Contents

Keyboard and Mouse Input

7/6/2022 • 7 minutes to read • Edit Online

Overview of the Keyboard and Mouse Input technology.

The Keyboard and Mouse Input technology is not associated with any headers.

For programming guidance for this technology, see:

- Keyboard and Mouse Input

Enumerations

TOOLTIP_DISMISS_FLAGS

Functions

_TrackMouseEvent Posts messages when the mouse pointer leaves a window or hovers over a window for a specified amount of time. This function calls TrackMouseEvent if it exists, otherwise it emulates it.
ActivateKeyboardLayout Sets the input locale identifier (formerly called the keyboard layout handle) for the calling thread or the current process. The input locale identifier specifies a locale as well as the physical layout of the keyboard.
BlockInput Blocks keyboard and mouse input events from reaching applications.
DefRawInputProc Verifies that the size of the RAWINPUTHEADER structure is correct.
DragDetect Captures the mouse and tracks its movement until the user releases the left button, presses the ESC key, or moves the mouse outside the drag rectangle around the specified point.
EnableWindow Enables or disables mouse and keyboard input to the specified window or control. When input is disabled, the window does not receive input such as mouse clicks and key presses. When input is enabled, the window receives all input.
GET_APPCOMMAND_LPARAM Retrieves the application command from the specified LPARAM value.

GET_DEVICE_LPARAM

Retrieves the input device type from the specified LPARAM value.

GET_FLAGS_LPARAM

Retrieves the state of certain virtual keys from the specified LPARAM value.

GET_KEYSTATE_LPARAM

Retrieves the state of certain virtual keys from the specified LPARAM value.

GET_KEYSTATE_WPARAM

Retrieves the state of certain virtual keys from the specified WPARAM value.

GET_NCHITTEST_WPARAM

Retrieves the hit-test value from the specified WPARAM value.

GET_RAWINPUT_CODE_WPARAM

Retrieves the input code from wParam in WM_INPUT.

GET_WHEEL_DELTA_WPARAM

Retrieves the wheel-delta value from the specified WPARAM value.

GET_XBUTTON_WPARAM

Retrieves the state of certain buttons from the specified WPARAM value.

GetActiveWindow

Retrieves the window handle to the active window attached to the calling thread's message queue.

GetAsyncKeyState

Determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to GetAsyncKeyState.

GetCapture

Retrieves a handle to the window (if any) that has captured the mouse. Only one window at a time can capture the mouse; this window receives mouse input whether or not the cursor is within its borders.

GetDoubleClickTime

Retrieves the current double-click time for the mouse.

GetFocus

Retrieves the handle to the window that has the keyboard focus, if the window is attached to the calling thread's message queue.

GetKBCodePage	Retrieves the current code page.
GetKeyboardLayout	Retrieves the active input locale identifier (formerly called the keyboard layout).
GetKeyboardLayoutList	Retrieves the input locale identifiers (formerly called keyboard layout handles) corresponding to the current set of input locales in the system. The function copies the identifiers to the specified buffer.
GetKeyboardLayoutNameA	Retrieves the name of the active input locale identifier (formerly called the keyboard layout) for the system.
GetKeyboardLayoutNameW	Retrieves the name of the active input locale identifier (formerly called the keyboard layout) for the system.
GetKeyboardState	Copies the status of the 256 virtual keys to the specified buffer.
GetKeyboardType	Retrieves information about the current keyboard.
GetKeyNameTextA	Retrieves a string that represents the name of a key.
GetKeyNameTextW	Retrieves a string that represents the name of a key.
GetKeyState	Retrieves the status of the specified virtual key. The status specifies whether the key is up, down, or toggled (on, off�alternating each time the key is pressed).
GetLastInputInfo	Retrieves the time of the last input event.
GetMouseMovePointsEx	Retrieves a history of up to 64 previous coordinates of the mouse or pen.
GetRawInputBuffer	Performs a buffered read of the raw input data.

GetRawInputData	Retrieves the raw input from the specified device.
GetRawInputDeviceInfoA	Retrieves information about the raw input device.
GetRawInputDeviceInfoW	Retrieves information about the raw input device.
GetRawInputDeviceList	Enumerates the raw input devices attached to the system.
GetRegisteredRawInputDevices	Retrieves the information about the raw input devices for the current application.
IsWindowEnabled	Determines whether the specified window is enabled for mouse and keyboard input.
keybd_event	Synthesizes a keystroke.
LoadKeyboardLayoutA	Loads a new input locale identifier (formerly called the keyboard layout) into the system.
LoadKeyboardLayoutW	Loads a new input locale identifier (formerly called the keyboard layout) into the system.
MapVirtualKeyA	Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code.
MapVirtualKeyExA	Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code. The function translates the codes using the input language and an input locale identifier.
MapVirtualKeyExW	Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code. The function translates the codes using the input language and an input locale identifier.
MapVirtualKeyW	Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code.

mouse_event	The mouse_event function synthesizes mouse motion and button clicks.
NEXTRAWINPUTBLOCK	Retrieves the location of the next structure in an array of RAWINPUT structures.
OemKeyScan	Maps OEMASCII codes 0 through 0x0FF into the OEM scan codes and shift states. The function provides information that allows a program to send OEM text to another program by simulating keyboard input.
RegisterForTooltipDismissNotification	Lets apps or UI frameworks register and unregister windows to receive notification to dismiss their tooltip windows.
RegisterHotKey	Defines a system-wide hot key.
RegisterRawInputDevices	Registers the devices that supply the raw input data.
ReleaseCapture	Releases the mouse capture from a window in the current thread and restores normal mouse input processing.
SendInput	Synthesizes keystrokes, mouse motions, and button clicks.
SetActiveWindow	Activates a window. The window must be attached to the calling thread's message queue.
SetCapture	Sets the mouse capture to the specified window belonging to the current thread.
SetDoubleClickTime	Sets the double-click time for the mouse.
SetFocus	Sets the keyboard focus to the specified window. The window must be attached to the calling thread's message queue.
SetKeyboardState	Copies an array of keyboard key states into the calling thread's keyboard input-state table. This is the same table accessed by the GetKeyboardState and GetKeyState functions. Changes made to this table do not affect keyboard input to any other thread.

SwapMouseButton	Reverses or restores the meaning of the left and right mouse buttons.
ToAscii	Translates the specified virtual-key code and keyboard state to the corresponding character or characters.
ToAsciiEx	Translates the specified virtual-key code and keyboard state to the corresponding character or characters. The function translates the code using the input language and physical keyboard layout identified by the input locale identifier.
ToUnicode	Translates the specified virtual-key code and keyboard state to the corresponding Unicode character or characters.
ToUnicodeEx	Translates the specified virtual-key code and keyboard state to the corresponding Unicode character or characters.
TrackMouseEvent	Posts messages when the mouse pointer leaves a window or hovers over a window for a specified amount of time.
UnloadKeyboardLayout	Unloads an input locale identifier (formerly called a keyboard layout).
UnregisterHotKey	Frees a hot key previously registered by the calling thread.
VkKeyScanA	Translates a character to the corresponding virtual-key code and shift state for the current keyboard.
VkKeyScanExA	Translates a character to the corresponding virtual-key code and shift state. The function translates the character using the input language and physical keyboard layout identified by the input locale identifier.
VkKeyScanExW	Translates a character to the corresponding virtual-key code and shift state. The function translates the character using the input language and physical keyboard layout identified by the input locale identifier.
VkKeyScanW	Translates a character to the corresponding virtual-key code and shift state for the current keyboard.

