the object to query its current details data.

Parameters

self (WPObject *) – input

The pointer to this object.

***ppDetailsData** (PVOID) – output

A pointer to detail data information.

pcp (PULONG) – input/output

The length of *ppDetailsData buffer. If ppDetailsData is set to NULL, the actual size of ppDetailsData is returned in pcb.

Returns

Success indicator:

True Successful completion

False Error occurred.

Remarks

All objects which have information to display in *details* view must override this method. *ppDetailsData is a pointer to the beginning of the buffer into which details data should be written. The override should write whatever data it is responsible for and then increment the pointer to the beginning of the area of the next class in the hierarchy (1 byte past the last field for which it is responsible). Note that *ppDetailsData *must* be modified so that the subclasses write in the appropriate place.

The details data returned by a class must match the information returned in *wpclsQueryDetailsInfo*.

Usage

This method can be called at any time in order to determine the current object details.

How to Override

All objects which have information to display in *details* view must override this method.

Note that the parent method must always be called before writing the data and adjusting the pointer.

Related Methods

- wpclsQueryDetails
- wpclsQueryDetailsInfo

Example Code

For example, if writing the following structure:

```
typedef struct _SAMPLE_DETAIL_DATA {
  CDATE   cdate;
  CTIME   ctime;
  PSZ     psz;
} SAMPLE_DETAIL_DATA;
```

the pointer would be modified as follows.

```
((PBYTE) (*ppDetailsData)) += sizeof(SAMPLE_DETAIL_DATA);
```

wpQueryError – WPObject instance method

#define INCL_WINWORKPLACE

```
ULONG wpQueryError  (WPObject * self)
```

The wpQueryError instance method is called to retrieve the error identity of the last error condition.

Parameters

self (WPObject *) – input

The pointer to the object.

Returns

Error identities:

WPERR_PROTECTED_CLASS	Class is protected.
WPERR_INVALID_CLASS	Class is invalid.
WPERR_INVALID_SUPERCLASS	Superclass is invalid.
WPERR_NO_MEMORY	Out of memory.
WPERR_SEMAPHORE_ERROR	Semaphore error.
WPERR_BUFFER_TOO_SMALL	Buffer too small.
WPERR_CLSLOADMOD_FAILED	Unable to load class library module.
WPERR_CLSPROCADDR_FAILED	Unable to find class entry point.
WPERR_OBJWORD_LOCATION	Bad object word location.
WPERR_INVALID_OBJECT	Object pointer is invalid.
WPERR_MEMORY_CLEANUP	Cannot free memory.
WPERR_INVALID_MODULE	Invalid class library module.
WPERR_NO_ERROR	No error conditions have been set.
WPERR_USER	Application-defined errors are above this value.

This value can also include any of the base (ERROR_*) error codes.

Remarks

The wpQueryError method will retrieve the error identity that was set on this object by the last call to the wpSetError method.

Usage

This method can be called at any time in order to determine the identity of the last error that occurred. This method is typically called after calling a method that returned a failure.

How to Override

This method is generally not overridden.

Related Methods

- wpSetError

wpQueryFldrAttr – WPFolder instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpQueryFldrAttr (WPFolder * self, ULONG ulView)

The wpQueryFldrAttr instance method is called to allow the folder to query its current view attributes for the WC_CONTAINER window used in each view window.

Parameters

self (WPFolder *) – input

The pointer to this object.

ulView (ULONG) – input

A flag indicating the view to query.

OPEN_CONTENTS	Open content view.
OPEN_DETAILS	Open details view.
OPEN_HELP	Display HelpPanel.
OPEN_TREE	Open tree view.

Returns

Flag containing current folder view attribute.

These are the CV_* attributes defined by the Container Control Window. See CNRINFO on page A-15 for a detailed description.

Usage

This method can be called at any time in order to determine the view attributes currently set.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryFldrFlags
- wpQueryFldrFont
- wpSetFldrAttr
- wpSetFldrFlags
- wpSetFldrFont

wpQueryFldrDetailsClass – WPFolder instance method

#define INCL_WINWORKPLACE

```
M_WPObject * wpQueryFldrDetailsClass (WPFolder * self)
```

The wpQueryFldrDetailsClass instance method is called to determine which class of details are set for a folder.

Parameters

self (WPFolder *) – input

The pointer to the object.

Returns

Success indicator:

NULL Error occurred.

Other Pointer to the class object for which details are to be displayed.

Remarks

Since folders can contain objects of different classes which can have different details, it is often necessary for the user to specify which class of details to be displayed. The value set by this method is not used until a details view of the folder is opened. The wpSetFldrDetailsClass method can be called to set the current class of details to be displayed.

Usage

This method can be called at any time in order to determine the current class of details to be displayed.

How to Override

This method is generally not overridden.

Related Methods

- wpSetFldrDetailsClass

wpQueryFldrFlags – WPFolder instance method

#define INCL_WINWORKPLACE

```
ULONG wpQueryFldrFlags (WPFolder * self)
```

The wpQueryFldrFlags instance method is called to allow the folder to query its current flags.

Parameters

self (WPFolder *) – input

The pointer to this object.

Returns

Flag indicating the current folder state.

FOI_POPULATEDWITHALL	This flag is set if the folder was populated with all its contents.
FOI_POPULATEDWITHFOLDERS	This flag is set if the folder was populated only with folders that it contains.
FOI_WORKAREA	This flag is set if the user sets the workarea property.

Usage

This method is called at any time in order to determine a folder's flag state.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryFldrAttr
- wpQueryFldrFont
- wpSetFldrAttr
- wpSetFldrFlags
- wpSetFldrFont

wpQueryFldrFont — WPFolder instance method

#define INCL_WINWORKPLACE

```
PSZ wpQueryFldrFont (WPFolder * self, ULONG ulView)
```

The wpQueryFldrFont instance method is called to allow the folder to query its current font.

Parameters

self (WPFolder *) — input

The pointer to this object.

ulView (ULONG) — input

A flag indicating the view to query.

OPEN_CONTENTS	Open content view.
OPEN_DETAILS	Open details view.
OPEN_TREE	Open tree view.

Returns

A pointer to the font string for the specified open view. The font string is in the format of point size followed by a period which is followed by the face name.

For example, "10.Helvetica." (a presentation parameter string).

Remarks

There is only one font for each view. Concurrent views will all have the same font.

Usage

This method can be called at any time in order to determine the current font for a view.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryFldrAttr
- wpQueryFldrFlags
- wpSetFldrAttr
- wpSetFldrFlags
- wpSetFldrFont

wpQueryHandle – WPObject instance method

```
#define INCL_WINWORKPLACE
```

HOBJECT wpQueryHandle (WPObject * somSelf)

The wpQueryHandle instance method returns a persistent object handle for the given object instance.

Parameters

somSelf (WPObject *) – input

The pointer to the instance object.

Returns

The persistent handle for this object.

Remarks

The object handle returned from this method is the same handle that can be used from the WinCreateObject, WinSetObjectData, or WinDestroyObject function calls. The handle is unique on a given machine so the returned object handle can be passed to other processes, or stored for later use (even across IPLs of the system). It is recommended that the *wpQueryHandle* instance method be used sparingly on file based object classes, since the operating system keeps track of the current location of every single file object that has been allocated an object handle. Performance may be adversely affected if object handles were obtained to every single file system object that was ever awakened.

Usage

This method can be called at any time to get a handle that is both persistent across IPLs and completely unique on the given machine.

How to Override

This method should not be overridden

Related Methods

- wpclsQueryObject

wpQueryIcon — WPObject instance method

#define INCL_WINWORKPLACE

```
HPOINTER wpQueryIcon (WPObject * self)
```

The wpQueryIcon instance method is called to allow the object to query its current icon.

Parameters

self (WPObject *) — input

The pointer to this object.

Returns

Success indicator:

NULLHANDLE	Error occurred.
Other	Handle to an icon.

Usage

This method can be called at any time in order to get the handle to the current icon for this object.

How to Override

This method is generally not overridden. The default icon for a class is typically set from an override of wpclsQueryIcon and the instance's icon may be altered with *wpSetIcon*.

Related Methods

- wpclsQueryIcon
- wpQueryIconData
- wpSetIcon
- wpSetIconData

wpQueryIconData – WPObject instance method

```
#define INCL_WINWORKPLACE
```

BOOL wpQueryIconData (WPObject * self, PICONINFO ppiconInfo)

The wpQueryIconData instance method is called to allow the object to query the data to be used for its current icon.

Parameters

self (WPObject *) – input

The pointer to this object.

ppiconInfo (PICONINFO) – output

A pointer to an ICONINFO structure containing an icon specification.

Returns

Success indicator:

True Successful completion

False Error occurred.

Remarks

If pbBuffer is NULL, the size of the icon data is returned in cbBufferSize.

Usage

This method is called at any time in order to query the data for the current icon for this object.

How to Override

This method is generally not overridden.

Related Methods

- wpAddObjectGeneralPage
- wpclsQueryIcon
- wpQueryIcon
- wpSetIcon
- wpSetIconData

wpQueryLogicalDrive – WPDisk instance method

```
#define INCL_WINWORKPLACE

ULONG wpQueryLogicalDrive  (WPDisk * somSelf)
```

The wpQueryLogicalDrive instance method returns the logical drive number that is represented by this disk object.

Parameters

somSelf (WPDisk *) – input

The pointer to the instance object.

Returns

The logical drive identifier.

Remarks

Every instance of the *WPDisk* class that is created in the system must represent a logical drive partition. There should never be more than one disk object per logical drive.

Usage

This method can be called at any time.

How to Override

This method should not be overridden.

wpQueryNextIconPos – WPFolder instance method

#define INCL_WINWORKPLACE

```
PPOINTL wpQueryNextIconPos (WPFolder * self)
```

The wpQueryNextIconPos instance method is called to allow the folder to query the next icon position.

Parameters

self (WPFolder *) – input

The pointer to this object.

Returns

A pointer to the next position at which icons will be inserted.

Remarks

The next icon is the next available *parking space* within the folder's client area.

Usage

This method can be called at any time in order to determine the next position which objects will be inserted in the file.

How to Override

This method is generally not overridden.

Related Methods

- wpSetNextIconPos

wpQueryPaletteHelp — WPPalette instance method

```
#define INCL_WINWORKPLACE
```

ULONG wpQueryPaletteHelp (WPPalette * somSelf)

The wpQueryPaletteHelp instance method returns the help panel ID that is displayed when the help pushbutton is used from an open palette view.

Parameters

somSelf (WPPalette *) — input

The pointer to the instance object.

Returns

The help panel ID within this class's help module as specified by the wpQueryDefaultHelp method for this object instance.

Remarks

This method returns the help panel ID that is visible from the open palette view. That panel should describe what the palette cells represent, how to edit them, and how to apply the cell values to other windows or objects. The palette object can specify class default and instance specific helps for the object in addition to this specialized help which only applies to the palette view.

Usage

This method may be called at any time.

How to Override

All subclasses of *WPPalette* need to override this method to provide help about their open palette view window.

Related Methods

- wpQueryDefaultHelp
- wpclsQueryDefaultHelp
- wpSetDefaultHelp

wpQueryPaletteInfo – WPPalette instance method

#define INCL_WINWORKPLACE

```
BOOL wpQueryPaletteInfo (WPPalette * self, PPALINFO pPalInfo)
```

The wpQueryPaletteInfo instance method is called to determine current information about the palette.

Parameters

self (WPPalette *) – input

The pointer to the object.

pPalInfo (PPALINFO) – input

A pointer to a PALINFO structure.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

Remarks

The palette information can be set by issuing a call to the wpSetPaletteInfo method.

Usage

This method can be called at any time in order to get current information about the palette.

How to Override

This method is generally not overridden.

Related Methods

- wpSetPaletteInfo

wpQueryPrinterName – WPPrinter instance method

#define INCL_WINWORKPLACE

BOOL wpQueryPrinterName (WPPrinter * self, PSZ pszPrinterName)

The wpQueryPrinterName instance method is called to query the name of the printer.

Parameters

self (WPPrinter *) – input

The pointer to the object to be queried.

pszPrinterName (PSZ) – output

The returned printer name of the object queried.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryComputerName
- wpSetPrinterName

wpQueryProgDetails – WPProgram instance method

#define INCL_WINWORKPLACE

BOOL wpQueryProgDetails (WPProgram * self, PPROGDETAILS pProgDetails, PULONG pSize)

The wpQueryProgDetails instance method is called to allow the object to query its program details.

Parameters

self (WPProgram *) – input

The pointer to this object.

pProgDetails (PPROGDETAILS) – input

A pointer to the program details.

pSize (PULONG) – input/output

The size of pProgDetails buffer. If NULL is specified for pProgDetails then the size of the current pProgDetails will be returned in Size.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Usage

This method can be called at any time in order to determine the details on this object.

How to Override

This method is generally not overridden.

Related Methods

- wpSetProgDetails

wpQueryProgDetails – WPProgramFile instance method

#define INCL_WINWORKPLACE

```
BOOL wpQueryProgDetails (WPProgramFile * self, PPROGDETAILS pProgDetails,
                         PULONG pSize)
```

The wpQueryProgDetails instance method is called to allow the object to query its program details.

Parameters

self (WPProgramFile *) – input

The pointer to this object.

pProgDetails (PPROGDETAILS) – input

A pointer to the program details.

pSize (PULONG) – input/output

The size of pProgDetails buffer. If NULL is specified for pProgDetails then the size of the current pProgDetails will be returned in Size.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Usage

This method can be called at any time in order to determine the details on this object.

How to Override

This method is generally not overridden.

Related Methods

- wpSetProgDetails

wpQueryRealName – WPFileSystem instance method

#define INCL_WINWORKPLACE

```
BOOL wpQueryRealName  (WPFileSystem * self, PSZ pszFilename, PULONG pcb,
                       BOOL fQualified)
```

The wpQueryRealName instance method is called to allow the object to query its physical file name.

Parameters

self (WPFileSystem *) – input

The pointer to this object.

pszFilename (PSZ) – output

The pointer to the buffer in which to place the real file name of the object.

pcb (PULONG) – input/output

The size of the file name buffer. If pszFileName is set to NULL, the actual length of the file is returned.

fQualified (BOOL) – input
Success indicator:

TRUE Return the fully qualified file name.

FALSE Return the unqualified file name.

Returns

Success indicator:

TRUE Successful completion.

FALSE An error occurred.

Remarks

This method returns the fully qualified pathname for this object. Generally, the object's real name and an object's title are the same. For file systems which don't support the features of a title, for example, characters, mixed case, and spaces, the title is stored in the .LONGNAME extended attribute and then the title and real name may differ. The real name of the file object can be used with any of the DOSXXX functions which act on file names.

Usage

This method can be called at any time in order to determine the physical file name for this object.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryType
- wpSetRealName
- wpSetTitle

wpQueryRootFolder — WPDisk instance method

#define INCL_WINWORKPLACE

```
WPRootFolder * wpQueryRootFolder (WPDisk * somSelf)
```

The wpQueryRootFolder instance method returns the root folder object for the logical drive that is represented by the *WPDisk* object.

Parameters

somSelf (WPDisk *) — input

The pointer to the instance object.

Returns

The root folder object. Null if an error occurred.

Remarks

Every instance of the *WPDisk* class that is created in the system points to a root folder, the root directory of the logical device that the disk object represents.

Usage

This method can be called at any time.

How to Override

This method should not be overridden.

wpQueryShadowedObject – WPShadow instance method

```
#define INCL_WINWORKPLACE
```

```
WPObject * wpQueryShadowedObject (WPShadow * self, BOOL fLock)
```

The wpQueryShadowedObject instance method is called to allow the shadow object to query the object with which it is currently linked.

Parameters

self (WPShadow *) – input

The pointer to this object.

fLock (BOOL) – input

The lock object flag.

If this flag is false, the newly created object will be made dormant whenever the object and the folder containing the object are closed. If this flag is true, the new flag will remain awake until the caller issues the wpUnlockObject method on it.

Returns

Success indicator:

NULL Error occurred.

Other A pointer to the object with which this shadow is linked.

Usage

This method is called at any time in order to determine the object with which this shadow is currently linked.

How to Override

This method is generally not overridden.

Related Methods

- wpCreateShadowObject

wpQueryStyle – WPObject instance method

#define INCL_WINWORKPLACE

ULONG wpQueryStyle (WPObject * self)

The wpQueryStyle instance method allows the object to query its current class style.

Parameters

self (WPObject *) – input

The pointer to this object.

Returns

The flags which are *ored* together to indicate the object's style.

OBJSTYLE_NOCOPY	Cannot be copied.
OBJSTYLE_NODELETE	Cannot be deleted.
OBJSTYLE_NODRAG	Cannot be dragged.
OBJSTYLE_NOSHADOW	Cannot have shadow created.
OBJSTYLE_NOMOVE	Cannot move.
OBJSTYLE_NOPRINT	Cannot be printed.
OBJSTYLE_NOTDEFAULTICON	Destroy icon when object goes dormant
OBJSTYLE_TEMPLATE	This object is a template.
OBJSTYLE_NOTVISIBLE	The object is hidden.
OBJSTYLE_NORENAME	Cannot be renamed.