Structures

HARDWAREINPUT	Contains information about a simulated message generated by an input device other than a keyboard or mouse.
INPUT	Used by SendInput to store information for synthesizing input events such as keystrokes, mouse movement, and mouse clicks.
KEYBDINPUT	Contains information about a simulated keyboard event.
LASTINPUTINFO	Contains the time of the last input.
MOUSEINPUT	Contains information about a simulated mouse event.
MOUSEMOVEPOINT	Contains information about the mouse's location in screen coordinates.
RAWHID	Describes the format of the raw input from a Human Interface Device (HID).
RAWINPUT	Contains the raw input from a device.
RAWINPUTDEVICE	Defines information for the raw input devices.
RAWINPUTDEVICELIST	Contains information about a raw input device.
RAWINPUTHEADER	Contains the header information that is part of the raw input data.
RAWKEYBOARD	Contains information about the state of the keyboard.
RAWMOUSE	Contains information about the state of the mouse.

RID_DEVICE_INFO	Defines the raw input data coming from any device.
RID_DEVICE_INFO_HID	Defines the raw input data coming from the specified Human Interface Device (HID).
RID_DEVICE_INFO_KEYBOARD	Defines the raw input data coming from the specified keyboard.
RID_DEVICE_INFO_MOUSE	Defines the raw input data coming from the specified mouse.
TRACKMOUSEEVENT	Used by the TrackMouseEvent function to track when the mouse pointer leaves a window or hovers over a window for a specified amount of time.

commctrl.h header

7/6/2022 • 65 minutes to read • Edit Online

This header is used by multiple technologies. For more information, see:

- Keyboard and Mouse Input
- The Windows Shell
- Windows Controls

commctrl.h contains the following programming interfaces:

Functions

_TrackMouseEvent Posts messages when the mouse pointer leaves a window or hovers over a window for a specified amount of time. This function calls TrackMouseEvent if it exists, otherwise it emulates it.
Animate_Close Closes an AVI clip. You can use this macro or send the ACM_OPEN message explicitly, passing in NULL parameters.
Animate_Create Creates an animation control. Animate_Create calls the CreateWindow function to create the animation control.
Animate_IsPlaying Checks to see if an Audio-Video Interleaved (AVI) clip is playing. You can use this macro or send an ACM_ISPLAYING message.
Animate_Open Opens an AVI clip and displays its first frame in an animation control. You can use this macro or send the ACM_OPEN message explicitly.
Animate_OpenEx Opens an AVI clip from a resource in a specified module and displays its first frame in an animation control. You can use this macro or send the ACM_OPEN message explicitly.
Animate_Play Plays an AVI clip in an animation control. The control plays the clip in the background while the thread continues executing. You can use this macro or send the ACM_PLAY message explicitly.
Animate_Seek Directs an animation control to display a particular frame of an AVI clip. The control displays the clip in the background while the thread continues executing. You can use this macro or send the ACM_PLAY message explicitly.

Animate_Stop	Stops playing an AVI clip in an animation control. You can use this macro or send the ACM_STOP message explicitly.
Button_GetIdealSize	Gets the size of the button that best fits the text and image, if an image list is present. You can use this macro or send the BCM_GETIDEALSIZE message explicitly.
Button_GetImageList	Gets the BUTTON_IMAGELIST structure that describes the image list that is set for a button control. You can use this macro or send the BCM_GETIMAGELIST message explicitly.
Button_GetNote	Gets the text of the note associated with a command link button. You can use this macro or send the BCM_GETNOTE message explicitly.
Button_GetNoteLength	Gets the length of the note text that may be displayed in the description for a command link. Use this macro or send the BCM_GETNOTELENGTH message explicitly.
Button_GetSplitInfo	Gets information for a specified split button control. Use this macro or send the BCM_GETSPLITINFO message explicitly.
Button_GetTextMargin	Gets the margins used to draw text in a button control. You can use this macro or send the BCM_GETTEXTMARGIN message explicitly.
Button_SetDropDownState	Sets the drop down state for a specified button with style of BS_SPLITBUTTON. Use this macro or send the BCM_SETDROPDOWNSTATE message explicitly.
Button_SetElevationRequiredState	Sets the elevation required state for a specified button or command link to display an elevated icon. Use this macro or send the BCM_SETSHIELD message explicitly.
Button_SetImageList	Assigns an image list to a button control. You can use this macro or send the BCM_SETIMAGELIST message explicitly.
Button_SetNote	Sets the text of the note associated with a specified command link button. You can use this macro or send the BCM_SETNOTE message explicitly.
Button_SetSplitInfo	Sets information for a specified split button control. Use this macro or send the BCM_SETSPLITINFO message explicitly.

Button_SetTextMargin	Sets the margins for drawing text in a button control. You can use this macro or send the BCM_SETTEXTMARGIN message explicitly.
ComboBox_GetCueBannerText	Gets the cue banner text displayed in the edit control of a combo box. Use this macro or send the CB_GETCUEBANNER message explicitly.
ComboBox_GetMinVisible	Gets the minimum number of visible items in the drop-down list of a combo box.
ComboBox_SetCueBannerText	Sets the cue banner text that is displayed for the edit control of a combo box.
ComboBox_SetMinVisible	Sets the minimum number of visible items in the drop-down list of a combo box.
CreateMappedBitmap	Creates a bitmap for use in a toolbar.
CreateStatusWindowA	Creates a status window, which is typically used to display the status of an application.
CreateStatusWindowW	Creates a status window, which is typically used to display the status of an application.
CreateToolbarEx	Creates a toolbar window and adds the specified buttons to the toolbar.
CreateUpDownControl	Creates an up-down control. Note:_This function is obsolete. It is a 16 bit function and cannot handle 32 bit values for range and position.
DateTime_CloseMonthCal	Closes the date and time picker (DTP) control. Use this macro or send the DTM_CLOSEMONTHCAL message explicitly.
DateTime_GetDateTimePickerInfo	Gets information for a specified date and time picker (DTP) control.
DateTime_GetIdealSize	Gets the size needed to display the control without clipping. Use this macro or send the DTM_GETIDEALSIZE message explicitly.

DateTime_GetMonthCal

Gets the handle to a date and time picker's (DTP) child month calendar control. You can use this macro or send the DTM_GETMONTHCAL message explicitly.

DateTime_GetMonthCalColor

Gets the color for a given portion of the month calendar within a date and time picker (DTP) control. You can use this macro or send the DTM_GETMCCOLOR message explicitly.

DateTime_GetMonthCalFont

Gets the font that the date and time picker (DTP) control's child month calendar control is currently using. You can use this macro or send the DTM_GETMCFONT message explicitly.

DateTime_GetMonthCalStyle

Gets the style of a specified date and time picker (DTP) control. Use this macro or send the DTM_GETMCSTYLE message explicitly.

DateTime_GetRange

Gets the current minimum and maximum allowable system times for a date and time picker (DTP) control. You can use this macro, or send the DTM_GETRANGE message explicitly.

DateTime_GetSystemtime

Gets the currently selected time from a date and time picker (DTP) control and places it in a specified SYSTEMTIME structure. You can use this macro, or send the DTM_GETSYSTEMTIME message explicitly.

DateTime_SetFormat

Sets the display of a date and time picker (DTP) control based on a given format string. You can use this macro or send the DTM_SETFORMAT message explicitly.

DateTime_SetMonthCalColor

Sets the color for a given portion of the month calendar within a date and time picker (DTP) control. You can use this macro or send the DTM_SETMCCOLOR message explicitly.

DateTime_SetMonthCalFont

Sets the font to be used by the date and time picker (DTP) control's child month calendar control. You can use this macro or explicitly send the DTM_SETMCFONT message.

DateTime_SetMonthCalStyle

Sets the style for a specified date and time picker (DTP) control. Use this macro or send the DTM_SETMCSTYLE message explicitly.

DateTime_SetRange

Sets the minimum and maximum allowable system times for a date and time picker (DTP) control. You can use this macro or send the DTM_SETRANGE message explicitly.

DateTime_SetSystemtime

Sets a date and time picker (DTP) control to a given date and time. You can use this macro or send the DTM_SETSYSTEMTIME message explicitly.

DefSubclassProc

Calls the next handler in a window's subclass chain. The last handler in the subclass chain calls the original window procedure for the window.

DrawInsert

Draws the insert icon in the parent window of the specified drag list box.

DrawShadowText

Draws text that has a shadow.

DrawStatusTextA

The DrawStatusText function draws the specified text in the style of a status window with borders.

DrawStatusTextW

The DrawStatusText function draws the specified text in the style of a status window with borders.

Edit_EnableSearchWeb

Enables or disables the "Search with Bing..." context menu item in edit controls. You can use this macro or send the EM_ENABLESEARCHWEB message explicitly.

Edit_GetCaretIndex

Gets the character index of the caret location for a given edit control. You can use this macro or send the EM_GETCARETINDEX message explicitly.

Edit_GetCueBannerText

Gets the text that is displayed as a textual cue, or tip, in an edit control. You can use this macro or send the EM_GETCUEBANNER message explicitly.

Edit_GetEndOfLine

Gets the end of line character used for the content of the edit control. You can use this macro or send the EM_GETENDOFLINE message explicitly.

Edit_GetExtendedStyle

Gets the extended styles that are currently in use for a given edit control. You can use this macro or send the EM_GETEXTENDEDSTYLE message explicitly.

Edit_GetFileLine

Gets the text of the specified file (or logical) line (text wrap delimiters are ignored). You can use this macro or send the EM_GETFILELINE message explicitly.

Edit_GetFileLineCount	Gets the number of file (or logical) lines (text wrap delimiters are ignored). You can use this macro or send the EM_GETFILELINECOUNT message explicitly.
Edit_GetFileLineFromChar	Gets the index of the file (or logical) line of text that includes the specified character index (text wrap delimiters are ignored). You can use this macro or send the EM_FILELINEFROMCHAR message explicitly.
Edit_GetFileLineIndex	Gets the index of the file (or logical) line of text based on the specified visible line. You can use this macro or send the EM_FILELINEINDEX message explicitly.
Edit_GetFileLineLength	Gets the length of the file (or logical) line of text from the specified character index (text wrap delimiters are ignored). You can use this macro or send the EM_FILELINELENGTH message explicitly.
Edit_GetHilite	This macro is not implemented.
Edit_GetZoom	Gets the current zoom ratio of an edit control (the zoom ratio is always between 1/64 and 64). You can use this macro or send the EM_GETZOOM message explicitly.
Edit_HideBalloonTip	Hides any balloon tip associated with an edit control. You can use this macro or send the EM_HIDEBALLOONTIP message explicitly.
Edit_NoSetFocus	Prevents a single-line edit control from receiving keyboard focus. You can use this macro or send the EM_NOSETFOCUS message explicitly.
Edit_SearchWeb	Invokes the "Search with Bing..." context menu item in edit controls. You can use this macro or send the EM_SEARCHWEB message explicitly.
Edit_SetCaretIndex	Sets the character index at which to locate the caret. You can use this macro or send the EM_SETCARETINDEX message explicitly.
Edit_SetCueBannerText	Sets the text that is displayed as the textual cue, or tip, for an edit control. You can use this macro or send the EM_SETCUEBANNER message explicitly.

Edit_SetCueBannerTextFocused Sets the text that is displayed as the textual cue, or tip, for an edit control. You can use this macro or send the EM_SETCUEBANNER message explicitly.	

Edit_SetCueBannerTextFocused

Sets the text that is displayed as the textual cue, or tip, for an edit control. You can use this macro or send the EM_SETCUEBANNER message explicitly.

Edit_SetEndOfLine

Sets the end of line character used for the content of the edit control. You can use this macro or send the EM_SETENDOFLINE message explicitly.

Edit_SetExtendedStyle

Sets extended styles for edit controls using the style mask. You can use this macro or send the EM_SETEXTENDEDSTYLE message explicitly.

Edit_SetHilite

This macro is not implemented.

Edit_SetZoom

Sets the current zoom ratio of an edit control (the zoom ratio is always between 1/64 and 64). You can use this macro or send the EM_SETZOOM message explicitly.

Edit_ShowBalloonTip

Displays a balloon tip associated with an edit control. You can use this macro or send the EM_SHOWBALLOONTIP message explicitly.

Edit_TakeFocus

Forces a single-line edit control to receive keyboard focus. You can use this macro or send the EM_TAKEFOCUS message explicitly.

FIRST_IPADDRESS

Extracts the field 0 value from a packed IP address retrieved with the IPM_GETADDRESS message.

FlatSB_EnableScrollBar

Enables or disables one or both flat scroll bar direction buttons. If flat scroll bars are not initialized for the window, this function calls the standard EnableScrollBar function.

FlatSB_GetScrollInfo

Gets the information for a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard GetScrollInfo function.

FlatSB_GetScrollPos

Gets the thumb position in a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard GetScrollPos function.

FlatSB_GetScrollProp	Gets the properties for a flat scroll bar. This function can also be used to determine if InitializeFlatSB has been called for this window.
FlatSB_GetScrollPropPtr	Gets the properties for a flat scroll bar.
FlatSB_GetScrollRange	Gets the scroll range for a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard GetScrollRange function.
FlatSB_SetScrollInfo	Sets the information for a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard SetScrollInfo function.
FlatSB_SetScrollPos	Sets the current position of the thumb in a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard SetScrollPos function.
FlatSB_SetScrollProp	Sets the properties for a flat scroll bar.
FlatSB_SetScrollRange	Sets the scroll range of a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard SetScrollRange function.
FlatSB_ShowScrollBar	Shows or hides a flat scroll bar. If flat scroll bars are not initialized for the window, this function calls the standard ShowScrollBar function.
FORWARD_WM_NOTIFY	Sends or posts the WM_NOTIFY message.
FOURTH_IPADDRESS	Extracts the field 3 value from a packed IP address retrieved with the IPM_GETADDRESS message.
GetEffectiveClientRect	Calculates the dimensions of a rectangle in the client area that contains all the specified controls.
GetMUILanguage	Gets the language currently in use by the common controls for a particular process.