Usage

This method can be called at any time in order to determine the current style for an object class. To determine the default style for an object class, the wpclsQueryStyle method should be called.

How to Override

This method is generally not overridden.

Related Methods

- wpclsQueryStyle
- wpSetStyle

wpQueryTitle – WPObject instance method

```
#define INCL_WINWORKPLACE

PSZ wpQueryTitle (WPObject * self)
```

The wpQueryTitle instance method is called to allow the object to query its current title.

Parameters

self (WPObject *) – input

The pointer to this object.

Returns

A pointer to the object's title.

Remarks

The object's title may get altered by the user at any time. Objects should always use this method to access the current title and never store the string pointer that is returned.

Usage

This method can be called at any time in order to determine the current title of an object. To determine the default title for an object's class, the wpclsQueryTitle method should be called.

How to Override

This method is generally not overridden.

Related Methods

- wpclsQueryTitle
- wpSetTitle

wpQueryType — WPFileSystem instance method

#define INCL_WINWORKPLACE

```
PSZ wpQueryType  (WPFileSystem * self)
```

The wpQueryType instance method is called to allow the object to query the type of its file.

Parameters

self (WPFileSystem *) — input

The pointer to this object.

Returns

The pointer to a buffer containing file type. This string can contain a list of types delineated by a line feed character. For example, "Plain Text\nC Code."

Remarks

This method returns the type of a file-system-based object. The type of a file is designated by its .TYPE extended attribute value.

Usage

This method can be called at any time in order to determine the type of the file object.

How to Override

This method is generally not overridden.

Related Methods

- wpAddFileTypePage
- wpSetTitle

wpRedrawCell – WPPalette instance method

#define INCL_WINWORKPLACE

```
BOOL wpRedrawCell (WPPalette * somSelf, PCELL pCell)
```

The wpRedrawCell instance method forces a palette object to repaint the specified cell area.

Parameters

somSelf (WPPalette *) – input

The pointer to the instance object.

pCell (PCELL) – input

The pointer to the cell within the palette that needs repainting.

Returns

TRUE The method call was successful.

FALSE The method call was unsuccessful.

Remarks

This method is used as cell values within the palette are altered. For example, when the color selector dialog is changing the color of a palette cell in the color palette this method is invoked to refresh the color back in the open views of the color palette object.

Usage

This method may be called at any time to force the palette object to repaint the specified cell in all of its currently open views.

How to Override

Overriding this method is not recommended

Related Methods

- wpPaintCell

wpRefresh — WPFileSystem instance method

#define INCL_WINWORKPLACE

```
BOOL wpRefresh (WPFileSystem * self, ULONG ulView, PVOID pReserved)
```

The wpRefresh instance method is called to refresh the contents of a folder.

Parameters

self (WPFileSystem *) — input

The pointer to the object.

ulView (ULONG) — input

A flag indicating which view to refresh.

OPEN_CONTENT Refresh content (icon) view.

OPEN_DETAILS Refresh details view.

pReserved (PVOID) — input

Reserved value must be 0.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

Usage

This method can be called at any time to refresh the contents of a folder.

How to Override

This method is generally not overridden.

Related Methods

- wpPopulate

wpRegisterView – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpRegisterView (WPObject * self, HWND hwndFrame, PSZ pszViewTitle)
```

The wpRegisterView instance method is called to allow the object to register a new open view.

Parameters

self (WPObject *) – input

The pointer to this object.

hwndFrame (HWND) – input

A handle to the frame window containing the new view.

pszViewTitle (PSZ) – input

The pointer to a string containing name of view.

Returns

Success indicator:

True Successful completion

False Error occurred.

Remarks

Registering a view will set the object title as the title of the frame window and add a view title as the current view in window list and title bar. In-use emphasis is managed by the *wpAddToObjectUseList* method.

Usage

This method is generally called during the processing of the wpOpen method in order to register a new view with the object.

How to Override

This method is generally not overridden.

Related Methods

- wpAddToObjUseList

wpReleaseJob — WPJob instance method

#define INCL_WINWORKPLACE

```
BOOL wpReleaseJob  (WPJob * self)
```

The wpReleaseJob instance method is called to release a job object.

Parameters

self (WPJob *) — input

The pointer to the object to be released.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method is generally not overridden.

Related Methods

- wpHoldJob

wpReleasePrinter – WPPrinter instance method

#define INCL_WINWORKPLACE

```
BOOL wpReleasePrinter (WPPrinter * self)
```

The wpReleasePrinter instance method is called to release a print object.

Parameters

self (WPPrinter *) – input

The pointer to the object to be released.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method is generally not overridden.

Related Methods

- wpHoldPrinter

wpRender – WPObject instance method

#define INCL_WINWORKPLACE

MRESULT wpRender (WPObject * self, PDRAGTRANSFER ppdxfer)

The wpRender instance method is called to request a drag or drop rendering format from the object.

Parameters

self (WPObject *) – input

The pointer to the object.

ppdxfer (PDRAGTRANSFER) – input

A pointer to a DRAGTRANSFER structure.

Returns

Refer to the "DM_RENDER" on page 29-10 message for a description of the return value.

Remarks

The wpRender method is called when the object receives a DM_RENDER message. Refer to the DM_RENDER message for more information.

Usage

This method is generally called only by the system as the folder containing the object processed by the DM_RENDER message.

How to Override

This method should be overridden to return a class-specific rendering mechanism and format.

Related Methods

- wpDragOver
- wpDrop
- wpFormatDragItem
- wpRenderComplete
- wpEndConversation

wpRenderComplete – WPObject instance method

#define INCL_WINWORKPLACE

MRESULT wpRenderComplete (WPObject * self, PDRAGTRANSFER ppdxfer, ULONG ulResult)

The wpRenderComplete instance method is called to notify the object that the drag or drop rendering request is complete.

Parameters

self (WPObject *) – input

The pointer to the object.

ppdxfer (PDRAGTRANSFER) – input

A pointer to a DRAGTRANSFER structure.

ulResult (ULONG) – input

The flag indicating whether the operation was performed successfully. Refer to "DM_RENDERCOMPLETE" on page 29-11 for more information about this parameter.

Returns

Refer to the DM_RENDERCOMPLETE message for a description of the return value.

Remarks

The wpRenderComplete method is called when the object receives a DM_RENDERCOMPLETE message. Refer to the DM_RENDERCOMPLETE message for more information.

Usage

This method is generally called only by the system as the folder containing the object processed by the DM_RENDERCOMPLETE message.

How to Override

This method should be overridden if the class has a special rendering mechanism and format.

Related Methods

- wpDragOver
- wpDrop
- wpFormatDragItem
- wpRender
- wpEndConversation

wpRestore — WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpRestore  (WPObject * self)
```

The wpRestore instance method is called to allow the object to restore its views from the hidden or minimized states.

Parameters

self (WPObject *) — input

The pointer to this object.

Returns

Success indicator:

True Successful completion

False Error occurred.

Remarks

This method is the inverse of the wpHide method.

Usage

This method can be called at any time in order to restore all views of this object from the hidden or minimized state.

How to Override

This method is generally not overridden.

Related Methods

- wpHide

wpRestoreData – WPObject instance method

```
#define INCL_WINWORKPLACE
```

```
BOOL wpRestoreData (WPObject * self, PSZ pszClass, ULONG ulKey, PBYTE pbValue,
                    PULONG cbValue)
```

The wpRestoreData instance method is called to allow the object to restore its binary instance data.

Parameters

self (WPObject *) – input

The pointer to this object.

pszClass (PSZ) – input

A pointer to a zero terminated string which contains any unique string. The class name is recommended but not enforced.

ulKey (ULONG) – input

A class defined identifier that correlates to a particular instance data variable.

pbValue (PBYTE) – input/output

The address of the data to be restored.

cbValue (PULONG) – input/output

The size of the data block to be restored. If pbValue is NULL, the actual size will be returned in cbValue.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This method restores data that was saved by a call to wpSaveData.

Usage

This method can be called only during the processing of the wpRestoreState method.

How to Override

This method is generally not overridden.

Related Methods

- wpRestoreLong
- wpRestoreState
- wpRestoreString
- wpSaveData
- wpSaveImmediate
- wpSaveLong
- wpSaveState
- wpSaveString

wpRestoreLong – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpRestoreLong (WPObject * self, PSZ pszClass, ULONG ulKey, PULONG pValue)

The wpRestoreLong instance method is called to allow the object to restore a 32-bit instance data value.

Parameters

self (WPObject *) – input

The pointer to this object.

pszClass (PSZ) – input

A pointer to a zero terminated string which contains any unique string. The class name is recommended but not enforced.

ulKey (ULONG) – input

A class defined identifier that correlates to a particular instance data variable.

pValue (PULONG) – output

The address of the long value.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This method restores a 32-bit data value that was saved by a call to wpSaveLong.

Usage

This method can be called only during the processing of the wpRestoreState method.

How to Override

This method is generally not overridden.

Related Methods

- wpRestoreData
- wpRestoreState
- wpRestoreString
- wpSaveData
- wpSaveImmediate
- wpSaveLong
- wpSaveState
- wpSaveString

wpRestoreState – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpRestoreState (WPObject * self, ULONG ulReserved)

This method restores the state of the object which was saved during the processing of the wpSaveState method.

Parameters

self (WPObject *) – input

The pointer to this object.

ulReserved (ULONG) – input

The reserved value must be 0.

Returns

Success indicator:

True Successful completion

False Error occurred.

Remarks

This method restores the state of the object which was saved during the processing of the wpSaveState method.

Usage

This method is generally called only by the system while it is processing the wpInitData method.

How to Override

This method should be overridden by all classes which provide settings that can be saved. An override of the wpSaveState method is a prerequisite if persistent instance data is desired.

Override processing of this method typically includes a series of calls to any combination of the restore state methods:

- wpRestoreData
- wpRestoreLong
- wpRestoreString

Related Methods

- wpRestoreData
- wpRestoreLong
- wpRestoreString
- wpSaveData
- wpSaveImmediate
- wpSaveLong
- wpSaveState
- wpSaveString

wpRestoreString – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpRestoreString (WPObject * self, PSZ pszClass, ULONG ulKey, PSZ pszValue,
                      PULONG pulValue)
```

The wpRestoreString instance method is called to allow the object to restore an ASCIIZ instance data string.

Parameters

self (WPObject *) – input

The pointer to this object.

pszClass (PSZ) – input

A pointer to a zero terminated string which contains any unique string. The class name is recommended but not enforced.

ulKey (ULONG) – input

A class defined identifier that correlates to a particular instance data variable.

pszValue (PSZ) – input/output

The address of the string to be restored.

pulValue (PULONG) – input/output

The size of the string to be restored. If pszValue is NULL, the actual size will be returned in pulValue.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This method restores an ASCIIZ string that was saved by a call to wpSaveString.

Usage

This method can be called only during the processing of the wpRestoreState method.

How to Override

This method is generally not overridden.

Related Methods

- wpRestoreData
- wpRestoreLong
- wpRestoreState
- wpSaveData
- wpSaveImmediate
- wpSaveLong
- wpSaveState
- wpSaveString

wpSaveData – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpSaveData (WPObject * self, PSZ pszClass, ULONG ulKey, PBYTE pbValue,
                 ULONG ulValue)
```

The wpSaveData instance method is called to allow an object to save its binary instance data.

Parameters

self (WPObject *) – input

The pointer to this object.

pszClass (PSZ) – input

A pointer to a zero terminated string which contains any unique string. The class name is recommended but not enforced.

ulKey (ULONG) – input

A class defined identifier that correlates to a particular instance data variable.

pbValue (PBYTE) – input

The address of the block of data to be stored.

ulValue (ULONG) – input

The size of the block of data to be stored.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

The saved data can be restored by issuing a call to wpRestoreData.

Usage

This method can be called only during the processing of the wpSaveState method.

How to Override

This method is generally not overridden.

Related Methods

- wpRestoreData
- wpRestoreLong
- wpRestoreState
- wpRestoreString
- wpSaveImmediate
- wpSaveLong
- wpSaveState
- wpSaveString

wpSaveDeferred – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpSaveDeferred (WPPalette * somSelf)

The wpSaveDeferred instance method tells the system to make the object to save itself asynchronously.

Parameters

somSelf (WPPalette *) – input

The pointer to the instance object.

Returns

TRUE The save request was accepted.

FALSE The save request was not accepted.

Remarks

The system maintains a list of objects that currently need to be saved and will periodically ask those objects to save their state data to persistent storage using the *wpSaveImmediate* method. The *wpSaveDeferred* method should always be used in preference to the *wpSaveImmediate* method because of performance. The only exception is if the state data was changed in some critical way. For example, if the object wanted to save a new password, it would use *wpSaveImmediate* instead of *wpSaveDeferred* to guarantee that the password has been saved before continuing.

Usage

This method should be called by all object classes whenever state data is altered. The workplace classes invoke this method each time a wpSetXXX method is used.

How to Override

This method should not be overridden

Related Methods

- wpSaveImmediate

wpSaveImmediate – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpSaveImmediate (WPObject * self)
```

The wpSaveImmediate instance method is called to allow the object to save its current state.

Parameters

self (WPObject *) – input
The pointer to this object.

Returns

Success indicator:

True Successful completion

False Error occurred.

Remarks

This method will cause the wpSaveState method to be called.

Usage

This method is called automatically for all objects when they are made dormant or when the system is shut down. However, an object can call this method on itself at any time when a critical instance variable is changed.

How to Override

This method is generally not overridden.

Related Methods

- wpRestoreData
- wpRestoreLong
- wpRestoreState
- wpRestoreString
- wpSaveData
- wpSaveLong
- wpSaveState
- wpSaveString

wpSaveLong – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpSaveLong (WPObject * self, PSZ pszClass, ULONG ulKeyle, ULONG ulValue)
```

The wpSaveLong instance method is called to allow the object to save a 32-bit instance data value.

Parameters

self (WPObject *) – input

The pointer to this object.

pszClass (PSZ) – input

A pointer to a zero terminated string which contains any unique string. The class name is recommended but not enforced.

ulKeyle (ULONG) – input

A class defined identifier that correlates to a particular instance data variable.

ulValue (ULONG) – input

The value (ULONG) to be stored.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

The saved 32-bit data value can be restored by issuing a call to wpRestoreLong.

Usage

This method can be called only during the processing of the wpSaveState method.

How to Override

This method is generally not overridden.

Related Methods

- wpRestoreData
- wpRestoreLong
- wpRestoreState
- wpRestoreString
- wpSaveData
- wpSaveImmediate
- wpSaveState
- wpSaveString

wpSaveState –
WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpSaveState (WPObject * self)
```

The wpSaveState instance method is called to allow the object to save its state.

Parameters

self (WPObject *) – input

The pointer to this object.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

The saved state of the object is restored during the processing of the wpRestoreState method.

Usage

This method is generally called by the system while it is processing either the wpClose or wpSaveImmediate methods. If an immediate save is required, the wpSaveImmediate method can be called.

How to Override

This method should be overridden by all classes which provide savable settings. An override of the wpRestoreState method is a prerequisite.

Override processing of this method typically includes a series of calls to any combination of the save state methods:

- wpSaveData
- wpSaveLong
- wpSaveString

Related Methods

- wpRestoreData
- wpRestoreLong
- wpRestoreState
- wpRestoreString
- wpSaveData
- wpSaveImmediate
- wpSaveLong
- wpSaveString

wpSaveString – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpSaveString (WPObject * self, PSZ pszClass, ULONG ulKey, PSZ pszValue)

The wpSaveString instance method is called to allow the object to save an ASCIIZ instance data string.

Parameters

self (WPObject *) – input

The pointer to this object.

pszClass (PSZ) – input

A pointer to a zero terminated string which contains any unique string. The class name is recommended but not enforced.

ulKey (ULONG) – input

A class defined identifier that correlates to a particular instance data variable.

pszValue (PSZ) – input

String to be stored.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

The saved ASCIIZ string can be restored by issuing a call to wpRestoreString.

Usage

This method can be called only during the processing of the wpSaveState method.

How to Override

This method is generally not overridden.

Related Methods

- wpRestoreData
- wpRestoreLong
- wpRestoreState
- wpRestoreString
- wpSaveData
- wpSaveImmediate
- wpSaveLong
- wpSaveState

wpScanSetupString — WPObject instance method

```
#define INCL_WINWORKPLACE
```

```
BOOL wpScanSetupString (WPObject * self, PSZ pszSetupString, PSZ pszKey, PSZ pszValue,
                        PULONG pcbValue)
```

The wpScanSetupString instance method is called to allow an object to parse the setup string that is passed when the object is created.

Parameters

self (WPObject *) — input

The pointer to this object.

pszSetupString (PSZ) — input

The class specific setup parameters for an object.

pszKey (PSZ) — input

The key to scan for.

pszValue (PSZ) — input

The buffer for the value.

pcbValue (PULONG) — input/output

If pszValue is null, the length of the string plus one is returned in pcbValue.

Returns

Success indicator:

True Successful completion

False Error occurred.

Remarks

If a comma or semicolon is needed in the setup string, the escape character ^ can be used.

Usage

This method is generally called from within an override of the wpSetup method.

How to Override

This method is generally not overridden.

Related Methods

- wpclsNew
- wpSetup
- WinCreateObject

Example Code

If *NOMOVE=YES* is in the setup string, make the object non-moveable.