GetWindowSubclass	Retrieves the reference data for the specified window subclass callback.
HANDLE_WM_NOTIFY	Calls a function that processes the WM_NOTIFY message.
Header_ClearAllFilters	Clears all of the filters for a given header control. You can use this macro or send the HDM_CLEARFILTER message explicitly.
Header_ClearFilter	Clears the filter for a given header control. You can use this macro or send the HDM_CLEARFILTER message explicitly.
Header_CreateDragImage	Creates a transparent version of an item image within an existing header control. You can use this macro or send the HDM_CREATEDRAGIMAGE message explicitly.
Header_DeleteItem	Deletes an item from a header control. You can use this macro or send the HDM_DELETEITEM message explicitly.
Header_EditFilter	Moves the input focus to the edit box when a filter button has the focus.
Header_GetBitmapMargin	Gets the width of the margin (in pixels) of a bitmap in an existing header control. You can use this macro or send the HDM_GETBITMAPMARGIN message explicitly.
Header_GetFocusedItem	Gets the item in a header control that has the focus. Use this macro or send the HDM_GETFOCUSEDITEM message explicitly.
Header_GetImageList	Gets the handle to the image list that has been set for an existing header control. You can use this macro or send the HDM_GETIMAGELIST message explicitly.
Header_GetItem	Gets information about an item in a header control. You can use this macro or send the HDM_GETITEM message explicitly.
Header_GetItemCount	Gets a count of the items in a header control. You can use this macro or send the HDM_GETITEMCOUNT message explicitly.
Header_GetItemDropDownRect	Gets the coordinates of the drop-down button for a specified item in a header control. The header control must be of type HDF_SPLITBUTTON. Use this macro or send the HDM_GETITEMDROPDOWNRECT message explicitly.

Header_GetItemRect	Gets the bounding rectangle for a given item in a header control. You can use this macro or send the HDM_GETITEMRECT message explicitly.
Header_GetOrderArray	Gets the current left-to-right order of items in a header control. You can use this macro or send the HDM_GETORDERARRAY message explicitly.
Header_GetOverflowRect	Gets the coordinates of the drop-down overflow area for a specified header control. The header control must be of type HDF_SPLITBUTTON. Use this macro or send the HDM_GETOVERFLOWRECT message explicitly.
Header_GetStateImageList	Gets the handle to the image list that has been set for an existing header control state.
Header_GetUnicodeFormat	Gets the Unicode character format flag for the control. You can use this macro or send the HDM_GETUNICODEFORMAT message explicitly.
Header_InsertItem	Inserts a new item into a header control. You can use this macro or send the HDM_INSERTITEM message explicitly.
Header_Layout	Retrieves the correct size and position of a header control within the parent window. You can use this macro or send the HDM_LAYOUT message explicitly.
Header_OrderToIndex	Retrieves an index value for an item based on its order in the header control. You can use this macro or send the HDM_ORDERTOINDEX message explicitly.
Header_SetBitmapMargin	Sets the width of the margin for a bitmap in an existing header control. You can use this macro or send the HDM_SETBITMAPMARGIN message explicitly.
Header_SetFilterChangeTimeout	Sets the timeout interval between the time a change takes place in the filter attributes and the posting of an HDN_FILTERCHANGE notification. You can use this macro or send the HDM_SETFILTERCHANGETIMEOUT message explicitly.
Header_SetFocusedItem	Sets the focus to a specified item in a header control. Use this macro or send the HDM_SETFOCUSEDITEM message explicitly.
Header_SetHotDivider	Changes the color of a divider between header items to indicate the destination of an external drag-and-drop operation. You can use this macro or send the HDM_SETHOTDIVIDER message explicitly.

Header_SetImageList	Assigns an image list to an existing header control. You can use this macro or send the HDM_SETIMAGELIST message explicitly.
Header_SetItem	Sets the attributes of the specified item in a header control. You can use this macro or send the HDM_SETITEM message explicitly.
Header_SetOrderArray	Sets the left-to-right order of header items. You can use this macro or send the HDM_SETORDERARRAY message explicitly.
Header_SetStateImageList	Assigns an image list to an existing header control state.
Header_SetUnicodeFormat	Sets the UNICODE character format flag for the control.
HIMAGELIST_QueryInterface	Retrieves a pointer to an IImageList or IImageList2 object that corresponds to the image list's HIMAGELIST handle.
ImageList_Add	Adds an image or images to an image list.
ImageList_AddIcon	Adds an icon or cursor to an image list. ImageList_AddIcon calls the ImageList_ReplaceIcon function.
ImageList_AddMasked	Adds an image or images to an image list, generating a mask from the specified bitmap.
ImageList_BeginDrag	Begins dragging an image.
ImageList_Copy	Copies images within a given image list.
ImageList_Create	Creates a new image list.
ImageList_Destroy	Destroys an image list.

ImageList_DragEnter	Displays the drag image at the specified position within the window.
ImageList_DragLeave	Unlocks the specified window and hides the drag image, allowing the window to be updated.
ImageList_DragMove	Moves the image that is being dragged during a drag-and-drop operation. This function is typically called in response to a WM_MOUSEMOVE message.
ImageList_DragShowNolock	Shows or hides the image being dragged.
ImageList_Draw	Draws an image list item in the specified device context.
ImageList_DrawEx	Draws an image list item in the specified device context. The function uses the specified drawing style and blends the image with the specified color.
ImageList_DrawIndirect	Draws an image list image based on an IMAGELISTDRAWPARAMS structure.
ImageList_Duplicate	Creates a duplicate of an existing image list.
ImageList_EndDrag	Ends a drag operation.
ImageList_ExtractIcon	Calls the ImageList_GetIcon function to create an icon or cursor based on an image and mask in an image list.
ImageList_GetBkColor	Retrieves the current background color for an image list.
ImageList_GetDragImage	Retrieves the temporary image list that is used for the drag image. The function also retrieves the current drag position and the offset of the drag image relative to the drag position.
ImageList_GetIcon	Creates an icon from an image and mask in an image list.

ImageList_GetIconSize	Retrieves the dimensions of images in an image list. All images in an image list have the same dimensions.
ImageList_GetImageCount	Retrieves the number of images in an image list.
ImageList_GetImageInfo	Retrieves information about an image.
ImageList_LoadBitmap	Calls the ImageList_LoadImage function to create an image list from the specified bitmap resource.
ImageList_LoadImageA	Creates an image list from the specified bitmap.
ImageList_LoadImageW	Creates an image list from the specified bitmap.
ImageList_Merge	Creates a new image by combining two existing images. The function also creates a new image list in which to store the image.
ImageList_Read	Reads an image list from a stream.
ImageList_ReadEx	Reads an image list from a stream, and returns an IImageList interface to the image list.
ImageList_Remove	Removes an image from an image list.
ImageList_RemoveAll	Calls the ImageList_Remove function to remove all of the images from an image list.
ImageList_Replace	Replaces an image in an image list with a new image.
ImageList_ReplaceIcon	Replaces an image with an icon or cursor.

ImageList_SetBkColor

Sets the background color for an image list. This function only works if you add an icon or use ImageList_AddMasked with a black and white bitmap. Without a mask, the entire image is drawn; hence the background color is not visible.

ImageList_SetDragCursorImage

Creates a new drag image by combining the specified image (typically a mouse cursor image) with the current drag image.

ImageList_SetIconSize

Sets the dimensions of images in an image list and removes all images from the list.

ImageList_SetImageCount

Resizes an existing image list.