```
  UCHAR   szValue[255];
  ULONG   cbValue = 255;

if ( wpScanSetupString ( self, pszSetup,
                         "NOMOVE",
                         szValue,
                         &cbValue ) ) {

   if ( !strcmpi ( szValue, "YES" ) ) {
     _wpSetStyle (self, _wpQueryStyle (self) | OBJSTYLE_NOMOVE);
   }
}
```

wpSetAssociationFilter — WPProgram instance method

```
#define INCL_WINWORKPLACE
```

BOOL wpSetAssociationFilter (WPProgram * self, PSZ pszFilter)

The wpSetAssociationFilter instance method is called to set an association of the program object to a data file object based on a file title filter.

Parameters

self (WPProgram *) — input

The pointer to the object.

pszFilter (PSZ) — input

A pointer to a string containing file title filters to associate. This string can contain several file title filters separated by a comma. Example: pszFilter = "*.TXT, *.DOC"

Returns

Success indicator:

TRUE Successful completion.

FALSE Error occurred.

Remarks

The association filter is used to designate this program as an available open view for data file objects which have a title that matches one of the association filters which are set. If a data file object matches a filter in a program object or program file object, the title of the program object or program file object will appear in the data file object's Open cascade of its pop-up menu. A call to the wpSetAssociationFilter method will cause the existing association filters for this object to be replaced. To determine the existing association filters that are set on this object, a call to the wpQueryAssociationFilter method can be made. The wpSetAssociationType method can be called to set an association based on the type of data file object.

Usage

This method can be called at any time in order to set an association of the program object to a data file object based on a file title filter.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryAssociationType
- wpSetAssociationFilter
- wpSetAssociationType

wpSetAssociationFilter — WPProgramFile instance method

#define INCL_WINWORKPLACE

BOOL wpSetAssociationFilter (WPProgramFile * self, PSZ pszFilter)

The wpSetAssociationFilter instance method is called to set an association of the program object to a data file object based on a file title filter.

Parameters

self (WPProgramFile *) — input

The pointer to the object.

pszFilter (PSZ) — input

A pointer to a string containing file title filters to associate. This string can contain several file title filters separated by a comma.

Example: pszFilter = "*.TXT, *.DOC"

Returns

Success indicator:

TRUE Successful completion.

FALSE Error occurred.

Remarks

The association filter is used to designate this program as an available open view for data file objects which have a title that matches one of the association filters which are set. If a data file object matches a filter in a program object or program file object, the title of the program object or program file object will appear in the data file object's Open cascade of its pop-up menu. A call to the wpSetAssociationFilter method will cause the existing association filters for this object to be replaced. To determine the existing association filters that are set on this object, a call to the wpQueryAssociationFilter method can be made. The wpSetAssociationType method can be called to set an association based on the type of data file object.

Usage

This method can be called at any time in order to set an association of the program object to a data file object based on a file title filter.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryAssociationType
- wpSetAssociationType

wpSetAssociationType – WPProgram instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetAssociationType (WPProgram * self, PSZ pszType)
```

The wpSetAssociationType instance method is called to set an association of the program object to a data file object based on a file type.

Parameters

self (WPProgram *) – input

The pointer to the object.

pszType (PSZ) – input

A pointer to a string containing file types to associate. This string can contain several file types separated by a comma. Example: pszType = "in Text,C Code"

Returns

Success indicator:

TRUE Successful completion.

FALSE Error occurred.

Remarks

The association type is used to designate this program as an available open view for data file objects which have a type that matches one of the association types which are set. If a data file object matches a type in a program object or program file object, the title of the program object or program file object will appear in the data file object's Open cascade of its popup menu. A call to the wpSetAssociationType method will cause the existing association types for this object to be replaced. To determine the existing association type(s) that are set on this object, a call to the wpQueryAssociationType method can be made. The wpSetAssociationFilter method can be called to set an association based on the title of data file object.

Usage

This method can be called at any time in order to set an association of the program object to a data file object based on a file type.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryAssociationType
- wpSetAssociationFilter
- wpSetAssociationType

wpSetAssociationType – WPProgramFile instance method

#define INCL_WINWORKPLACE

BOOL wpSetAssociationType (WPProgramFile * self, PSZ pszType)

The wpSetAssociationType instance method is called to set an association of the program object to a data file object based on a file type.

Parameters

self (WPProgramFile *) – input

The pointer to the object.

pszType (PSZ) – input

A pointer to a string containing file types to associate. This string can contain several file types separated by a comma.

Example: pszType = "Plain Text,C Code"

Returns

Success indicator:

TRUE Successful completion.

FALSE Error occurred.

Remarks

The association type is used to designate this program as an available open view for data file objects which have a type that matches one of the association types which are set. If a data file object matches a type in a program object or program file object, the title of the program object or program file object will appear in the data file object's Open cascade of its popup menu. A call to the wpSetAssociationType method will cause the existing association types for this object to be replaced. To determine the existing association types that are set on this object, a call to the wpQueryAssociationType method can be made. The wpSetAssociationFilter method can be called to set an association based on the title of data file object.

Usage

This method can be called at any time in order to set an association of the program object to a data file object based on a file type.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryAssociationType
- wpSetAssociationFilter

wpSetComputerName – WPPrinter instance method

#define INCL_WINWORKPLACE

BOOL wpSetComputerName (WPPrinter * self, PSZ pszComputerName)

The wpSetComputerName instance method is called to set the name of the computer that the printer exists on.

Parameters

self (WPPrinter *) – input

The pointer to the object to set the computer name.

pszComputerName (PSZ) – input

The computer name to be set.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryComputerName
- wpSetPrinterName

wpSetDefaultHelp — WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetDefaultHelp (WPObject * self, ULONG ulPanelID, PSZ pszHelpLibrary)
```

The wpSetDefaultHelp instance method is called to set the default help panel for the object.

Parameters

self (WPObject *) — input

The pointer to the object.

ulPanelID (ULONG) — input

The help panel identity.

pszHelpLibrary (PSZ) — input

The pointer to name of Help Library. A NULL value implies the default should be used.

Returns

Success indicator:

TRUE Successful completion.

FALSE Error occurred.

Remarks

The default help panel for this class can be determined by calling the wpclsQueryDefaultHelp method.

Usage

This method can be called at any time in order to set the default help panel for this object.

How to Override

This method is generally not overridden.

Related Methods

- wpDisplayHelp
- wpMenuItemHelpSelected
- wpQueryDefaultHelp
- wpclsQueryDefaultHelp

wpSetDefaultPrinter — WPPrinter instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetDefaultPrinter (WPPrinter * self)
```

The wpSetDefaultPrinter instance method is called to set a default print object.

Parameters

self (WPPrinter *) — input

The pointer to the object to be the default.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method is generally not overridden.

wpSetDefaultView – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpSetDefaultView (WPObject * self, ULONG ulView)

The wpSetDefaultView instance method is called to allow an object to change its current default open view.

Parameters

self (WPObject *) – input

The pointer to this object.

ulView (ULONG) – input

Specifies which view to open.

OPEN_CONTENTS	Open content view.
OPEN_DEFAULT	Open default view (same as double-click).
OPEN_DETAILS	Open details view.
OPEN_HELP	Display HelpPanel.
OPEN_RUNNING	Execute object.
OPEN_SETTINGS	Open settings notebook.
OPEN_TREE	Open tree view.
OPEN_USER	Class specific views have a greater value than this.

Returns

Success indicator:

True	Successful completion
False	Error occurred.

Remarks

The default open view for this class can be determined by calling the wpclsQueryDefaultView method.

Usage

This method can be called at any time in order to set the default open view for this object.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryDefaultView
- wpclsQueryDefaultView

wpSetError – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetError (WPObject * self, ULONG ulErrorID)
```

The wpSetError instance method is called to identify an error condition.

Parameters

self (WPObject *) – input

The pointer to the object.

ulErrorID (ULONG) – input

An error identity. Refer to the wpQueryError method for information about this value.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

Remarks

The wpSetError method will set the last error on an object. The error identity is retrievable by issuing a call to the wpQueryError method.

Usage

This method can be called at any time in order to identify an error condition. This method is typically called prior to returning unsuccessfully from a method.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryError

wpSetFldrAttr — WPFolder instance method

#define INCL_WINWORKPLACE

BOOL wpSetFldrAttr (WPFolder * self, ULONG ulAttr, ULONG ulView)

The wpSetFldrAttr instance method is called to allow the folder to change its current view attributes.

Parameters

self (WPFolder *) — input

The pointer to this object.

ulAttr (ULONG) — input

A flag indicating the object's attributes. These are the CV_* attributes defined by the Container Control Window. See CNRINFO on page A-15 for detailed description.

ulView (ULONG) — input

A flag indicating an object's open view.

OPEN_CONTENTS	Open content view.
OPEN_DETAILS	Open details view.
OPEN_TREE	Open tree view.

Returns

Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The flags should be valid for the specified view. For example, CV_TREE should not be specified for details view.

Usage

This method can be called at any time in order to set the view attributes.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryFldrAttr
- wpQueryFldrFlags
- wpQueryFldrFont
- wpSetFldrFlags
- wpSetFldrFont

wpSetFldrDetailsClass – WPFolder instance method

```
#define INCL_WINWORKPLACE
```

BOOL wpSetFldrDetailsClass (WPFolder * self, M_WPObject * Class)

The wpSetFldrDetailsClass instance method is called to set the class for which details in the folder will be displayed.

Parameters

self (WPFolder *) – input

The pointer to the object.

Class (M_WPObject *) – input

The pointer to the class object for which details are to be displayed.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

Remarks

Since folders can contain objects of different classes which can have different details, it is often necessary for the user to specify which class of details to be displayed. The value set by this method is not used until a details view of the folder is opened. The wpQueryFldrDetailsClass method can be called to determine the class of details currently set. All column visibility states are reset by this method.

Usage

This method can be called at any time in order to set the current class of details to be displayed.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryFldrDetailsClass

wpSetFldrFlags — WPFolder instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetFldrFlags (WPFolder * self, ULONG ulFlags)
```

The wpSetFldrFlags instance method is called to allow the folder to change its current flags.

Parameters

self (WPFolder *) — input

The pointer to this object.

ulFlags (ULONG) — input

The folder flags to be set

FOI_POPULATEDWITHALL	This flag is set if the folder was populated with all its contents.
FOI_POPULATEDWITHFOLDERS	This flag is set if the folder was populated only with folders that it contains.
FOI_WORKAREA	This flag is set if the user sets the workarea property.

Returns

Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Usage

This method can be called at any time in order to set a folder's flag.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryFldrAttr
- wpQueryFldrFlags
- wpQueryFldrFont
- wpSetFldrAttr
- wpSetFldrFont

wpSetFldrFont – WPFolder instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetFldrFont (WPFolder * self, PSZ pszFont, ULONG ulView)
```

The wpSetFldrFont instance method is called to allow the folder to change its current font.

Parameters

self (WPFolder *) – input

The pointer to this object.

pszFont (PSZ) – input

A pointer to a buffer containing the font name.

If pszFont is NULL, the font presentation parameter will be set back to the default font. If a pointer to a NULL string is passed, the font presentation parameter will be set to the currently defined font.

ulView (ULONG) – input

A flag indicating an object's open view.

OPEN_CONTENTS	Open content view.
OPEN_DETAILS	Open details view.
OPEN_TREE	Open tree view.

Returns

Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The font name should be a valid presentation parameter string.

Usage

This method can be called at any time in order to change the current font for a view.

How to Override

This method is generally not overridden.

wpSetIcon – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetIcon (WPObject * self, HPOINTER hptrNewIcon)
```

The wpSetIcon instance method is called to allow the object to set its current icon.

Parameters

self (WPObject *) – input

The pointer to this object.

hptrNewIcon (HPOINTER) – input

A pointer to the object handle.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

If the OBJSTYLE_NOTDEFAULTICON style is currently set for the object, the object's icon will be destroyed if the object is destroyed or made dormant.

Usage

This method can be called at any time in order to change the visible icon for this object. To permanently change the icon, the wpSetIconData method should be called.

How to Override

This method is generally not overridden.

Related Methods

- wpclsQueryIcon
- wpQueryIcon
- wpQueryIconData
- wpSetIconData

wpSetIconData – WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpSetIconData (WPObject * self, PICONINFO ppiconInfo)

The wpSetIconData instance method is called to allow the object to permanently set its current icon.

Parameters

self (WPObject *) – input

The pointer to this object.

ppiconInfo (PICONINFO) – input

A pointer to an ICONINFO structure containing an icon specification.

Returns

Success indicator:

True Successful completion

False Error occurred.

Usage

This method can be called at any time in order to permanently change the icon for this object. To temporarily change or refresh the icon for this object, the wpSetIcon method should be called.

How to Override

This method is generally not overridden.

Related Methods

- wpclsQueryIcon
- wpQueryIcon
- wpQueryIconData
- wpSetIcon

wpSetNextIconPos – WPFolder instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetNextIconPos (WPFolder * self, PPOINTL pptl)
```

The wpSetNextIconPos instance method is called to allow the folder to change the next icon position.

Parameters

self (WPFolder *) – input

The pointer to this object.

pptl (PPOINTL) – input

The position within the folder.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

The next icon position is typically set during a drag or drop operation in order to ensure that the items dropped into a folder are located where the user wanted them.

Usage

This method can be called at any time in order to set the next icon position at which objects will be inserted in the folder. If this method is used, the previous next position should be queried and restored afterwards.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryNextIconPos

wpSetPaletteInfo – WPPalette instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetPaletteInfo (WPPalette * self, PPALINFO pPalInfo)
```

The wpSetPaletteInfo instance method is called to set palette information.

Parameters

self (WPPalette *) – input

The pointer to the object.

pPalInfo (PPALINFO) – input

A pointer to a PALINFO structure.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

Remarks

The palette information can be retrieved by issuing a call to wpQueryPaletteInfo.

Usage

This method can be called at any time in order to set the palette information.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryPaletteInfo

wpSetPrinterName – WPPrinter instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetPrinterName  (WPPrinter * self, PSZ pszPrinterName)
```

The wpSetPrinterName instance method is called to set the name of the printer.

Parameters

self (WPPrinter *) – input

The pointer to the object to set the printer name.

pszPrinterName (PSZ) – input

The printer name to be set.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryPrinterName
- wpSetComputerName

wpSetProgDetails – WPProgram instance method

```
#define INCL_WINWORKPLACE
```

```
BOOL wpSetProgDetails (WPProgram * self, PPROGDETAILS pProgDetails)
```

The wpSetProgDetails instance method is called to allow the object to change its program details.

Parameters

self (WPProgram *) – input

The pointer to this object.

pProgDetails (PPROGDETAILS) – input

The pointer to the program details.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Usage

This method can be called at any time in order to set the details for this object.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryProgDetails

wpSetProgDetails — WPProgramFile instance method

#define INCL_WINWORKPLACE

BOOL wpSetProgDetails (WPProgramFile * self, PPROGDETAILS pProgDetails)

The wpSetProgDetails instance method is called to allow the object to change its program details.

Parameters

self (WPProgramFile *) — input

The pointer to this object.

pProgDetails (PPROGDETAILS) — input

The pointer to the program details.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Usage

This method can be called at any time in order to set the details for this object.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryProgDetails

wpSetRealName – WPFileSystem instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetRealName (WPObject * self, PSZ pszFilename)
```

The wpSetRealName instance method will set the physical name of a file system object.

Parameters

self (WPObject *) – input

The pointer to the object.

pszFilename (PSZ) – input

A pointer to a new filename. This file can not be fully qualified.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

Remarks

In most cases, the file system object's real name and title are identical. When a title is set that the file system containing the object cannot handle, the real name is different. In this situation, real name is set to be a truncated title. In the case where the real name and the title are different, the title is stored in the file's ".LONGNAME" extended attribute.

Usage

This method can be called at any time in order to set the physical name for a file system object.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryRealName

wpSetShadowTitle — WPShadow instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetShadowTitle (WPShadow * self, PSZ pszTitle)
```

The wpSetShadowTitle instance method is called to set the title on the shadow without affecting the title on the object it is shadowing.

Parameters

self (WPShadow *) — input

The pointer to the object.

pszTitle (PSZ) — input

A pointer to a title.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

Usage

This method can be called at any time in order to set a title on a shadow object without affecting the title on the object it is shadowing.

How to Override

This method is generally not overridden.

Related Methods

- wpSetTitle

wpSetStyle –
WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetStyle (WPObject * self, ULONG ulNewStyle)
```

The wpSetStyle instance method is called to allow an object to set its current object style.

Parameters

self (WPObject *) – input

The pointer to this object.

ulNewStyle (ULONG) – input

OBJSTYLE_NOCOPY	Cannot be copied.
OBJSTYLE_NODELETE	Cannot be deleted.
OBJSTYLE_NODRAG	Cannot be dragged.
OBJSTYLE_NOSHADOW	Cannot have shadow created.
OBJSTYLE_NOMOVE	Cannot move.
OBJSTYLE_NOPRINT	Cannot be printed.
OBJSTYLE_NOTDEFAULTICON	Destroy icon when object goes dormant
OBJSTYLE_TEMPLATE	This object is a template.
OBJSTYLE_NOTVISIBLE	This object is not visible.
OBJSTYLE_NORENAME	Cannot be renamed.

Returns

Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Usage

This method can be called at any time in order to change an object's style.

How to Override

This method is generally not overridden.

Related Methods

- wpclsQueryStyle
- wpQueryStyle

wpSetTitle — WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetTitle (WPObject * self, PSZ pszNewTitle)
```

The wpSetTitle instance method is called to allow the object to set its current title.

Parameters

self (WPObject *) — input

The pointer to this object.

pszNewTitle (PSZ) — input

A pointer to a zero terminated string which contains the title of the object.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

Valid titles must be less than CCHMAXPATHCOMP characters in length — currently defined as 256.

Usage

This method can be called at any time in order to set an object's title.

How to Override

This method is generally not overridden.

Related Methods

- wpQueryTitle
- wpclsQueryTitle

wpSetType – WPFileSystem instance method

#define INCL_WINWORKPLACE

```
BOOL wpSetType  (WPFileSystem * self, PSZ pszTypes)
```

The wpSetType instance method is called to allow the object to change the type of its file.

Parameters

self (WPFileSystem *) – input

The pointer to this object.

pszTypes (PSZ) – input

Pointer to a buffer containing type to set. This string can contain a list of types delineated by a line feed character. For example, psztypes="Plain Text\nC code".

Returns

Success indicator:

True Successful completion

False Error occurred.

Remarks

This method will cause the file's .TYPE extended attribute to be set.

Usage

This method can be called at any time in order to set the type on the file object.

How to Override

This method is generally not overridden.

wpSetup — WPObject instance method

#define INCL_WINWORKPLACE

BOOL wpSetup (WPObject * self, PSZ pszSetupString)

The wpSetup instance method is called to allow the newly created object to initialize itself.

Parameters

self (WPObject *) — input

The pointer to this object.

pszSetupString (PSZ) — input

The pointer to setup string.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

If wpSetup returns FALSE, the creation of the object is terminated. The pszSetupString contains a series of "keyname=value" pairs separated by semicolons, that change the behavior of the object. Each object class defines the keynames and the parameters it expects to see immediately following. Note that all parameters have safe defaults, so it is never required to pass parameters to an object. If a comma or semicolon is needed in the setup string, the escape character ^ can be used.

The following table shows the keyname-value pairs supported by the *WPObject* class.

KEYNAME	VALUE	Description
TITLE	Title	This sets the object's title. This is equivalent to calling the wpSetTitle method.
ICONFILE	filename	This sets the object's icon. This is equivalent to calling the wpSetIconData method.
HELPPANEL	id	This sets the object's default help panel. This is equivalent to calling the wpSetDefaultHelp method.
HELPLIBRARY	filename	This sets the help library.
TEMPLATE	YES	This sets the object's template property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_TEMPLATE.
	NO	This resets the object's template property.

wpSetup –
WPObject instance method

KEYNAME	VALUE	Description
NODELETE	YES	This sets the object's no delete property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_NODELETE.
	NO	This resets the object's no delete property.
NOCOPY	YES	This sets the object's no copy property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_NOCOPY.
	NO	This resets the object's no copy style.
NOMOVE	YES	This sets the object's no move property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_NOMOVE.
	NO	This resets the object's no move property.
NOSHADOW	YES	This sets the object's no link property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_NOSHADOW.
	NO	This resets the object's no link property.
NOTVISIBLE	YES	This sets the object's not visible property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_NOTVISIBLE.
	NO	This resets the object's not visible property.
NOPRINT	YES	This sets the object's no print property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_NOPRINT.
	NO	This resets the object's no print property.
ICONRESOURCE	id,module	This sets the object's icon. This is equivalent to calling the wpSetIconData method. 'id' is the identity of an icon resource in the 'module' dynamic link library (DLL).
ICONPOS	x,y	This sets the object's initial icon position. The x and y values represent the position in the object's folder in percentage coordinates.

KEYNAME	VALUE	Description
OBJECTID	<name>	This sets the object's identity. The object id will stay with the object even if it is moved or renamed. The object pointer or handle can be retrieved via the wpclsQueryObject method or the WinQueryObject function, respectively. An object id is any unique string preceded with a '<' and terminated with a '>'.
NORENAME	YES	This sets the object's no rename property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_NORENAME.
	NO	This resets the object's no rename property
NODRAG	YES	This sets the object's no drag property. This is equivalent to calling the wpSetStyle method with a style of OBJSTYLE_NODRAG.
	NO	This resets the object's no drag property.
VIEWBUTTON	HIDE	Views of this object will have a hide button as opposed to a minimize button.
	MINIMIZE	Views of this object will have a minimize button as opposed to a hide button.
MINWIN	HIDE	Views of this object will hide when their minimize button is selected.
	VIEWER	Views of this object will minimize to the minimized window viewer when their minimize button is selected.
	DESKTOP	Views of this object will minimize to the Desktop when their minimize button is selected.
CONCURRENTVIEW	YES	New views of this object will be created every time the user selects open.
	NO	Open views of this object will resurface when the user selects open.
OPEN	SETTINGS	Open settings view when object is created or when WinSetObjectData is called.
	DEFAULT	Open default view when object is created or when WinSetObjectData is called.

wpSetup – WPObject instance method

Usage

This method is generally only called by the system during the processing of wpclsNew, WinCreateObject, and WinSetObjectData.

How to Override

This method is overridden by classes which introduce their own KEYNAMES.

Related Methods

- WinCreateObject
- wpclsNew
- WinSetObjectData
- wpScanSetupString

Example Code

```
pszSetupString="TITLE=MYObject,ICONFILE=myobj.ico,HELPPANEL=132,
                TEMPLATE=YES,NODELETE=NO"

  obj = wpclsNew ( _WPDataFile,
                   "My new file",
                   pszSetupString,
                   myfolder,
                   FALSE );
```

#define INCL_WINWORKPLACE

```
BOOL wpSetupCell (WPPalette * self, PVOID pCellData, ULONG ulcb, ULONG ulx,
                  ULONG uly)
```

The wpSetupCell instance method is called to initialize a cell.

Parameters

self (WPPalette *) – input

The pointer to the object.

pCellData (PVOID) – input

A pointer to the data to be stored.

ulcb (ULONG) – input

The size of the data to be stored..

ulx (ULONG) – input

X-coordinate of the cell to be setup.

uly (ULONG) – input

Y-coordinate of the cell to be setup.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

Usage

This method can be called at any time in order to initialize a cell.

How to Override

This method is generally not overridden.

wpShowPalettePointer – WPPalette instance method

#define INCL_WINWORKPLACE

```
BOOL wpShowPalettePointer (WPPalette * somSelf)
```

The wpShowPalettePointer instance method displays the applicator pointer for the palette.

Parameters

somSelf (WPPalette *) – input

The pointer to the instance object.

Returns

TRUE The pointer was successfully changed to the applicator symbol for this palette.

FALSE The pointer was not successfully changed to the applicator symbol for this palette.

Remarks

To provide the user with a visual clue to the purpose of a palette window, the mouse cursor is always changed to the applicator symbol when it is within the bounds of an open palette view window. For example, when the cursor is within the color palette window it displays a paintbrush. Note that this method should cause just the applicator to be displayed. When the wpDragCell method is invoked, the applicator *plus* the attribute should be shown if possible. For example, the paintbrush used in the color palette appears to have been dipped in the color that is being applied.

Usage

This method may be called at any time, however it is unlikely to be useful.

How to Override

All subclasses of *wpPalette* need to override this method to ensure that the cursor changes to an applicator while it is within the palette window.

Related Methods

- wpDragCell

wpStartJobAgain – WPJob instance method

```
#define INCL_WINWORKPLACE
```

```
BOOL wpStartJobAgain (WPJob * self)
```

The wpStartJobAgain instance method is called to start printing a job object again.

Parameters

self (WPJob *) – input

The pointer to the object to be re-started.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

How to Override

This method is generally not overridden.

wpSwitchTo – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpSwitchTo (WPObject * self, ULONG ulView)
```

The wpSwitchTo instance method is called to allow the object to give focus to the specified open view.

Parameters

self (WPObject *) – input

The pointer to this object.

ulView (ULONG) – input

A flag indicating open view to which focus is to be given.

OPEN_CONTENTS	Open content view.
OPEN_DEFAULT	Open default view (same as double-click).
OPEN_DETAILS	Open details view.
OPEN_HELP	Display HelpPanel.
OPEN_RUNNING	Execute object.
OPEN_SETTINGS	Open settings notebook.
OPEN_TREE	Open tree view.
OPEN_USER	Class specific views have a greater value than this.

Returns

Success indicator:

TRUE	Successful completion
FALSE	Error occurred.

Remarks

The focus is given to the specified open view of the object if it exists. This is done by scanning the in-use list.

Usage

This method can be called at any time in order to switch to an existing view of this object.

How to Override

This method is generally not overridden.

Related Methods

- wpOpen

wpUnlockObject – WPObject instance method

#define INCL_WINWORKPLACE

```
BOOL wpUnlockObject (WPObject * self)
```

The wpUnlockObject instance method is called to allow an object to go into the dormant state.

Parameters

self (WPObject *) – input

The pointer to the object.

Returns

Success indicator:

TRUE Successful completion.

FALSE Error occurred.

Remarks

The wpUnlockObject method will unlock the specified object. When an object is unlocked, it will automatically go into the dormant state when it no longer has open views and the container it is in is no longer open.

Usage

This method can be called at any time in order to allow the object to go into the dormant state.

This method is used in conjunction with other methods which create new instances of objects. If these methods are called with the fLock flag set to TRUE, the new object will be locked into the awake state until the wpUnlockObject method is called.

How to Override

This method is generally not overridden.

Related Methods

- wpclsCreateDefaultTemplates
- wpQueryContent
- wpCopyObject
- wpclsNew

wpUnInitData – WPObject instance method

#define INCL_WINWORKPLACE

VOID wpUnInitData (WPObject * self)

The wpUnInitData instance method is called to allow the object to free allocated resources.

Parameters

self (WPObject *) – input

The pointer to this object.

Returns

The return value is VOID.

Usage

This method is generally only called by the system when the object is made dormant. The object is made dormant when it is destroyed or when there are no open views and the object and the folder containing the object is not open.

How to Override

This method is overridden to deallocate resources allocated during the processing of wpInitData.

Related Methods

- wpInitData

Workplace Class Methods

The following pages contain an alphabetical listing of the Workplace Class methods.

wpclsCreateDefaultTemplates – WPObject class method

#define INCL_WINWORKPLACE

ULONG wpclsCreateDefaultTemplates (M_WPObject * self, WPFolder * Folder)

The wpclsCreateDefaultTemplates method is called to allow the specified class to create default template instances of its class.

Parameters

self (M_WPObject *) – input

The pointer to the class object.

Folder (WPFolder *) – input

A pointer to the folder in which to create the templates.

Returns

Number of templates created.

Usage

This method is generally only called by the system when the class is registered. A class is registered by a call to the WinRegisterObjectClass function.

When the system calls this method, "Folder" is a pointer to the Templates folder.

How to Override

This method should be overridden by classes which need to create default template instances of their class.

wpclsFindObjectEnd – WPObject class method

#define INCL_WINWORKPLACE

BOOL wpclsFindObjectEnd (M_WPObject * self, HFIND hfindfind)

The wpclsFindObjectEnd method is called to end the find operation started by a call to the wpclsFindObjectFirst method.

Parameters

self (M_WPObject *) – input

The pointer to the class object.

hfindfind (HFIND) – input

The handle associated with a previous wpclsFindObjectFirst or wpclsFindObjectNext method call.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

Usage

This method should be called to terminate a find operation that was started by a previous call to the wpclsFindObjectFirst method.

How to Override

This method is generally not overridden.

Related Methods

- wpclsFindObjectFirst
- wpclsFindObjectNext

wpclsFindObjectFirst – WPObject class method

#define INCL_WINWORKPLACE

```
BOOL wpclsFindObjectFirst (M_WPObject * self, PHFIND pfind, PSZ pszTitle,
                           WPFolder * Folder, BOOL fSubfolders, PVOID pExtendedCriteria,
                           POBJECTS pBuf, PULONG pCount)
```

The wpclsFindObjectFirst method is called to find workplace objects.

Parameters

self (M_WPObject *) – input

The pointer to the class object.

pfind (PHFIND) – output

The address of the handle associated with this wpclsFindObjectFirst method. This handle is used with subsequent calls to wpclsFindObjectNext and wpclsFindObjectEnd.

pszTitle (PSZ) – input

The pointer to the title specification for objects to be searched.

This title may include the wildcard characters "*" and "?."

Folder (WPFolder *) – input

A pointer to the folder in which to find objects.

This pointer can be determined by issuing a call to the wpclsQueryFolder method.

fSubfolders (BOOL) – input
Scope indicator.

FALSE Search only the folder specified by Folder.

TRUE Search Folder and all folders in its tree.

pExtendedCriteria (PVOID) – input

A pointer to a buffer that contains the class-specific extended search criteria.

pBuf (POBJECTS) – output

A pointer to a buffer that contains an array of object pointers.

The size of this buffer must be large enough to hold the number of requested entries specified by pulCount.

pCount (PULONG) – input/output

The address of the number of matching entries requested in pobjectsBuf. On return, this field contains the number of entries placed into pobjectsBuf.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

FALSE can indicate:

WPERR_OBJECT_NOT_FOUND

WPERR_BUFFER_OVERFLOW

Remarks

wpclsFindObjectFirst returns object pointers (up to the number requested in pulCount) for as many objects which match the specifications, and which fits in pobjectsBuf. On output, pulCount contains the actual number of object pointers returned.

wpclsFindObjectNext uses the find object handle associated with wpclsFindObjectFirst to continue the search started by the wpclsFindObjectFirst request.

If wpclsFindObjectFirst returns FALSE, the wpQueryError method can be called to retrieve the error code. Error codes include:

WPERR_OBJECT_NOT_FOUND No objects matching the specified criteria were found.
WPERR_BUFFER_OVERFLOW The pobjectsBuf buffer was not large enough to fit all objects matching the specified criteria. The wpclsFindObjectNext method should be called to retrieve the rest of the objects matching the specified criteria.

The wpclsFindObjectEnd method should be called to terminate the find operation.

Usage

This method can be called at any time in order to find objects.

How to Override

This method is generally not overridden.

Related Methods

- wpclsFindObjectNext
- wpclsFindObjectEnd

wpclsFindObjectNext – WPObject class method

```
#define INCL_WINWORKPLACE
```

```
BOOL wpclsFindObjectNext (M_WPObject * self, HFIND hfindfind, POBJECTS pBuf,
                          PULONG pCount)
```

The wpclsFindObjectNext method is called to find the next set of matching objects.

Parameters

self (M_WPObject *) – input

The pointer to the class object.

hfindfind (HFIND) – input

The handle associated with a previous wpclsFindObjectFirst or wpclsFindObjectNext method call.

pBuf (POBJECTS) – output

A pointer to a buffer that contains an array of object pointers.

The size of this buffer must be large enough to hold the number of requested entries specified by pulCount.

pCount (PULONG) – input/output

Address of the number of matching entries requested in pobjectsBuf. On return, this field contains the number of entries placed into pobjectsBuf.

Returns

Success indicator.

TRUE Successful completion.

FALSE Error occurred.

FALSE can indicate:

WPERR_OBJECT_NOT_FOUND

WPERR_BUFFER_OVERFLOW

Remarks

wpclsFindObjectNext returns object pointers (up to the number requested in pulCount) for as many objects which match the specifications, and which fits in pobjectsBuf. On output, pulCount contains the actual number of object pointers returned.

wpclsFindObjectNext uses the find object handle associated with wpclsFindObjectFirst to continue the search started by the wpclsFindObjectFirst request.

If wpclsFindObjectNext returns FALSE, the wpQueryError method can be called to retrieve the error code. Error codes include:

WPERR_OBJECT_NOT_FOUND No objects matching the specified criteria were found.

WPERR_BUFFER_OVERFLOW The pobjectsBuf buffer was not large enough to fit all objects matching the specified criteria. The wpclsFindObjectNext method should be called to retrieve the rest of the objects matching the specified criteria.

The wpclsFindObjectEnd method should be called to terminate the find operation.

wpclsFindObjectNext — WPObject class method

Usage

This method can be called at any time in order to find the next set of matching objects. This method should only be called after a previous call to the wpclsFindObjectFirst method returned an error of WPERR_BUFFER_OVERFLOW.