ImageList_SetOverlayImage

Adds a specified image to the list of images to be used as overlay masks. An image list can have up to four overlay masks in version 4.70 and earlier and up to 15 in version 4.71. The function assigns an overlay mask index to the specified image.

ImageList_Write

Writes an image list to a stream.

ImageList_WriteEx

Writes an image list to a stream.

INDEXTOOVERLAYMASK

Prepares the index of an overlay mask so that the ImageList_Draw function can use it.

INDEXTOSTATEIMAGEMASK

Prepares the index of a state image so that a tree-view control or list-view control can use the index to retrieve the state image for an item.

InitCommonControls

Registers and initializes certain common control window classes. This function is obsolete. New applications should use the InitCommonControlsEx function.

InitCommonControlsEx

Ensures that the common control DLL (Comctl32.dll) is loaded, and registers specific common control classes from the DLL. An application must call this function before creating a common control.

InitializeFlatSB

Initializes flat scroll bars for a particular window.

InitMUILanguage

Enables an application to specify a language to be used with the common controls that is different from the system language.

LBItemFromPt	Retrieves the index of the item at the specified point in a list box.
ListView_ApproximateViewRect	Calculates the approximate width and height required to display a given number of items. You can use this macro or send the LVM_APPROXIMATEVIEWRECT message explicitly.
ListView_Arrange	Arranges items in icon view. You can use this macro or send the LVM_ARRANGE message explicitly.
ListView_CancelEditLabel	Cancels an item text editing operation. You can use this macro or send the LVM_CANCELEDITLABEL message explicitly.
ListView_CreateDragImage	Creates a drag image list for the specified item. You can use this macro or send the LVM_CREATEDRAGIMAGE message explicitly.
ListView_DeleteAllItems	Removes all items from a list-view control. You can use this macro or send the LVM_DELETEALLITEMS message explicitly.
ListView_DeleteColumn	Removes a column from a list-view control. You can use this macro or send the LVM_DELETECOLUMN message explicitly.
ListView_DeleteItem	Removes an item from a list-view control. You can use this macro or send the LVM_DELETEITEM message explicitly.
ListView_EditLabel	Begins in-place editing of the specified list-view item's text. The message implicitly selects and focuses the specified item. You can use this macro or send the LVM_EDITLABEL message explicitly.
ListView_EnableGroupView	Enables or disables whether the items in a list-view control display as a group. You can use this macro or send the LVM_ENABLEGROUPVIEW message explicitly.
ListView_EnsureVisible	Ensures that a list-view item is either entirely or partially visible, scrolling the list-view control if necessary. You can use this macro or send the LVM_ENSUREVISIBLE message explicitly.
ListView_FindItem	Searches for a list-view item with the specified characteristics. You can use this macro or send the LVM_FINDITEM message explicitly.

ListView_GetBkColor

Gets the background color of a list-view control. You can use this macro or send the LVM_GETBKCOLOR message explicitly.

ListView_GetBkImage

Gets the background image in a list-view control. You can use this macro or send the LVM_GETBKIMAGE message explicitly.

ListView_GetCallbackMask

Gets the callback mask for a list-view control. You can use this macro or send the LVM_GETCALLBACKMASK message explicitly.

ListView_GetCheckState

Determines if an item in a list-view control is selected. This should be used only for list-view controls that have the LVS_EX_CHECKBOXES style.

ListView_GetColumn

Gets the attributes of a list-view control's column. You can use this macro or send the LVM_GETCOLUMN message explicitly.

ListView_GetColumnOrderArray

Gets the current left-to-right order of columns in a list-view control. You can use this macro or send the LVM_GETCOLUMNORDERARRAY message explicitly.

ListView_GetColumnWidth

Gets the width of a column in report or list view. You can use this macro or send the LVM_GETCOLUMNWIDTH message explicitly.

ListView_GetCountPerPage

Calculates the number of items that can fit vertically in the visible area of a list-view control when in list or report view. Only fully visible items are counted. You can use this macro or send the LVM_GETCOUNTPERPAGE message explicitly.

ListView_GetEditControl

Gets the handle to the edit control being used to edit a list-view item's text. You can use this macro or send the LVM_GETEDITCONTROL message explicitly.

ListView_GetEmptyText

Gets the text meant for display when the list-view control appears empty. Use this macro or send the LVM_GETEMPTYTEXT message explicitly.

ListView_GetExtendedListViewStyle

Gets the extended styles that are currently in use for a given list-view control. You can use this macro or send the LVM_GETEXTENDEDLISTVIEWSTYLE message explicitly.

ListView_GetFocusedGroup

Gets the group that has the focus. Use this macro or send the LVM_GETFOCUSEDGROUP message explicitly.

ListView_GetFooterInfo	Gets information on the footer of a specified list-view control. Use this macro or send the LVM_GETFOOTERINFO message explicitly.
ListView_GetFooterItem	Gets information on a footer item for a specified list-view control. Use this macro or send the LVM_GETFOOTERITEM message explicitly.
ListView_GetFooterItemRect	Gets the coordinates of a footer for a specified item in a list-view control. Use this macro or send the LVM_GETFOOTERITEMRECT message explicitly.
ListView_GetFooterRect	Gets the coordinates of the footer for a specified list-view control. Use this macro or send the LVM_GETFOOTERRECT message explicitly.
ListView_GetGroupCount	Gets the number of groups. You can use this macro or send the LVM_GETGROUPCOUNT message explicitly.
ListView_GetGroupHeaderImageList	Gets the group header image list that has been set for an existing list-view control.
ListView_GetGroupInfo	Gets group information. You can use this macro or send the LVM_GETGROUPINFO message explicitly.
ListView_GetGroupInfoByIndex	Gets information on a specified group. Use this macro or send the LVM_GETGROUPINFOBYINDEX message explicitly.
ListView_GetGroupMetrics	Gets information about the display of groups. You can use this macro or send the LVM_GETGROUPMETRICS message explicitly.
ListView_GetGroupRect	Gets the rectangle for a specified group. Use this macro or send the LVM_GETGROUPRECT message explicitly.
ListView_GetGroupState	Gets the state for a specified group. Use this macro or send the LVM_GETGROUPSTATE message explicitly.
ListView_GetHeader	Gets the handle to the header control used by a list-view control. You can use this macro or send the LVM_GETHEADER message explicitly.

ListView_GetHotCursor

Gets the HCURSOR used when the pointer is over an item while hot tracking is enabled. You can use this macro or send the LVM_GETHOTCURSOR message explicitly.

ListView_GetHotItem

Gets the index of the hot item. You can use this macro or send the LVM_GETHOTITEM message explicitly.

ListView_GetHoverTime

Gets the amount of time that the mouse cursor must hover over an item before it is selected. You can use this macro or send the LVM_GETHOVERTIME message explicitly.

ListView_GetImageList

Gets the handle to an image list used for drawing list-view items. You can use this macro or send the LVM_GETIMAGELIST message explicitly.

ListView_GetInsertMark

Gets the position of the insertion point. You can use this macro or send the LVM_GETINSERTMARK message explicitly.

ListView_GetInsertMarkColor

Gets the color of the insertion point. You can use this macro or send the LVM_GETINSERTMARKCOLOR message explicitly.

ListView_GetInsertMarkRect

Gets the rectangle that bounds the insertion point. You can use this macro or send the LVM_GETINSERTMARKRECT message explicitly.

ListView_GetISearchString

Gets the incremental search string of a list-view control. You can use this macro or send the LVM_GETISEARCHSTRING message explicitly.