How to Override

This method is generally not overridden.

Related Methods

- wpclsFindObjectFirst
- wpclsFindObjectEnd

wpclsInitData – WPObject class method

```
#define INCL_WINWORKPLACE
```

VOID wpclsInitData (M_WPObject * self)

The wpclsInitData method is called to allow the class object to initialize its instance data.

Parameters

self (M_WPObject *) – input

The pointer to the class object.

Returns

The return value is VOID.

Remarks

This method will be called immediately after the class object is first awakened. When the class object is made dormant, the wpclsUnInitData method is called to give the class object the opportunity to deallocate resources allocated during the processing of wpclsInitData.

Usage

This method is generally only called by the system when the class object is awakened. The class object is awakened when the first instance of this class is either awakened or newly created. It is made dormant again when the last instance of this class is made dormant.

How to Override

Any class that has metaclass instance variables should override this method so that those variables are all initially in a known state. It is essential to pass this method onto the parent class object before performing any override processing.

Related Methods

- wpclsUnInitData

wpclsMakeAwake — WPObject class method

#define INCL_WINWORKPLACE

```
WPObject * wpclsMakeAwake (M_WPObject * self, PSZ pszTitle, ULONG ulStyle,
                           HPOINTER hptrIcon, POBJDATA pObjData, WPFolder * Folder,
                           ULONG ulUser)
```

The wpclsMakeAwake method is called to allow the specified class to awaken an object.

Parameters

self (M_WPObject *) — input

The pointer to the class object.

pszTitle (PSZ) — input

A pointer to a string containing a title to set on the object. If this value is NULL, the class default value will be used.

ulStyle (ULONG) — input

The object style flags to set on the object. If this value is NULL, the current or default value will be used.

Predefined object style bits are as follows:

OBJSTYLE_NOCOPY	Do not allow copy on the object.
OBJSTYLE_NODELETE	Do not allow delete on the object.
OBJSTYLE_NODDRAG	Do not allow drag of this object.
OBJSTYLE_NOLINK	Do not allow link on the object.
OBJSTYLE_NOMOVE	Do not allow move on the object.
OBJSTYLE_NOPRINT	Do not allow print of this object.
OBJSTYLE_NOTDEFAULTICON	Destroy icon when object goes to sleep.
OBJSTYLE_NOTVISIBLE	Make this object hidden.
OBJSTYLE_TEMPLATE	This object is a template.

hptrIcon (HPOINTER) — input

The icon to set on the object. If this value is NULL, the class default value will be used.

pObjData (POBJDATA) — input

A pointer to the object data. If this value is NULL, the class default value will be used.

Folder (WPFolder *) — input

A pointer to a folder object which contains the new object. This pointer can be determined by issuing a call to the wpclsQueryFolder method.

ulUser (ULONG) — input

This value is defined by the user and used by the base storage class.

Returns

Success indicator:

NULL Error occurred.

Other Pointer to the awakened object.

wpclsMakeAwake – WPObject class method

Remarks

An object is made awake when it is created in an open folder or when a folder containing the object is opened. An object awakened by the *wpclsMakeAwake* class method is automatically locked. The wpUnlockObject method can be called to allow the object to return to its dormant state.

Usage

This method is generally called only by base storage classes in order to awaken an object from the dormant state.

How to Override

This method is generally not overridden.

Related Methods

- wpclsNew
- wpInitData
- wpUnlockObject

wpclsNew – WPObject class method

#define INCL_WINWORKPLACE

```
WPObject * wpclsNew  (M_WPObject * self, PSZ pszTitle, PSZ pszSetupEnv,
                      WPFolder * Folder, BOOL fLock)
```

The wpclsNew method is called to make an new instance of this class of object.

Parameters

self (M_WPObject *) – input

The pointer to the class in which a new instance is to be created.

pszTitle (PSZ) – input

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.

pszSetupEnv (PSZ) – input

A pointer to a zero terminated string which contains the object-specific parameters to the new object. Refer to wpSetup for the description of this field for the class of object being created. The *pszSetupEnv* string is extracted when the wpSetup method is called.

Folder (WPFolder *) – input

A pointer to a Folder object in which to place this new object. This pointer can be determined by issuing a call to the wpclsQueryFolder method.

fLock (BOOL) – input
The lock object flag.

If this flag is false, the newly created object will be made dormant whenever the object and the folder containing the object are closed. If this flag is true, the new flag will remain awake until the caller issues the wpUnlockObject method on it.

Returns

Success indicator:

NULL Error occurred.

Other A pointer to the new object created.

Remarks

This method is a modified version of somNew that takes arguments. These arguments provide a way to create a new object with a defined state.

Usage

This method can be called at any time in order to create a new workplace object. The created object is a persistent instance of the class specified by WPClass.

How to Override

This method is generally not overridden.

wpclsNew – WPObject class method

Related Methods

- wpFree
- wpclsQueryFolder
- wpUnlockObject
- wpSetup

wpclsQueryDefaultHelp — WPObject class method

#define INCL_WINWORKPLACE

```
ULONG wpclsQueryDefaultHelp  (M_WPObject * self, PULONG pHelpPanelId,
                              PSZ pszHelpLibrary)
```

The wpclsQueryDefaultHelp method is called to allow the class object to specify its default help panel for its instances.

Parameters

self (M_WPObject *) — input

The pointer to the class object.

pHelpPanelId (PULONG) — output

The pointer to the help panel id.

pszHelpLibrary (PSZ) — output

The pointer to the buffer in which to place the name of help library. This buffer should be at least the length of CCHMAXPATH bytes.

Returns

Success indicator:

TRUE Successful completion

FALSE Error occurred.

Remarks

This class method is called during the default processing of wpQueryDefaultHelp.

Usage

This method can be called at any time in order to determine the default help panel for this object class.

How to Override

The default WPObject class does not process this method other than returning FALSE.

Related Methods

- wpDisplayHelp
- wpMenuItemHelpSelected
- wpQueryDefaultHelp
- wpSetDefaultHelp

wpclsQueryDefaultView – WPObject class method

#define INCL_WINWORKPLACE

ULONG wpclsQueryDefaultView (M_WPObject * self)

The wpclsQueryDefaultView method is called to allow the class object to specify the default open view for its instance.

Parameters

self (M_WPObject *) – input

The pointer to the class object.

Returns

Default open view.

OPEN_CONTENTS	Open content view.
OPEN_DEFAULT	Open default view (same as double-click).
OPEN_DETAILS	Open details view.
OPEN_HELP	Display HelpPanel.
OPEN_RUNNING	Execute object.
OPEN_SETTINGS	Open settings notebook.
OPEN_TREE	Open tree view.
OPEN_UNKNOWN	Unknown view.
OPEN_USER	Class specific views have a greater value than this.

Usage

This method can be called at any time in order to query the default open view for instances of this class.

How to Override

All classes should override this method, so that new objects in their class will always have a sensible default view (device objects typically have a default view of OPEN_SETTINGS). The default view is used for both the conditional Open cascade menu and double-clicking on the object.

Related Methods

- wpQueryDefaultView
- wpSetDefaultView

wpclsQueryDetails — WPObject class method

#define INCL_WINWORKPLACE

PCLASSDETAILS wpclsQueryDetails (M_WPObject * self)

The wpclsQueryDetails method is called to allow the class object to specify the default details view items for its instances.

Parameters

self (M_WPObject *) — input

The pointer to the class object.

Returns

Success indicator:

NULL Error occurred.

Other Pointer to details information.

Related Methods

- wpQueryDetailsData
- wpclsQueryDetailsInfo

wpclsQueryDetailsInfo – WPObject class method

#define INCL_WINWORKPLACE

```
ULONG wpclsQueryDetailsInfo (M_WPObject * self, PCLASSFIELDINFO pClassFieldInfo,
                             PULONG pSize)
```

The wpclsQueryDetailsInfo method is called to allow the class object to specify its details to be used for its instances.

Parameters

self (M_WPObject *) – input

The pointer to the class object.

pClassFieldInfo (PCLASSFIELDINFO) – input/output

A pointer to details information.

pSize (PULONG) – input/output

The total number of bytes of details data. This total includes the details added by this class and ancestor classes.

Returns

The sum of the number of detail columns for the object. This sum includes details added by this class and ancestor classes.

Remarks

All objects, which have information to display in details view, must override this method.

The two possible queries are:

Query 1: A request for the CLASSFIELDINFO linked list segment associated with an object. This information is needed just prior to changing the view of container a control to details.

If ppClassFieldInfo is NON-NULL, *ppClassFieldInfo points to the head of a linked list of CLASSFIELDINFO structures to which a linked list of CLASSFIELDINFO structures describing the details fields of objects of this subclass should be appended. (*ppClassFieldInfo may be NULL if no subclasses have appended details data).

For example:

on input *ppClassFieldInfo:

```
→CLASSFIELDINFO_1_grandparent_
→CLASSFIELDINFO_2_grandparent→
  CLASSFIELDINFO_1_parent
```

on output *ppClassFieldInfo:

```
→CLASSFIELDINFO_1_grandparent_
→CLASSFIELDINFO_2_grandparent→
  CLASSFIELDINFO_1_parent
→CLASSFIELDINFO_1_self
```

The correct way to handle this request is to:

- Call the parent method. The parent method returns the number of columns the ancestors have contributed.
- Walk the chain of CLASSFIELDINFO structures to the end and append the chain of CLASSFIELDINFO structures for the current class (using the pNextFieldInfo element).
- If *ppClassFieldInfo is zero (indicating no parent columns), assign *ppClassFieldInfo = beginning of CLASSFIELDINFO chain for this subclass.
- Return the sum of the details columns of the parent and the current subclass.

Note the following differences between the CLASSFIELDINFO and FIELDINFO structures:

- The offFieldData and usLenFieldData fields are required so that appropriate offsets for the data may be computed. The application is responsible for providing offFieldData and usLenFieldData fields.
- The ulReserved field should be left NULL.
- Owner draw, comparison, and sort functions may be specified for the field.

For example if an object has three fields:

```
typedef struct _SAMPLE_DETAIL_DATA {
  CDATE   cdate;
  CTIME   ctime;
  PSZ     psz;
}  SAMPLE_DETAIL_DATA;

classfieldinfo[0].offFieldData   = FIELDOFFSET(SAMPLE,cdate);
classfieldinfo[0].ulLenFieldData = FIELDOFFSET(SAMPLE,ctime)
                                          FIELDOFFSET(SAMPLE,cdate);

classfieldinfo[1].offFieldData   = FIELDOFFSET(SAMPLE,ctime);
classfieldinfo[1].ulLenFieldData = FIELDOFFSET(SAMPLE,psz) -
                                          FIELDOFFSET(SAMPLE,ctime);

classfieldinfo[2].offFieldData   = FIELDOFFSET(SAMPLE,psz);
classfieldinfo[2].ulLenFieldData = sizeof(SAMPLE) -
                                          FIELDOFFSET(SAMPLE,psz);
```

Note that *ppClassFieldInfo must be modified to point to the beginning of the CLASSFIELDINFO linked list only if *ppClassFieldInfo is 0.

The application must return the sum of the details columns of the parent and itself:

```
return(n_cols + n_parent_cols);
```

Note also that the fields MUST be set up sequentially. classfieldinfo[0] must point to data at an offset of 0. classfieldinfo[n] must point to data adjacent and directly following that described by classfieldinfo[–1].

Query 2: The number of bytes of details data associated with an object. This information is needed prior to allocating memory for a container control insert record.

If pSize is NON-NULL, the override should adjust *pSize by the number of bytes which must be added to the end of a MINIRECORDCORE structure to hold the details information for objects of this class.

wpclsQueryDetailsInfo – WPObject class method

Example:

```
*pSize += bytes_of_details_data;
```

In the case of the above example,

```
*pSize += sizeof(SAMPLE_DETAIL_DATA);
```

NOTE: All class field pointers returned should be pointers to static data areas.

Usage

This method is generally only called by the system.

How to Override

This method should be overridden by classes which introduce class-specific details to be displayed in details view.

Related Methods

- wpQueryDetailsData
- wpclsQueryDetails

wpclsQueryEditString – WPPalette class method

#define INCL_WINWORKPLACE

```
PSZ wpclsQueryEditString  (M_WPPalette  *  self)
```

The wpclsQueryEditString method is called to allow the class object to specify the text to be used in the edit pushbutton of the palette object's open view.

Parameters

self (M_WPPalette *) – input

The pointer to the class object.

Returns

A pointer to the edit pushbutton string.

Usage

This method can be called at any time in order to determine the text of the edit pushbutton.

How to Override

This method should be overridden in order to specify class-specific edit pushbutton text. The parent method is generally not called.

wpclsQueryError – WPObject class method

```
#define INCL_WINWORKPLACE
```

ULONG wpclsQueryError (M_WPObject * somSelf)

The wpclsQueryError class method queries the current error code held within a class object.

Parameters

somSelf (M_WPObject *) – input

The pointer to the class object.

Returns

The last error that occurred when using this class object.

Remarks

When an error occurs within a class method and that method subsequently fails, the calling procedure can retrieve the error code for that failed method call by using the *wpclsQueryError* method. Note that the error code is always that of the *last method that failed.* A successful method does not modify the error code held within a class object. This function is analogous to the *WinGetErrorInfo* and the *WinGetLastError* function calls that are used by Presentation Manager applications to diagnose the reason for the previous failing call to a Presentation Manager function call.

The system provided class methods will return error codes as defined in the header file PMERR.H. For example: the wpclsQueryObject method will normally log an error of WPERR_OBJECT_NOT_FOUND if it is unable to return an object pointer.

Usage

This method should be called immediately after a class method has failed, in order to diagnose why the failure occurred.

How to Override

Never override this class method

Related Methods

- wpclsSetError

wpclsQueryFolder – WPObject class method

#define INCL_WINWORKPLACE

```
WPFolder * wpclsQueryFolder (M_WPObject * self, PSZ pszLocation, BOOL fLock)
```

The wpclsQueryFolder method is called to get a pointer to a folder object that corresponds to a given file system location.

Parameters

self (M_WPObject *) – input

The pointer to the class object.

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.
"<WP_DESKTOP>"	The 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.

fLock (BOOL) – input
The lock object flag.

If this flag is false, the newly created object will be made dormant whenever the object and the folder containing the object are closed. If this flag is true, the new flag will remain active until the caller issues the wpUnlockObject method on it.

Returns

Success indicator:

NULL Error occurred.

Other Pointer to a folder object.

Remarks

To obtain a real name from an object pointer, the wpQueryRealName method should be called.

Usage

This method can be called at any time in order to determine the object pointer for a folder.

How to Override

This method is generally not overridden.

wpclsQueryIcon – WPObject class method

```
#define INCL_WINWORKPLACE
```

```
HPOINTER wpclsQueryIcon (M_WPObject * self)
```

The wpclsQueryIcon method is called to allow the class object to specify the default icon to be used for its instances.

Parameters

self (M_WPObject *) – input

The pointer to the class object.

Returns

Success indicator:

NULL Error occurred.

Other Handle to an icon.

Remarks

The class default icon can be loaded on wpclsInitData and freed on wpclsUnInitData.

Usage

This method can be called at any time in order to determine the default icon for instances of this class.

How to Override

This method is overridden in order to change the default icon for an instance of the class.

Related Methods

- wpQueryIcon
- wpQueryIconData
- wpSetIcon
- wpSetIconData
- wpclsQueryIconData

wpclsQueryIconData – WPObject class method

#define INCL_WINWORKPLACE

ULONG wpclsQueryIconData (M_WPObject * somSelf, PICONINFO piconInfo)

The wpclsQueryIconData class method allows the system to build the class default icon for a given class.

Parameters

somSelf (M_WPObject *) – input

The pointer to a class object.

piconInfo (PICONINFO) – input/output

A handle to the container control window. If this parameter is NULLHANDLE, the size should still be returned correctly.

Returns

The size of the buffer needed to accommodate the *ICONOINFO* buffer that is returned by this particular class object.

Remarks

If NULLHANDLE is passed for the piconInfo parameter, the caller is asking for the size of the ICONINFO buffer needed for this class usually for memory allocation purposes. Otherwise, the piconInfo parameter can always be assumed to be large enough to accommodate the ICONINFO for this class.

Note that the ICONINFO structure allows you to specify the default icon in three different ways:

- An icon filename
- A module name and resource identifier
- A block of binary data

However, only one mechanism need be supported any given class. For example, a caller cannot request one of the three formats by prefilling the ICONINFO structure.

Usage

This method may be called at any time. Typically, it would not be useful for another object class to make calls to this method.

How to Override

Workplace classes that wish to have a unique class default icon must override this method and fill out the appropriate fields within the ICONINFO structure. In addition, the correct size for the ICONINFO must always be returned.

Related Methods

- wpQueryIcon
- wpQueryIconData
- wpSetIcon
- wpSetIconData

wpclsQueryInstanceFilter – WPFileSystem class method

#define INCL_WINWORKPLACE

```
PSZ wpclsQueryInstanceFilter  (M_WPFileSystem  *  self)
```

The wpclsQueryInstanceFilter method is called to allow the class object to specify the file title filters for instances of its class.

Parameters

self (M_WPFileSystem *) – input

The pointer to the class object.

Returns

Success indicator.

NULL Error occurred.

Other A pointer to a string containing file title filters. This string can contain several file title filters separated by a comma.

Example: "*.TXT, *.DOC"

Remarks

It is important that the values returned by this class method are restricted to class specific filters. For example, returning a filter of "*.*" could effectively make the system unstable.

Usage

This method can be called at any time in order to determine which file title filters are used to determine instances of this class.

How to Override

This method should be overridden in order to automatically designate file objects as instances of this class. The value returned by the override method will replace the current title filter string which is used to designate instances. If the parent method is called, it should be called first.

Related Methods

- wpclsQueryInstanceType

wpclsQueryInstanceType – WPFileSystem class method

#define INCL_WINWORKPLACE

PSZ wpclsQueryInstanceType (M_WPFileSystem * self)

The wpclsQueryInstanceType method is called to allow the class object to specify the file types for instances of its class.

Parameters

self (M_WPFileSystem *) – input

The pointer to the class object.

Returns

Success indicator.

NULL Error occurred.

Other A pointer to a string containing file types. This string can contain several file types separated by a comma.

Example: "Plain Text,C Code"

Remarks

It is recommended that object classes should define their own special type strings.

Usage

This method can be called at any time in order to determine which file types are used to determine instances of this class.

How to Override

This method should be overridden in order to automatically designate file objects as instances of this class. The value returned by the override method will replace the current type string which is used to designate instances. If the parent method is called, it should be called first.

Related Methods

- wpclsQueryInstanceFilter

wpclsQueryObject – WPObject class method

#define INCL_WINWORKPLACE

```
WPObject * wpclsQueryObject (M_WPObject * somSelf, HOBJECT hObject)
```

The wpclsQueryObject class method returns the object pointer for a given persistent object handle.

Parameters

somSelf (M_WPObject *) – input

The pointer to the class object.

hObject (HOBJECT) – input

The handle for a given object instance.

Returns

The pointer to the object that corresponds to the given object handle or NULLHANDLE if that object no longer exists.

Remarks

All workplace objects can be uniquely identified by a persistent object handle within a given machine. Once an object handle is obtained for an object (using the wpQueryHandle instance method), the handle can be used at any subsequent time even if the system was powered off in the meantime, provided that the object instance has not been destroyed. This method is intended for use by objects that wish to communicate with other objects using method calls. At any point in time, an object can reestablish contact with another object by using this method. For example, a *WPShadow* object will use this method to get the object pointer to the object it is currently shadowing when it is made awake. Note that the returned object is locked, so that the called object can access the returned pointer without the risk of the object being made dormant. When the object pointer is no longer needed, the *wpUnlockObject* method should be invoked to permit the system to make that object dormant when all other locks on it have been released.

Usage

This class method is callable at any time.

How to Override

This class method should not be overridden.

Related Methods

- wpQueryHandle

wpclsQueryOpenFolders – WPFolder class method

#define INCL_WINWORKPLACE

```
WPFolder * wpclsQueryOpenFolders (M_WPFolder * self, WPFolder * Folder,
                                  ULONG ulOption, BOOL fLock)
```

The wpclsQueryOpenFolders method is called to allow the specified class to enumerate all open folders.

Parameters

self (M_WPFolder *) – input

The pointer to the class object.

Folder (WPFolder *) – input

The pointer to a folder object.

This field is ignored unless QC_NEXT is specified in *ulOption*

ulOption (ULONG) – input

A flag indicating the folder to query:

QC_FIRST Return the first open folder

QC_LAST Return the last open folder

QC_NEXT Return the next open folder after *Folder*.

fLock (BOOL) – input

The lock object flag.

If this flag is false, the newly created object will be made dormant whenever the object and the folder containing the object are closed. If this flag is true, the new flag will remain active until the caller issues the wpUnlockObject method on it.

Returns

Success indicator.

NULL Error occurred or QC_NEXT was requested on last folder.

Other A pointer to the specified folder object.

Usage

This method can be called at any time in order to determine the open folders.

How to Override

This method is generally not overridden.

wpclsQuerySettingsPageSize – WPObject class method

```
#define INCL_WINWORKPLACE

BOOL wpclsQuerySettingsPageSize (M_WPObject * somSelf, PSIZEL pSizl)
```

The wpclsQuerySettingsPageSize class method returns the default size of a settings page in dialog units for instances of this object class.

Parameters

somSelf (M_WPObject *) – input

The pointer to the class object.

pSizl (PSIZEL) – input/output

A pointer to the size structure containing the *cx* and *cy* dimensions of the default settings page for this class.

Returns

TRUE The method class call was successful.

FALSE The method class call was unsuccessful.

Remarks

An object class that has abnormally shaped settings pages (for example the pages may be very wide) can use this method to ensure that when a settings notebook is initially displayed for an instance of the object class, that settings notebook will be of a suitable size to view the whole settings page without clipping it. The size specified by this method is only used the very first time that a settings notebook is displayed, thereafter the settings notebook size and position will be saved by the system on a per object basis.

Usage

This function is callable, but unlikely to be useful.

How to Override

Object classes with unusual size requirements for their settings pages must override this class method and change the *cx* and *cy* values as necessary to accommodate their settings page sizes.

Related Methods

- wpOpen

wpclsQueryStyle – WPObject class method

#define INCL_WINWORKPLACE

```
ULONG wpclsQueryStyle  (M_WPObject * self)
```

The wpclsQueryStyle method is called to allow the class object to specify the default object class style for its instances.

Parameters

self (M_WPObject *) – input

The pointer to the class object.

Returns

Class style for this object:

CLSSTYLE_NEVERMOVE	Do not allow move of this object.
CLSSTYLE_NEVERSHADOW	Do not allow create shadow of this object.
CLSSTYLE_NEVERCOPY	Do not allow copy of this object.
CLSSTYLE_NEVERTEMPLATE	Do not allow create template for this object.
CLSSTYLE_NEVERDELETE	Do not allow delete of this object.
CLSSTYLE_NEVERPRINT	Do not allow print of this object.
CLSSTYLE_NEVERDRAG	Do not allow drag of this object.
CLSSTYLE_NEVERVISIBLE	Make instances always invisible.
CLSSTYLE_NEVERRENAME	Do not allow the renaming of this object.

Remarks

When an instance is initially created, it has the same object style (OBJSTYLE_xxx) flags as its class style (CLSSTYLE_xxx).

Usage

This method can be called at any time in order to determine the default style for instances of this class.

How to Override

This method should be overridden in order to modify the default object style for instances of this class.

Related Methods

- wpQueryStyle
- wpSetStyle

wpclsQueryTitle – WPObject class method

```
#define INCL_WINWORKPLACE
```

```
PSZ wpclsQueryTitle  (M_WPObject * self)
```

The wpclsQueryTitle method is called to allow the class object to specify the default title for its instances.

Parameters

self (M_WPObject *) – input

The pointer to the class object.

Returns

Pointer to default title for objects of this class.

Remarks

The title is used as the default for new instances. In addition, the title is used to describe the class in facilities such as Find, Include, Details, and Sort. The title can be loaded on wpclsInitData and freed on wpclsUnInitData.

Usage

This method can be called at any time in order to determine the default title for instances of this class.

How to Override

All classes should override this method, so that new objects and their classes always have a sensible default title.

Related Methods

- wpQueryTitle
- wpSetTitle

wpclsSetError – WPObject class method

#define INCL_WINWORKPLACE

```
BOOL wpclsSetError  (M_WPObject * somSelf, ULONG ulErrorId)
```

The wpclsSetError class method sets the current error code within a class object.

Parameters

somSelf (M_WPObject *) – input

The pointer to the class object.

ulErrorId (ULONG) – input

The error code.

Returns

TRUE The error was successfully stored.

FALSE The error was not successfully stored.

Remarks

This method can be used when writing class methods for workplace objects that return boolean values. When a given class method fails, the class method can log its error code in the class object so that the caller can later retrieve it using the wpclsQueryError class method. The function is analogous to the *WinSetErrorInfo* function call that is used by Presentation Manager functions to log their error return codes.

Usage

It is recommended that this method should only be called by all class methods when they do not execute successfully, and nowhere else.

How to Override

Never override this class method

Related Methods

- wpclsQueryError

wpclsUnInitData – WPObject class method

#define INCL_WINWORKPLACE

```
VOID wpclsUnInitData (M_WPObject * self)
```

The wpclsUnInitData method is called to allow the class object to free allocated resources.

Parameters

self (M_WPObject *) – input

The pointer to the class object.

Returns

The return value is VOID.

Usage

This method is generally only called by the system when the class object is made dormant. The class object is made dormant when the last instance of this class is made dormant.

How to Override

Any class that overrides the wpclsInitData method to allocate resources for its metaclass instance variables should override this method to deallocate those resources. It is essential to pass this method onto the parent class object after performing override processing.

Related Methods

- wpclsInitData

Glossary

A

accelerator. A single key stroke that invokes an application-defined function.

accelerator table. Used to define which key strokes are treated as *accelerators* and the commands they are translated into.

access permission. All access rights that a user has regarding an object.

action. One of a set of defined tasks that a computer performs. Users request the application to perform an action in several ways, such as typing a command, pressing a function key, or selecting the action name from an action bar or menu.

action bar. The area at the top of a window that contains the choices currently available in the application program.

action point. The current position on the screen at which the pointer is pointing. (Contrast with *hot spot* and *input focus*.)

active program. A program currently running on the computer. See also *interactive program*, *noninteractive program*, and *foreground program*.

active window. The window with which the user is currently interacting.

address space. (1) The range of addresses available to a program. (2) The area of virtual storage available for a particular job.

alphanumeric video output. Output to the logical video buffer when the video adapter is in text mode and the logical video buffer is addressed by an application as a rectangular array of character cells.

anchor block. An area of Presentation Manager-internal resources allocated to a process or thread that calls WinInitialize.

anchor point. A point in a window used by a program designer or by a window manager to position a subsequently appearing window.

ANSI. American National Standards Institute.

APA. All points addressable.

API. Application programming interface. The formally-defined programming language that is between an IBM application program and the user of the program. See also *GPI*.

area. In computer graphics, a filled shape such as a solid rectangle.

ASCII. American National Standard Code for Information Interchange. A coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, data communications systems, and associated equipment.

ASCIIZ. A string of ASCII characters that is terminated with a byte containing the value 0.

aspect ratio. In computer graphics, the width-to-height ratio of an area, symbol, or shape.

asynchronous. (1) Without regular time relationship. (2) Unexpected or unpredictable with respect to the execution of a program's instructions. See also *synchronous*.

atom. A constant that represents a string. Once a string has been defined as an atom, the atom can be used in place of the string to save space. Strings are associated with their respective atoms in an *atom table*. See also *integer atom*.

atom table. Used to relate *atoms* with the strings that they represent. Also in the table is the mechanism by which the presence of a string can be checked.

attributes. Characteristics or properties that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also *graphics attributes* and *segment attributes*.

AVIO. Advanced Video Input/Output.

B

background color. The color in which the background of a graphic primitive is drawn.

background mix. An attribute that determines how the background of a graphic primitive is combined with the existing color of the graphics presentation space. Contrast with *mix*.

background program. In multiprogramming, a program that executes with a low priority. Contrast with *foreground program*.

Bézier curves. A mathematical technique of specifying smooth continuous lines and surfaces, which require a starting point and a finishing point with several intermediate points that influence or control the path of the linking curve. Named after Dr. P. Bézier.

bit map. A representation in memory of the data displayed on an APA device, usually the screen.

block. (1) A string of data elements recorded or transmitted as a unit. The elements may be characters, words, or logical records. (2) To combine two or more data elements in one block.

border. A visual indication (for example, a separator line or a background color) of the boundaries of a window.

breakpoint. (1) An instruction in a program for halting execution. Breakpoints are usually established at positions in a program where halts, caused by external intervention, are convenient for restarting. (2) A place in a program, specified by a command or a condition, where the system halts execution and gives control to the workstation user or to a specified program.

bucket. One or more fields in which the result of an operation is kept.

buffer. (1) A portion of storage used to hold input or output data temporarily. (2) To allocate and schedule the use of buffers.

button. A mechanism on a *pointing device*, such as a mouse, used to request or initiate an action. Contrast with *pushbutton* and *radio button*.

C

cache. A high-speed buffer storage that contains frequently accessed instructions and data; it is used to reduce access time.

cached micro presentation space. A presentation space from a Presentation Manager-owned store of micro presentation spaces. It can be used for drawing to a window only, and must be returned to the store when the task is complete.

call. (1) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (2) To transfer control to a procedure, program, routine, or subroutine.

calling order. A sequence of instructions together with any associated data necessary to perform a call. Also known as *calling sequence*.

cancel. An action that removes the current window or menu without processing it, and returns the previous window.

CASE statement. In C, provides the body of a window procedure. There is one CASE statement for each message type written to take specific actions.

cell. See *character cell*.

CGA. Color graphics adapter.

chained list. A list in which the data elements may be dispersed but in which each data element contains information for locating the next. Synonym for *linked list*.

character. A letter, digit, or other symbol.

character box. In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single character from a character set. See also *character mode*. Contrast with *character cell*.

character cell. The physical, rectangular space in which any single character is displayed on a screen or printer device. Position is addressed by row and column coordinates. Contrast with *character box*.

character code. The means of addressing a character in a character set, sometimes called *code point*.

character mode. The character mode, in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear, and angle attributes.

check box. A control window, shaped like a square button on the screen, that can be in a checked or unchecked state. It is used to select one or more items from a list. Contrast with *radio button*.

check mark. The symbol that is used to indicate a selected item on a pull-down.

child process. A process that is loaded and started by another process. Contrast with *parent process*.

child window. A window that is positioned relative to another window (either a main window or another child window). Contrast with *parent window*.

choice. An option that can be selected. The choice can be presented as text, as a symbol (number or letter), or as an icon (a pictorial symbol).

class. See *window class*.

class style. The set of properties that apply to every window in a window class.

client area. The area in the center of a window that contains the main information of the window.

clipboard. An area of main storage that can hold data being passed from one PM application to another. Various data formats can be stored.

clipping. In computer graphics, removing those parts of a display image that lie outside a given boundary.

clip limits. The area of the paper that can be reached by a printer or plotter.

clipping path. A clipping boundary in world-coordinate space.

CLOCK$. Character-device name reserved for the system clock.

code page. An assignment of graphic characters and control-function meanings to all code points.

code point. Synonym for *character code*.

code segment. An executable section of programming code within a load module.

color dithering. See *dithering*.

command. The name and parameters associated with an action that a program can perform.

command area. An area composed of a command field prompt and a command entry field.

command entry field. An entry field in which users type commands.

command line. On a display screen, a display line usually at the bottom of the screen, in which only commands can be entered.

command prompt. A field prompt showing the location of the command entry field in a panel.

Common Programming Interface (CPI). A consistent set of specifications for languages, commands, and calls to enable applications to be developed across all SAA environments. See also *Systems Application Architecture*.

Common User Access (CUA). A set of rules that define the way information is presented on the screen, and the techniques for the user to interact with the information.

compile. To translate a program written in a higher-level programming language into a machine language program.

COM1, COM2, COM3. Character-device names reserved for serial ports 1 through 3.

CON. Character-device name reserved for the console keyboard and screen.

contiguous. Touching or joining at a common edge or boundary, for example, an unbroken consecutive series of storage locations.

control. The means by which an operator gives input to an application. A *choice* corresponds to a control.

Control Panel. In PM, a program used to set up user preferences that act globally across the system.

Control Program. The basic function of OS/2, including DOS emulation and the support for keyboard, mouse, and video input/output.

control window. A class of window used to handle a specific kind of user interaction. Radio buttons and check boxes are examples.

correlation. The action of determining which element or object within a picture is at a given position on the display. This follows a *pick* operation.

CPI. Common Programming Interface.

critical extended attribute. An extended attribute that is necessary for the correct operation of the system or a particular application.

CUA. Common User Access.

current position. The point from which the next primitive will be drawn.

cursor. A symbol displayed on the screen and associated with an input device. The cursor indicates where input from the device will be placed. Types of cursors include text cursors, graphics cursors, and selection cursors. Contrast with *pointer* and *input focus*.

D

data structure. (ISO) The syntactic structure of symbolic expressions and their storage-allocation characteristics.

DBCS. See *double-byte character set*.

deadlock. (1) Unresolved contention for the use of a resource. (2) An error condition in which processing cannot continue because each of two elements of the process is waiting for an action by, or a response from, the other. (3) An impasse that occurs when multiple processes are waiting for the availability of a resource that will not become available because it is being held by another process that is in a similar wait state.

debug. To detect, diagnose, and eliminate errors in programs.

decipoint. In printing, one tenth of a point. There are 72 points in an inch.

default procedure. Function provided by the Presentation Interface that may be used to process standard messages from dialogs or windows.

default value. A value used when no value is explicitly specified by the user. For example, in the graphics programming interface, the default line-type is 'solid'.

descendant. A process or session that is loaded and started by a parent process or parent session.