ListView_GetItem

Gets some or all of a list-view item's attributes. You can use this macro or send the LVM_GETITEM message explicitly.

ListView_GetItemCount

Gets the number of items in a list-view control. You can use this macro or send the LVM_GETITEMCOUNT message explicitly.

ListView_GetItemIndexRect

Gets the bounding rectangle for all or part of a subitem in the current view of a specified list-view control. Use this macro or send the LVM_GETITEMINDEXRECT message explicitly.

ListView_GetItemPosition

Gets the position of a list-view item. You can use this macro or explicitly send the LVM_GETITEMPOSITION message.

ListView_GetItemRect	Gets the bounding rectangle for all or part of an item in the current view. You can use this macro or send the LVM_GETITEMRECT message explicitly.
ListView_GetItemSpacing	Determines the spacing between items in a list-view control. You can use this macro or send the LVM_GETITEMSPACING message explicitly.
ListView_GetItemState	Gets the state of a list-view item. You can use this macro or send the LVM_GETITEMSTATE message explicitly.
ListView_GetItemText	Gets the text of a list-view item or subitem. You can use this macro or send the LVM_GETITEMTEXT message explicitly.
ListView_GetNextItem	Searches for a list-view item that has the specified properties and bears the specified relationship to a specified item. You can use this macro or send the LVM_GETNEXTITEM message explicitly.
ListView_GetNextItemIndex	Gets the index of the item in a particular list-view control that has the specified properties and relationship to another specific item. Use this macro or send the LVM_GETNEXTITEMINDEX message explicitly.
ListView_GetNumberOfWorkAreas	Gets the number of working areas in a list-view control. You can use this macro or send the LVM_GETNUMBEROFWORKAREAS message explicitly.
ListView_GetOrigin	Gets the current view origin for a list-view control. You can use this macro or send the LVM_GETORIGIN message explicitly.
ListView_GetOutlineColor	Gets the color of the border of a list-view control if the LVS_EX_BORDERSELECT extended window style is set. You can use this macro or send the LVM_GETOUTLINECOLOR message explicitly.
ListView_GetSelectedColumn	Gets an integer that specifies the selected column. You can use this macro or send the LVM_GETSELECTEDCOLUMN message explicitly.
ListView_GetSelectedCount	Determines the number of selected items in a list-view control. You can use this macro or send the LVM_GETSELECTEDCOUNT message explicitly.
ListView_GetSelectionMark	Gets the selection mark from a list-view control. You can use this macro or explicitly send the LVM_GETSELECTIONMARK message.

ListView_GetStringWidth Determines the width of a specified string using the specified list-view control's current font. You can use this macro or send the LVM_GETSTRINGWIDTH message explicitly.	
ListView_GetSubItemRect Gets information about the rectangle that surrounds a subitem in a list-view control.	
ListView_GetTextBkColor Gets the text background color of a list-view control. You can use this macro or send the LVM_GETTEXTBKCOLOR message explicitly.	
ListView_GetTextColor Gets the text color of a list-view control. You can use this macro or send the LVM_GETTEXTCOLOR message explicitly.	
ListView_GetTileInfo Gets information about a tile in a list-view control. You can use this macro or send the LVM_GETTILEINFO message explicitly.	
ListView_GetTileViewInfo Gets information about a list-view control in tile view. You can use this macro or send the LVM_GETTILEVIEWINFO message explicitly.	
ListView_GetToolTips Gets the tooltip control that the list-view control uses to display tooltips. You can use this macro or send the LVM_GETTOOLTIPS message explicitly.	
ListView_GetTopIndex Gets the index of the topmost visible item when in list or report view. You can use this macro or send the LVM_GETTOPINDEX message explicitly.	
ListView_GetUnicodeFormat Gets the Unicode character format flag for the control. You can use this macro or send the LVM_GETUNICODEFORMAT message explicitly.	
ListView_GetView Gets the current view of a list-view control. You can use this macro or send the LVM_GETVIEW message explicitly.	
ListView_GetViewRect Gets the bounding rectangle of all items in the list-view control. The list view must be in icon or small icon view. You can use this macro or send the LVM_GETVIEWRECT message explicitly.	
ListView_GetWorkAreas Gets the working areas from a list-view control. You can use this macro, or send the LVM_GETWORKAREAS message explicitly.	

ListView_HasGroup

Determines whether the list-view control has a specified group. You can use this macro or send the LVM_HASGROUP message explicitly.

ListView_HitTest

Determines which list-view item, if any, is at a specified position. You can use this macro or send the LVM_HITTEST message explicitly.

ListView_HitTestEx

Determines which list-view item, if any, is at a specified position. You can use this macro or send the LVM_HITTEST message explicitly.

ListView_InsertColumn

Inserts a new column in a list-view control. You can use this macro or send the LVM_INSERTCOLUMN message explicitly.

ListView_InsertGroup

Inserts a group into a list-view control. You can use this macro or send the LVM_INSERTGROUP message explicitly.

ListView_InsertGroupSorted

Inserts a group into an ordered list of groups. You can use this macro or send the LVM_INSERTGROUPSORTED message explicitly.

ListView_InsertItem

Inserts a new item in a list-view control. You can use this macro or send the LVM_INSERTITEM message explicitly.

ListView_InsertMarkHitTest

Retrieves the insertion point closest to a specified point. You can use this macro or send the LVM_INSERTMARKHITTEST message explicitly.

ListView_IsGroupViewEnabled

Checks whether the list-view control has group view enabled. You can use this macro or send the LVM_ISGROUPVIEWENABLED message explicitly.

ListView_IsItemVisible

Indicates whether an item in the list-view control is visible. Use this macro or send the LVM_ISITEMVISIBLE message explicitly.

ListView_MapIDToIndex

Maps the ID of an item to an index. You can use this macro or send the LVM_MAPIDTOINDEX message explicitly.

ListView_MapIndexToID

Maps the index of an item to a unique ID. You can use this macro or send the LVM_MAPINDEXTOID message explicitly.

ListView_MoveGroup	This macro is not implemented.
ListView_MoveItemToGroup	This macro is not implemented.
ListView_RedrawItems	Forces a list-view control to redraw a range of items. You can use this macro or send the LVM_REDRAWITEMS message explicitly.
ListView_RemoveAllGroups	Removes all groups from a list-view control. You can use this macro or send the LVM_REMOVEALLGROUPS message explicitly.
ListView_RemoveGroup	Removes a group from a list-view control. You can use this macro or send the LVM_REMOVEGROUP message explicitly.
ListView_Scroll	Scrolls the content of a list-view control. You can use this macro or send the LVM_SCROLL message explicitly.
ListView_SetBkColor	Sets the background color of a list-view control. You can use this macro or send the LVM_SETBKCOLOR message explicitly.
ListView_SetBkImage	Sets the background image in a list-view control. You can use this macro or send the LVM_SETBKIMAGE message explicitly.
ListView_SetCallbackMask	Changes the callback mask for a list-view control. You can use this macro or send the LVM_SETCALLBACKMASK message explicitly.
ListView_SetCheckState	Selects or deselects an item in a list-view control. You can use this macro or send the LVM_SETITEMSTATE message explicitly.
ListView_SetColumn	Sets the attributes of a list-view column. You can use this macro or send the LVM_SETCOLUMN message explicitly.
ListView_SetColumnOrderArray	Sets the left-to-right order of columns in a list-view control. You can use this macro or send the LVM_SETCOLUMNORDERARRAY message explicitly.
ListView_SetColumnWidth	Used to change the width of a column in report view or the width of all columns in list-view mode. You can use this macro or send the LVM_SETCOLUMNWIDTH message explicitly.