Desktop Manager. In PM, a window that displays a list of groups of programs, each of which can be started or stopped.

desktop window. The window, corresponding to the physical device, against which all other types of windows are established.

device context. A logical description of a data destination such as memory, metafile, display, printer, or plotter. See also *direct device context*, *information device context*, *memory device context*, *metafile device context*, *queued device context*, and *screen device context*.

device driver. A file that contains the code needed to attach and use a device such as a display, printer, or plotter.

device space. Coordinate space in which graphics are assembled after all GPI transformations have been applied. Device space is defined in device-specific units.

dialog. The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box. A type of window that contains one or more controls for the formatted display and entry of data. Also known as a *pop-up window*. A modal dialog box is used to implement a pop-up window.

Dialog Box Editor. A *WYSIWYG* editor that creates dialog boxes for communicating with the application user.

dialog item. A component (for example, a menu or a button) of a dialog box. Dialog items are also used when creating dialog templates.

dialog tag language. A markup language used by the DTL compiler to create dialog objects.

dialog template. The definition of a dialog box, which contains details of its position, appearance, and window ID, and the window ID of each of its child windows.

direct device context. A logical description of a data destination that is a device other than the screen (for example, a printer or plotter), and where the output is not to go through the spooler. Its purpose is to satisfy queries. See also *device context*.

direct manipulation. The action of using the mouse to move objects around the screen. For example, moving files and directories around in the *File Manager*.

direct memory access (DMA). The transfer of data between main storage and input/output devices without intervention by the processor.

directory. A type of file containing the names and controlling information for other files or other directories.

display point. Synonym for *pel*.

dithering. The process used in color displays whereby every other pel is set to one color, and the intermediate pels are set to another. Together they produce the effect of a third color at normal viewing distances. This process can only be used on solid areas of color; it does not work on narrow lines, for example.

DMA. Direct memory access.

double-byte character set (DBCS). A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more characters than can be represented by 256 code points, require double-byte character sets. Since each character requires two bytes, the entering, displaying, and printing of DBCS characters requires hardware and software that can support DBCS.

doubleword. A contiguous sequence of bits or characters that comprises two computer words and is capable of being addressed as a unit.

dragging. In computer graphics, moving an object on the display screen as if it were attached to the pointer.

drawing chain. See *segment chain*.

drop. To fix the position of an object that is being dragged, by releasing the select button of the pointing device.

DTL. See *dialog tag language*.

dual-boot function. A feature of OS/2 that allows the user to start DOS from within OS/2, or OS/2 from within DOS.

duplex. Pertaining to communication in which data can be sent and received at the same time. Synonymous with *full duplex*.

dynamic linking. The process of resolving external references in a program module at load time or run time rather than during linking.

dynamic-link library. A collection of executable programming code and data that is bound to an application at load time or run time, rather than during linking. The programming code and data in a dynamic link library can be shared by several applications simultaneously.

dynamic-link module. A module that is linked at load time or run time.

dynamic segments. Graphics segments drawn in exclusive-OR mix mode so that they can be moved from one screen position to another without affecting the rest of the displayed picture.

dynamic storage. (1) A device that stores data in a manner that permits the data to move or vary with time such that the specified data is not always available for recovery. (2) A storage in which the cells require repetitive application of control signals in order to retain stored data. Such repetitive application of the control signals is called a refresh operation. A dynamic storage may use static addressing or sensing circuits. (3) See also *static storage*.

E

EBCDIC. Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters (9 bits including parity check), used for information interchange among data processing systems, data communications systems, and associated equipment.

EGA. Extended graphics adapter.

8.3 file-name format. A file-naming convention in which file names are limited to eight characters before and three characters after a single dot. Usually pronounced "eight-dot-three." See also *non-8.3 file-name format*.

element. An entry in a graphics segment that comprises one or more graphics orders and that is addressed by the element pointer.

entry field. An area on the screen, usually highlighted in some manner, in which users type information.

entry-field control. The means by which the application receives data entered by the user in an entry field. When it has the input focus, it displays a flashing pointer at the position where the next typed character will go.

entry panel. A defined panel type containing one or more entry fields and protected information such as headings, prompts, and explanatory text.

exception. An abnormal condition such as an I/O error encountered in processing a data set or a file.

exclusive system semaphore. A system semaphore that can be modified only by threads within the same process.

exit. The action that terminates the current function and returns the user to a higher level function. Repeated exit requests return the user to the point from which all functions provided to the system are accessible. Contrast with *cancel*.

extended attribute. An additional piece of information about a file object, such as its data format or category. It consists of a name and a value. A file object may have more than one extended attribute associated with it.

extended-choice selection. A mode that allows the user to select more than one item from a window. Not all windows allow extended choice selection. Contrast with *multiple-choice selection*.

extended help. A facility that provides users with information about an entire application panel rather than a particular item on the panel.

extent. Continuous space on a disk or diskette that is occupied by or reserved for a particular data set, data space, or file.

F

family-mode application. An application program that can run in the OS/2 environment and in the DOS environment. However, it cannot take advantage of many of the OS/2-mode facilities, such as multitasking, interprocess communication, and dynamic linking.

FAT. File allocation table.

FEA. Full extended attribute.

field-level help. Information specific to the field on which the cursor is positioned. This help function is "contextual" because it provides information about a specific item as it is currently used; the information is dependent upon the context within the work session.

file. A named set of records stored or processed as a unit.

file allocation table (FAT). In IBM personal computers, a table used by the operating system to allocate space on a disk for a file, and to locate and chain together parts of the file that may be scattered on different sectors so that the file can be used in a random or sequential manner.

file attribute. Any of the attributes that describe the characteristics of a file.

File Manager. In PM, a program that displays directories and files, and allows various actions on them.

file specification. The full identifier for a file, which includes its drive designation, path, file name, and extension.

file system driver (FSD). A program that manages file I/O and controls the format of information on the storage media.

fillet. A curve that is tangential to the end points of two adjoining lines. See also *polyfillet*.

flag. (1) An indicator or parameter that shows the setting of a switch. (2) A character that signals the occurrence of some condition, such as the end of a word.

focus. See *input focus*.

font. A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

foreground program. The program with which the user is currently interacting. Also known as *interactive program*. Contrast with *background program*.

frame. The part of a window that can contain several different visual elements specified by the application, but drawn and controlled by PM. The frame encloses the client area.

frame styles. Different standard window layouts provided by PM.

FSD. File system driver.

full duplex. Synonym for *duplex*.

full-screen application. An application program that occupies the whole screen.

function. (1) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a call. (2) A set of related control statements that cause one or more programs to be performed.

function key. A key that causes a specified sequence of operations to be performed when it is pressed, for example, F1 and Alt-K.

function key area. The area at the bottom of a window that contains function key assignments such as F1 = Help.

G

GDT. Global Descriptor Table.

general protection fault. An exception condition that occurs when a process attempts to use storage or a module that has some level of protection assigned to it, such as I/O privilege level. See also *IOPL code segment*.

Global Descriptor Table (GDT). Defines code and data segments available to all tasks in an application.

global dynamic-link module. A dynamic-link module that can be shared by all processes in the system that refer to the module name.

global file-name character. A special character used to refer to a set of file objects with a common base name. The asterisk (*) and question mark (?) are used as global file-name characters. For example, *.EXE can be used to refer to a set of files with the extension EXE.

glyph. A graphic symbol whose appearance conveys information.

GPI. Graphics programming interface. The formally-defined programming language that is between an IBM graphics program and the user of the program. See also *API*.

graphics. A picture defined in terms of graphic primitives and graphics attributes.

graphics attributes. Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition. See also *segment attributes*.

graphics field. The clipping boundary that defines the visible part of the presentation-page contents.

graphics model space. The conceptual coordinate space in which a picture is constructed after any model transforms have been applied. Also known as *model space*.

graphic primitive. A single item of drawn graphics, such as a line, arc, or graphics text string. See also *graphics segment*.

graphics segment. A sequence of related graphic primitives and graphics attributes. See also *graphic primitive*.

graying. The indication that a choice on a pull-down is unavailable.

group. A collection of logically-connected controls. For example, the buttons controlling paper size for a printer. See also *program group*.

H

handle. An identifier that represents an object, such as a device or window, to the Presentation Interface.

hard error. An error condition on a network that requires either that the system be reconfigured, or that the source of the error be removed before the system can resume reliable operation.

header. (1) System-defined control information that precedes user data. (2) The portion of a message that contains control information for the message, such as one or more destination fields, name of the originating station, input sequence number, character string indicating the type of message, and priority level for the message.

help. A function that provides information about a specific field, an application panel, or information about the help facility.

help index. A facility that allows the user to select topics for which help is available.

help panel. A panel with information to assist users that is displayed in response to a help request from the user.

help window. A Common User Access-defined secondary window that displays information when the user requests help.

heap. An area of free storage available for dynamic allocation by an application. Its size varies according to the storage requirements of the application.

hit testing. The means of identifying which window is associated with which input device event.

hook. A mechanism by which procedures are called when certain events occur in the system. For example, the filtering of mouse and keyboard input before it is received by an application program.

hook chain. A sequence of hook procedures that are "chained" together so that each event is passed, in turn, to each procedure in the chain.

hot spot. The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer. Contrast with *action point*.

I

icon. A pictorial representation of an item the user can select. Icons can represent items (such as a document file) that the user wants to work on, and actions that the user wants to perform. In PM, icons are used for data objects, system actions, and minimized programs.

icon area. In PM, the area at the bottom of the screen that is normally used to display the icons for minimized programs.

Icon Editor. The Presentation Manager-provided tool for creating icons.

image font. A set of symbols, each of which is described in a rectangular array of pels. Some of the pels in the array are set to produce the image of the symbol. Contrast with *outline font*.

information device context. A logical description of a data destination other than the screen (for example, a printer or plotter), but where no output will occur. Its purpose is to satisfy queries. See also *device context*.

information panel. A defined panel type characterized by a body containing only protected information.

input focus. The area of the screen that will receive input from an input device (typically the keyboard).

input router. An internal OS/2 process that removes messages from the system queue.

integer atom. A special kind of *atom* that represents a predefined system constant and carries no storage overhead. For example, names of window classes provided by PM are expressed as integer atoms.

interactive graphics. Graphics that can be moved or manipulated by a user at a terminal.

interactive program. A program that is running (active) and is ready to receive (or is receiving) input from the user. Compare with *active program* and contrast with *noninteractive program*.

Also known as a *foreground program*.

interchange file. Data that can be sent from one Presentation Interface application to another.

interval timer. (1) A timer that provides program interruptions on a program-controlled basis. (2) An electronic counter that counts intervals of time under program control.

IOCtl. A device-specific command that requests a function of a device driver through the *DosDevIOCtl* function.

I/O operation. An input operation to, or output operation from a device attached to a computer.

IOPL. Input/output privilege level.

IOPL code segment. An IOPL executable section of programming code that enables an application to directly manipulate hardware interrupts and ports without replacing the device driver. See also *privilege level*.

J

journal. A special-purpose file that is used to record changes made in the system.

K

Kanji. A graphic character set used in Japanese ideographic alphabets.

KBD$. Character-device name reserved for the keyboard.

kernel. The part of an operating system that performs basic functions, such as allocating hardware resources.

kerning. The design of graphics characters so that their character boxes overlap. Used to space text proportionally.

keys help. A facility that gives users a listing of all the key assignments for the current application.

L

label. In a graphics segment, an identifier of one or more elements that is used when editing the segment.

language support procedure. Function provided by the Presentation Interface for applications that do not, or cannot (as in the case of COBOL and FORTRAN programs), provide their own dialog or window procedures.

LDT. Local Descriptor Table.

LIFO stack. A data stack from which data is retrieved in last-in, first-out order.

linked list. Synonym for *chained list*.

list box. A control window containing a vertical list of selectable descriptions.

list panel. A defined panel type that displays a list of items from which users can select one or more choices and then specify one or more actions to work on those choices.

load-on-call. A function of a linkage editor that allows selected segments of the module to be disk resident while other segments are executing. Disk resident segments are loaded for execution and given control when any entry point that they contain is called.

load time. The point in time at which a program module is loaded into main storage for execution.

local area network (LAN). A data network located on the user's premises in which serial transmission is used for direct data communication among data stations.

Local Descriptor Table (LDT). Defines code and data segments specific to a single task.

lock. A serialization mechanism by means of which a resource is restricted for use by the holder of the lock.

LPT1, LPT2, LPT3. Character-device names reserved for parallel printers 1 through 3.

M

main window. The window that is positioned relative to the *desktop window*.

map. (1) A set of values having a defined correspondence with the quantities or values of another set. (2) To establish a set of values having a defined correspondence with the quantities or values of another set.

marker box. In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single marker from a marker set.

marker symbol. A symbol centered on a point. Graphs and charts can use marker symbols to indicate the plotted points.

maximize. A window-sizing action that makes the window the largest size possible.

media window. The part of the physical device (display, printer, or plotter) on which a picture is presented.

memory device context. A logical description of a data destination that is a memory bit map. See also *device context*.

memory management. A feature of the operating system for allocating, sharing, and freeing main storage.

menu. A type of panel that consists of one or more selection fields. Also called a *menu panel*.

message. (1) In PM, a packet of data used for communication between the Presentation Interface and windowed applications. (2) In a user interface, information not requested by users but presented to users by the computer in response to a user action or internal process.

message filter. The means of selecting which messages from a specific window will be handled by the application.

message queue. A sequenced collection of messages to be read by the application.

metafile. The generic name for the definition of the contents of a picture. Metafiles are used to allow pictures to be used by other applications.

metafile device context. A logical description of a data destination that is a metafile, which is used for graphics interchange. See also *device context*.

metalanguage. A language used to specify another language. For example, data types can be described using a metalanguage so as to make the descriptions independent of any one computer language.

mickey. A unit of measurement for physical mouse motion whose value depends on the mouse device driver currently loaded.

micro presentation space. A graphics presentation space in which a restricted set of the GPI function calls is available.

minimize. A window-sizing action that makes the window the smallest size possible. In PM, minimized windows are represented by icons.

mix. An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as *foreground mix*. Contrast with *background mix*.

mixed character string. A string containing a mixture of one-byte and *Kanji* or Hangeul (two-byte) characters.

mnemonic. A method of selecting an item on a pull-down by means of typing the highlighted letter in the menu item.

modal dialog box. The type of control that allows the operator to perform input operations on only the current dialog box or one of its child windows. Also known as a *serial dialog box*. Contrast with *parallel dialog box*.

modeless dialog box. The type of control that allows the operator to perform input operations on any of the application's windows. Also known as a *parallel dialog box*. Contrast with *modal dialog box*.

model space. See *graphics model space*.

module definition file. A file that describes the code segments within a load module. For example, it indicates whether a code segment is loadable before module execution begins (preload), or loadable only when referred to at run time (load-on-call).

mouse. A hand-held device that is moved around to position the pointer on the screen.

MOUSE$. Character-device name reserved for a mouse.

multiple-choice selection. A mode that allows users to select any number of choices, including none at all. See also *check box*. Contrast with *extended-choice selection*.

multitasking. The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

N

named pipe. A named buffer that provides client-to-server, server-to-client, or full duplex communication between unrelated processes. Contrast with *unnamed pipe*.

noncritical extended attribute. An extended attribute that is not necessary for the function of an application.

nondestructive read. A read process that does not erase the data in the source location.

non-8.3 file-name format. A file-naming convention in which path names can consist of up to 255 characters. See also *8.3 file-name format*.

noninteractive program. A program that is running (active) but is not ready to receive input from the user. Compare with *active program*, and contrast with *interactive program*.

nonretained graphics. Graphic primitives that are not remembered by the Presentation Interface once they have been drawn. Contrast with *retained graphics*.

NUL. Character-device name reserved for a nonexistent (dummy) device.

null-terminated string. A string of (n + 1) characters where the (n + 1)th character is the 'null' character (X'00'), and is used to represent an n-character string with implicit length. Also known as 'zero-terminated' string and 'ASCIIZ' string.

O

object window. A window that does not have a parent, but which may have child windows. An object window cannot be presented on a device.

open. To start working with a file, directory, or other object.

outline font. A set of symbols, each of which is created as a series of lines and curves. Synonymous with *vector font*. Contrast with *image font*.

output area. The area of the output device within which the picture is to be displayed, printed, or plotted.

owner window. A window into which specific events that occur in another (owned) window are reported.

owning process. The process that owns the resources that may be shared with other processes.