ListView_SetExtendedListViewStyle	Sets extended styles for list-view controls. You can use this macro or send the LVM_SETEXTENDEDLISTVIEWSTYLE message explicitly.
ListView_SetExtendedListViewStyleEx	Sets extended styles for list-view controls using the style mask. You can use this macro or send the LVM_SETEXTENDEDLISTVIEWSTYLE message explicitly.
ListView_SetGroupHeaderImageList	Assigns an image list to the group header of a list-view control.
ListView_SetGroupInfo	Sets group information. You can use this macro or send the LVM_SETGROUPINFO message explicitly.
ListView_SetGroupMetrics	Sets information about the display of groups. You can use this macro or send the LVM_SETGROUPMETRICS message explicitly.
ListView_SetGroupState	Sets the state for a specified group.
ListView_SetHotCursor	Sets the HCURSOR that the list-view control uses when the pointer is over an item while hot tracking is enabled. You can use this macro or send the LVM_SETHOTCURSOR message explicitly. To check whether hot tracking is enabled, call SystemParametersInfo.
ListView_SetHotItem	Sets the hot item in a list-view control. You can use this macro or send the LVM_SETHOTITEM message explicitly.
ListView_SetHoverTime	Sets the amount of time that the mouse cursor must hover over an item before it is selected. You can use this macro or send the LVM_SETHOVERTIME message explicitly.
ListView_SetIconSpacing	Sets the spacing between icons in list-view controls set to the LVS_ICON style. You can use this macro or send the LVM_SETICONSPACING message explicitly.
ListView_SetImageList	Assigns an image list to a list-view control. You can use this macro or send the LVM_SETIMAGELIST message explicitly.
ListView_SetInfoTip	Sets tooltip text. You can use this macro or send the LVM_SETINFOTIP message explicitly.

ListView_SetInsertMark	Sets the insertion point to the defined position. You can use this macro or send the LVM_SETINSERTMARK message explicitly.
ListView_SetInsertMarkColor	Sets the color of the insertion point. You can use this macro or send the LVM_SETINSERTMARKCOLOR message explicitly.
ListView_SetItem	Sets some or all of a list-view item's attributes. You can also use ListView_SetItem to set the text of a subitem. You can use this macro or send the LVM_SETITEM message explicitly.
ListView_SetItemCount	Causes the list-view control to allocate memory for the specified number of items. You can use this macro or send the LVM_SETITEMCOUNT message explicitly.
ListView_SetItemCountEx	Sets the virtual number of items in a virtual list view. You can use this macro or send the LVM_SETITEMCOUNT message explicitly.
ListView_SetItemIndexState	Sets the state of a specified list-view item. Use this macro or send the LVM_SETITEMINDEXSTATE message explicitly.
ListView_SetItemPosition	Moves an item to a specified position in a list-view control (in icon or small icon view). You can use this macro or send the LVM_SETITEMPOSITION message explicitly.
ListView_SetItemPosition32	Moves an item to a specified position in a list-view control (in icon or small icon view).
ListView_SetItemState	Changes the state of an item in a list-view control. You can use this macro or send the LVM_SETITEMSTATE message explicitly.
ListView_SetItemText	Changes the text of a list-view item or subitem. You can use this macro or send the LVM_SETITEMTEXT message explicitly.
ListView_SetOutlineColor	Sets the color of the border of a list-view control if the LVS_EX_BORDERSELECT extended window style is set. You can use this macro or send the LVM_SETOUTLINECOLOR message explicitly.
ListView_SetSelectedColumn	Sets the index of the selected column. You can use this macro or send the LVM_SETSELECTEDCOLUMN message explicitly.
ListView_SetSelectionMark	Sets the selection mark in a list-view control. You can use this macro or send the LVM_SETSELECTIONMARK message explicitly.

ListView_SetTextBkColor	Sets the background color of text in a list-view control. You can use this macro or send the LVM_SETTEXTBKCOLOR message explicitly.
ListView_SetTextColor	Sets the text color of a list-view control. You can use this macro or send the LVM_SETTEXTCOLOR message explicitly.
ListView_SetTileInfo	Sets information for an existing tile of a list-view control. You can use this macro or send the LVM_SETTILEINFO message explicitly.
ListView_SetTileViewInfo	Sets information that a list-view control uses in tile view. You can use this macro or send the LVM_SETTILEVIEWINFO message explicitly.
ListView_SetToolTips	Sets the tooltip control that the list-view control will use to display tooltips. You can use this macro or send the LVM_SETTOOLTIPS message explicitly.
ListView_SetUnicodeFormat	Sets the Unicode character format flag for the control.
ListView_SetView	Sets the view of a list-view control. You can use this macro or send the LVM_SETVIEW message explicitly.
ListView_SetWorkAreas	Sets the working areas within a list-view control. You can use this macro or send the LVM_SETWORKAREAS message explicitly.
ListView_SortGroups	Uses an application-defined comparison function to sort groups by ID within a list-view control. You can use this macro or send the LVM_SORTGROUPS message explicitly.
ListView_SortItems	Uses an application-defined comparison function to sort the items of a list-view control. The index of each item changes to reflect the new sequence. You can use this macro or send the LVM_SORTITEMS message explicitly.
ListView_SortItemsEx	Uses an application-defined comparison function to sort the items of a list-view control. The index of each item changes to reflect the new sequence. You can use this macro or send the LVM_SORTITEMSEX message explicitly.
ListView_SubItemHitTest	Determines which list-view item or subitem is located at a given position. You can use this macro or send the LVM_SUBITEMHITTEST message explicitly.

ListView_SubItemHitTestEx	Determines which list-view item or subitem is located at a given position. You can use this macro or send the LVM_SUBITEMHITTEST message explicitly.
ListView_Update	Updates a list-view item. If the list-view control has the LVS_AUTOARRANGE style, this macro causes the list-view control to be arranged. You can use this macro or send the LVM_UPDATE message explicitly.
LoadIconMetric	Loads a specified icon resource with a client-specified system metric.
LoadIconWithScaleDown	Loads an icon. If the icon is not a standard size, this function scales down a larger image instead of scaling up a smaller image.
MakeDragList	Changes the specified single-selection list box to a drag list box.
MAKEIPADDRESS	Packs four byte-values into a single LPARAM suitable for use with the IPM_SETADDRESS message.
MAKEIPRANGE	Packs two byte-values into a single LPARAM suitable for use with the IPM_SETRANGE message.
MenuHelp	Processes WM_MENUSELECT and WM_COMMAND messages and displays Help text about the current menu in the specified status window.
MonthCal_GetCalendarBorder	Gets the border size, in pixels, of a month calendar control. You can use this macro or send the MCM_GETCALENDARBORDER message explicitly.
MonthCal_GetCalendarCount	Gets the number of calendars currently displayed in the calendar control. You can use this macro or send the MCM_GETCALENDARCOUNT message explicitly.
MonthCal_GetCalendarGridInfo	Gets information about a calendar grid.
MonthCal_GetCALID	Gets the current calendar ID for the given calendar control. You can use this macro or send the MCM_GETCALID message explicitly.