P

page. A 4KB segment of contiguous physical memory.

page viewport. A boundary in device coordinates that defines the area of the output device in which graphics are to be displayed. The presentation-page contents are transformed automatically to the page viewport in device space.

paint. The action of drawing or redrawing the contents of a window.

panel. A particular arrangement of information grouped together for presentation to the user in a window.

panel area. An area within a panel that contains related information. The three major Common User Access-defined panel areas are the action bar, the function key area, and the panel body.

panel body. The portion of a panel not occupied by the action bar, function key area, title or scroll bars. The panel body may contain protected information, selection fields, and entry fields. The layout and content of the panel body determine the panel type.

panel body area. The part of a window not occupied by the action bar or function key area. The panel body area may contain information, selection fields, and entry fields. Also known as *client area*.

panel body area separator. A line or color boundary that provides users with a visual distinction between two adjacent areas of a panel.

panel definition. A description of the contents and characteristics of a panel. A panel definition is the application developer's mechanism for predefining the format to be presented to users in a window.

panel ID. A panel element located in the upper left-hand corner of a panel body that identifies that particular panel within the application.

panel title. A panel element that identifies the information in the panel.

paper size. The size of paper, defined in either standard U.S. or European names (for example, A, B, A4), and measured in inches or millimeters respectively.

parallel dialog box. See *modeless dialog box*.

parent process. A process that loads and starts other processes. Contrast with *child process*.

parent window. The window relative to which one or more child windows are positioned. Contrast with *child window*.

partition. (1) A fixed-size division of storage. (2) On an IBM personal computer fixed disk, one of four possible storage areas of variable size; one may be accessed by DOS, and each of the others may be assigned to another operating system.

path. The part of a file specification that lists a series of directory names. Each directory name is separated by the backslash character. In the file specification C:\MYFILES\MISC\GLOSSARY.SCR, the path consists of MYFILES\MISC\.

pel. The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonym for *display point*, *pixel*, and *picture element*.

pick. To select part of a displayed object using the pointer.

picture chain. See *segment chain*.

picture element. Synonym for *pel*.

PID. Process identification.

pipe. A named or unnamed buffer used to pass data between processes. A process reads from or writes to a pipe as if the pipe were a standard-input or standard-output file. See also *named pipe* and *unnamed pipe*.

pixel. Synonym for *pel*.

plotter. An output device that uses pens to draw its output on paper or on transparency foils.

PM. Presentation Manager.

pointer. (1) The symbol displayed on the screen that is moved by a pointing device, such as a *mouse*. The pointer is used to point at items that users can select. Contrast with *cursor*. (2) A data element that indicates the location of another data element.

POINTER$. Character-device name reserved for a pointer device (mouse screen support).

pointing device. A device (such as a mouse) used to move a pointer on the screen.

pointings. Pairs of x-y coordinates produced by an operator defining positions on a screen with a pointing device, such as a *mouse*.

polyfillet. A curve based on a sequence of lines. It is tangential to the end points of the first and last lines, and tangential also to the midpoints of all other lines. See also *fillet*.

polyline. A sequence of adjoining lines.

pop. To retrieve an item from a last-in-first-out stack of items. Contrast with *push*.

pop-up window. A window that appears on top of another window in a dialog. Each pop-up window must be completed before returning to the underlying window.

Presentation Manager (PM). The visual component of OS/2 that presents, in windows, a graphics-based interface to applications and files installed and running in OS/2.

presentation page. The coordinate space in which a picture is assembled for display.

presentation space (PS). Contains the device-independent definition of a picture.

primary window. The window in which the main dialog between the user and the application takes place. In a multiprogramming environment, each application starts in its own primary window. The primary window remains for the duration of the application, although the panel displayed will change as the user's dialog moves forward. See also *secondary window*.

primitive. See *graphic primitive*.

primitive attribute. A specifiable characteristic of a graphic primitive. See *graphics attributes*.

print job. The result of sending a document or picture to be printed.

Print Manager. In PM, the part of the spooler that manages the spooling process. It also allows users to view print queues and to manipulate print jobs.

privilege level. A protection level imposed by the hardware architecture of the IBM personal computer. There are four privilege levels (number 0 through 3). Only certain types of programs are allowed to execute at each privilege level. See also *IOPL code segment.*

procedure call. In programming languages, a language construct for invoking execution of a procedure.

process. An instance of an executing application and the resources it is using.

program details. Information about a program that is specified in the *Program Manager* window and is used when the program is started.

program group. In PM, several programs that can be acted upon as a single entity.

program name. The full file specification of a program. Contrast with *program title.*

program title. The name of a program as it is listed in the *Program Manager* window. Contrast with *program name.*

prompt. A displayed symbol or message that requests input from the user or gives operational information. The user must respond to the prompt in order to proceed.

protocol. A set of semantic and syntactic rules that determines the behavior o functional units in achieving communication.

pseudocode. An artificial language used to describe computer program algorithms without using the syntax of any particular programming language.

pull-down. An *action bar* extension that displays a list of choices available for a selected action bar choice. After users select an action bar choice, the pull-down appears with the list of choices. Additional *pop-up windows* may appear from pull-down choices to further extend the actions available to users.

push. To add an item to a last-in-first-out stack of items. Contrast with *pop.*

pushbutton. A control window, shaped like a round-cornered rectangle and containing text, that invokes an immediate action, such as 'enter' or 'cancel'.

Q

queue. A linked list of elements waiting to be processed. For example, a queue may be a list of print jobs waiting to be printed.

queued device context. A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also *device context.*

R

radio button. A control window, shaped like a round button on the screen, that can be in a checked or unchecked state. It is used to select a single item from list. Contrast with *check box.*

RAS. Reliability, availability, and serviceability.

raster. (1) In computer graphics, a predetermined pattern of lines that provides uniform coverage of a display space. (2) The coordinate grid that divides the display area of a display device.

read-only file. A file that may be read from but not written to.

realize. To cause the system to ensure, wherever possible, that the physical color table of a device is set to the closest possible match in the logical color table.

recursive routine. A routine that can call itself or be called by another routine called by the recursive routine.

reentrant. The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

reference phrase. A word or phrase that is emphasized in a device-dependent manner to inform the user that additional information for the word or phrase is available.

reference phrase help. Provides help information for a selectable word or phrase.

refresh. To update a window, with changed information, to its current status.

region. A clipping boundary in device space.

register. A storage device having a specified storage capacity such as a bit, byte, or computer word, and usually intended for a special purpose.

remote file system. A file-system driver that gains access to a remote system without a block device driver.

resource. The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts, templates, accelerators, and mnemonics; the definitions are held in a resource file.

resource file. A file containing information used in the definition of a window. Definitions can be of fonts, templates, accelerators, and mnemonics.

restore. To return a window to its original size or position following a sizing or moving action.

retained graphics. Graphic primitives that are remembered by the Presentation Interface after they have been drawn. Contrast with *nonretained graphics.*

return code. (1) A code used to influence the execution of succeeding instructions. (2) A value returned to a program to indicate the results of an operation requested by that program.

reverse video. A form of alphanumeric highlighting for a character, field, or cursor, in which its color is

exchanged with that of its background. For example, changing a red character on a black background to a black character on a red background.

RGB. Red-green-blue. For example, "RGB display".

roman. Relating to a type style with upright characters.

root segment. In a hierarchical database, the highest segment in the tree structure.

run time. (1) Any instant at which a program is being executed. (2) The time during which an instruction in an instruction register is decoded and performed.

S

SAA. Systems Application Architecture.

scheduler. A computer program designed to perform functions such as scheduling, initiation, and termination of jobs.

screen. The physical surface of a work station or terminal upon which information is presented to users.

screen device context. A logical description of a data destination that is a particular window on the screen. See also *device context*.

SCREEN$. Character-device name reserved for the display screen.

scroll bar. A control window, horizontally or vertically aligned, that allows the user to scroll additional data into an associated panel area.

scrollable entry field. An entry field larger than the visible field.

scrollable selection field. A selection field that contains more choices than are visible.

scrolling. Moving a display image vertically or horizontally in a manner such that new data appears at one edge, as existing data disappears at the opposite edge.

secondary window. A type of window associated with the primary window in a dialog. A secondary window begins a secondary and parallel dialog that runs at the same time as the primary dialog.

sector. An addressable subdivision of a track used to record one block of program code or data on a disk or diskette.

segment. See *graphics segment*.

segment attributes. Attributes that apply to the segment as an entity, as opposed to the individual primitives within the segment. For example, the visibility or detectability of a segment.

segment chain. All segments in a graphics presentation space that are defined with the 'chained' attribute. Synonym for *picture chain*.

segment priority. The order in which segments are drawn.

segment store. An area in a normal graphics presentation space where retained graphics segments are stored.

select. To mark or choose an item. Note that *select* means to mark or type in a choice on the screen; *enter* means to send all selected choices to the computer for processing.

select button. The button on a pointing device, such as a mouse, that is pressed to select a menu choice. Also known as button 1.

selection cursor. A type of cursor used to indicate the choice or entry field users want to interact with. It is represented by highlighting the item that it is currently positioned on.

selection field. A field containing a list of choices from which the user can select one or more.

semaphore. An object used by multi-threaded applications for signalling purposes and for controlling access to serially reusable resources.

separator. See *panel body area separator*.

serial dialog box. See *modal dialog box*.

serialization. The consecutive ordering of items.

serialize. To ensure that one or more events occur in a specified sequence.

serially reusable resource (SRR). A logical resource or object that can be accessed by only one task at a time.

session. A routing mechanism for user interaction via the console; a complete environment that determines how an application runs and how users interact with the application. OS/2 can manage more than one session at a time, and more than one process can run in a session. Each session has its own set of environment variables that determine where OS/2 looks for dynamic-link libraries and other important files.

shadow box. The area on the screen that follows mouse movements and shows what shape the window will take if the mouse button is released.

shared data. Data that is used by two or more programs.

shared memory. Memory that is used by two or more programs.

shear. The tilt of graphics text when each character leans to the left or right while retaining a horizontal baseline.

shell. (1) A software interface between a user and the operating system of a computer. Shell programs interpret commands and user interactions on devices such as keyboards, pointing devices, and touch-sensitive screens, and communicate them to the operating system. (2) Software that allows a kernel program to run under different operating-system environments.

Shutdown. The procedure required before the computer is switched off to ensure that data is not lost.

sibling processes. Child processes that have the same parent process.

sibling windows. Child windows that have the same parent window.

slider box. An area on the scroll bar that indicates the size and position of the visible information in a panel area in relation to the information available. Also known as *thumb mark*.

source file. A file that contains source statements for items such as high-level language programs and data description specifications.

source statement. A statement written in a programming language.

specific dynamic-link module. A dynamic-link module created for the exclusive use of an application.

spline. A sequence of one or more Bézier curves.

spooler. A program that intercepts the data going to printer devices and writes it to disk. The data is printed or plotted when it is complete, and the required device is available. The spooler prevents output from different sources from being intermixed.

stack. A list constructed and maintained so that the next data element to be retrieved is the most recently stored. This method is characterized as last-in-first-out (LIFO).

standard window. A collection of window elements that form a panel. The standard window can include one or more of the following window elements: sizing borders, system menu icon, title bar, maximize/minimize/restore icons, action bar and pull-downs, scroll bars, and client area.

static control. The means by which the application presents descriptive information (for example, headings and descriptors) to the user. The user cannot change this information.

static storage. (1) A read/write storage unit in which data is retained in the absence of control signals. Static storage may use dynamic addressing or sensing circuits. (2) Storage other than *dynamic storage*.

style. See *window style*.

suballocation. The allocation of a part of one extent for occupancy by elements of a component other than the one occupying the remainder of the extent.

subdirectory. In an IBM personal computer, a file referred to in a root directory that contains the names of other files stored on the diskette or fixed disk.

swapping. (1) A process that interchanges the contents of an area of real storage with the contents of an area in auxiliary storage. (2) In a system with virtual storage, a paging technique that writes the active pages of a job to auxiliary storage and reads pages of another job from auxiliary storage into real storage. (3) The process of temporarily removing an active job from main storage, saving it on disk, and processing another job in the area of main storage formerly occupied by the first job.

switch. (1) An action that moves the input focus from one area to another. This can be within the same window or from one window to another. (2) In a computer program, a conditional instruction and an indicator to be interrogated by that instruction. (3) A device or programming technique for making a selection, for example, a toggle, a conditional jump.

switch list. See *Task List*.

symbolic identifier. A text string that equates to an integer value in an include file, that is used to identify a programming object.

synchronous. Pertaining to events or operations that are predictable or occur at the same time. See also *asynchronous*.

System Menu. In PM, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

system queue. This is the master queue for all pointer device or keyboard events.

Systems Application Architecture (SAA). A formal set of rules that enables applications to be run without modification in different computer environments.

T

tag. One or more characters attached to a set of data that defines the formatting or other characteristics of the set, including its definition.

Task List. In PM, the list of programs that are active. The list can be used to switch to a program and to stop programs.

template. An ASCII-text definition of an action bar and pull-down menu, held in a resource file, or as a data structure in program memory.

text. Characters or symbols.

text cursor. A symbol displayed in an entry field that indicates where typed input will appear.

text window. Also known as the VIO window.

text-windowed application. The environment in which the operating system performs advanced&hyphn.video input and output operations.

thread. A unit of execution within a process. It uses the resources of the process.

thumb mark. The portion of the scroll bar that describes the range and properties of the data that is currently visible in a window. Also known as a *slider box*.

tilde. A mark used to denote the character that is to be used as a mnemonic when selecting text items within a menu.

time slice. (1) An interval of time on the processing unit allocated for use in performing a task. After the interval has expired, processing-unit time is allocated to another task, so a task cannot monopolize processing-unit time beyond a fixed limit. (2) In systems with time sharing, a segment of time allocated to a terminal job.

title bar. The area at the top of a window that contains the window title. The title bar is highlighted when that window has the input focus. Contrast with *panel title*.

transaction. An exchange between a workstation and another device that accomplishes a particular action or result.

transform. (1) The action of modifying a picture by scaling, shearing, reflecting, rotating, or translating. (2) The object that performs or defines such a modification; also referred to as a *transformation*.

Tree. In PM, the window in the *File Manager* that shows the organization of drives and directories.

truncate. (1) To end a computational process in accordance with some rule. (2) To remove the beginning or ending elements of a string. (3) To drop data that cannot be printed or displayed in the line width specified or available. (4) To shorten a field or statement to a specified length.

U

unnamed pipe. A circular buffer, created in memory, used by related processes to communicate with one another. Contrast with *named pipe*.

update region. A system-provided area of dynamic storage containing one or more (not necessarily contiguous) rectangular areas of a window, that are visually invalid or incorrect, and therefore in need of repainting.

user interface. Hardware, software, or both that allows a user to interact with and perform operations on a system, program, or device.

User Shell. A component of OS/2 that uses a graphics-based, windowed interface to allow the user to manage applications and files installed and running under OS/2.

utility program. (1) A computer program in general support of computer processes; for example, a diagnostic program, a trace program, a sort program. (2) A program designed to perform an everyday task such as copying data from one storage device to another.

V

vector font. A set of symbols, each of which is created as a series of lines and curves. Synonymous with *outline font*. Contrast with *image font*.

VGA. Video graphics array.

viewing pipeline. The series of transformations applied to a graphic object to map the object to the device on which it is to be presented.

viewing window. Clipping boundary that defines the visible part of model space.

VIO. Video Input/Output.

virtual memory (VM). Addressable space that is apparent to the user as the processor storage space, but not having a fixed physical location.

virtual storage. Synonymous with *virtual memory*.

visible region. A window's presentation space, clipped to the boundary of the window and the boundaries of any overlying window.

volume. (1) A file-system driver that uses a block device driver for input and output operations to a local or remote device. (2) A portion of data, together with its data carrier, that can be handled conveniently as a unit.

W

wild-card character. The global file-name characters asterisk (*) and question mark (?).

window. A rectangular area of the screen with visible boundaries within which information is displayed. A window can be smaller than or the same size as the screen. Windows can appear to overlap on the screen.

window class. The grouping of windows whose processing needs conform to the services provided by one window procedure.

window coordinates. The means by which a window position or size is defined; measured in device units, or *pels*.

window procedure. Code that is activated in response to a message. The procedure controls the appearance and behavior of its associated windows.

window rectangle. The means by which the size and position of a window is described in relation to the desktop window.

window style. The set of properties that influence how events related to a particular window will be processed.

workstation. A display screen together with attachments such as a keyboard, a local copy device, or a tablet.

world coordinates. Application-convenient coordinates used for drawing graphics.

world-coordinate space. Coordinate space in which graphics are defined before transformations are applied.

WYSIWYG. What You See Is What You Get. A capability that enables text to be displayed on a screen in the same way it will be formatted on a printer.

Z

z-order. The order in which sibling windows are presented. The topmost sibling window obscures any portion of the siblings that it overlaps; the same effect occurs down through the order of lower sibling windows.

zooming. In graphics applications, the process of increasing or decreasing the size of picture.

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

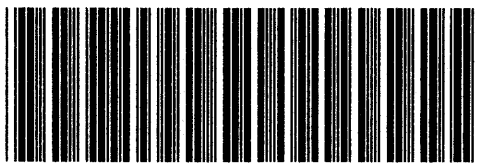
W

X

International Business
Machines Corporation

Printed in the
United States of America

10G6265



S10G-6265-00

P10G6265