MonthCal_GetColor

Retrieves the color for a given portion of a month calendar control. You can use this macro or send the MCM_GETCOLOR message explicitly.

MonthCal_GetCurrentView

Gets the view for a month calendar control. You can use this macro or send the MCM_GETCURRENTVIEW message explicitly.

MonthCal_GetCurSel

Retrieves the currently selected date. You can use this macro or send the MCM_GETCURSEL message explicitly.

MonthCal_GetFirstDayOfWeek

Retrieves the first day of the week for a month calendar control. You can use this macro or send the MCM_GETFIRSTDAYOFWEEK message explicitly.

MonthCal_GetMaxSelCount

Retrieves the maximum date range that can be selected in a month calendar control. You can use this macro or send the MCM_GETMAXSELCOUNT message explicitly.

MonthCal_GetMaxTodayWidth

Retrieves the maximum width of the "today" string in a month calendar control. This includes the label text and the date text. You can use this macro or send the MCM_GETMAXTODAYWIDTH message explicitly.

MonthCal_GetMinReqRect

Retrieves the minimum size required to display a full month in a month calendar control. Size information is presented in the form of a RECT structure. You can use this macro or send the MCM_GETMINREQRECT message explicitly.

MonthCal_GetMonthDelta

Retrieves the scroll rate for a month calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button. You can use this macro or send the MCM_GETMONTHDELTA message explicitly.

MonthCal_GetMonthRange

Retrieves date information (using SYSTEMTIME structures) that represents the high and low limits of a month calendar control's display. You can use this macro or send the MCM_GETMONTHRANGE message explicitly.

MonthCal_GetRange

Retrieves the minimum and maximum allowable dates set for a month calendar control. You can use this macro or send the MCM_GETRANGE message explicitly.

MonthCal_GetSelRange

Retrieves date information that represents the upper and lower limits of the date range currently selected by the user. You can use this macro or send the MCM_GETSELRANGE message explicitly.

MonthCal_GetToday

Retrieves the date information for the date specified as "today" for a month calendar control. You can use this macro or send the MCM_GETTODAY message explicitly.

MonthCal_GetUnicodeFormat

Retrieves the Unicode character format flag for the control. You can use this macro or send the MCM_GETUNICODEFORMAT message explicitly.

MonthCal_HitTest

Determines which portion of a month calendar control is at a given point on the screen. You can use this macro or send the MCM_HITTEST message explicitly.

MonthCal_SetCalendarBorder

Sets the border size, in pixels, of a month calendar control. You can use this macro or send the MCM_SETCALENDARBORDER message explicitly.

MonthCal_SetCALID

Sets the calendar ID for the given calendar control. You can use this macro or send the MCM_SETCALID message explicitly.

MonthCal_SetColor

Sets the color for a given portion of a month calendar control. You can use this macro or send the MCM_SETCOLOR message explicitly.

MonthCal_SetCurrentView

Sets the view for a month calendar control. You can use this macro or send the MCM_SETCURRENTVIEW message explicitly.

MonthCal_SetCurSel

Sets the currently selected date for a month calendar control. If the specified date is not in view, the control updates the display to bring it into view. You can use this macro or send the MCM_SETCURSEL message explicitly.

MonthCal_SetDayState

Sets the day states for all months that are currently visible within a month calendar control. You can use this macro or send the MCM_SETDAYSTATE message explicitly.

MonthCal_SetFirstDayOfWeek

Sets the first day of the week for a month calendar control. You can use this macro or send the MCM_SETFIRSTDAYOFWEEK message explicitly.

MonthCal_SetMaxSelCount

Sets the maximum number of days that can be selected in a month calendar control. You can use this macro or send the MCM_SETMAXSELCOUNT message explicitly.

MonthCal_SetMonthDelta	Sets the scroll rate for a month calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button. You can use this macro or send the MCM_SETMONTHDELTA message explicitly.
MonthCal_SetRange	Sets the minimum and maximum allowable dates for a month calendar control. You can use this macro or send the MCM_SETRANGE message explicitly.
MonthCal_SetSelRange	Sets the selection for a month calendar control to a given date range. You can use this macro or send the MCM_SETSELRANGE message explicitly.
MonthCal_SetToday	Sets the "today" selection for a month calendar control. You can use this macro or send the MCM_SETTODAY message explicitly.
MonthCal_SetUnicodeFormat	Sets the Unicode character format flag for the control.
MonthCal_SizeRectToMin	Calculates how many calendars will fit in the given rectangle, and then returns the minimum size that a rectangle needs to be to fit that number of calendars. You can use this macro or send the MCM_SIZERECTTOMIN message explicitly.
Pager_ForwardMouse	Enables or disables mouse forwarding for the pager control. When mouse forwarding is enabled, the pager control forwards WM_MOUSEMOVE messages to the contained window. You can use this macro or send the PGM_FORWARDMOUSE message explicitly.
Pager_GetBkColor	Retrieves the current background color for the pager control. You can use this macro or send the PGM_GETBKCOLOR message explicitly.
Pager_GetBorder	Retrieves the current border size for the pager control. You can use this macro or send the PGM_GETBORDER message explicitly.
Pager_GetButtonSize	Retrieves the current button size for the pager control. You can use this macro or send the PGM_GETBUTTONSIZE message explicitly.
Pager_GetButtonState	Retrieves the state of the specified button in a pager control. You can use this macro or send the PGM_GETBUTTONSTATE message explicitly.

Pager_GetDropTarget	Retrieves a pager control's IDropTarget interface pointer. You can use this macro or send the PGM_GETDROPTARGET message explicitly.
Pager_GetPos	Retrieves the current scroll position of the pager control. You can use this macro or send the PGM_GETPOS message explicitly.
Pager_RecalcSize	Forces the pager control to recalculate the size of the contained window. Using this macro will result in a PGN_CALCSIZE notification being sent. You can use this macro or send the PGM_RECALCSIZE message explicitly.
Pager_SetBkColor	Sets the current background color for the pager control. You can use this macro or send the PGM_SETBKCOLOR message explicitly.
Pager_SetBorder	Sets the current border size for the pager control. You can use this macro or send the PGM_SETBORDER message explicitly.
Pager_SetButtonSize	Sets the current button size for the pager control. You can use this macro or send the PGM_SETBUTTONSIZE message explicitly.
Pager_SetChild	Sets the contained window for the pager control.
Pager_SetPos	Sets the scroll position for the pager control. You can use this macro or send the PGM_SETPOS message explicitly.
Pager_SetScrollInfo	Sets the scrolling parameters of the pager control, including the timeout value, the lines per timeout, and the pixels per line. You can use this macro or send the PGM_SETSETSCROLLINFO message explicitly.
RemoveWindowSubclass	Removes a subclass callback from a window.
SECOND_IPADDRESS	Extracts the field 1 value from a packed IP address retrieved with the IPM_GETADDRESS message.
SetWindowSubclass	Installs or updates a window subclass callback.
ShowHideMenuCtl	Sets or removes the specified menu item's check mark attribute and shows or hides the corresponding control.

TabCtrl_AdjustRect	Calculates a tab control's display area given a window rectangle, or calculates the window rectangle that would correspond to a specified display area. You can use this macro or send the TCM_ADJUSTRECT message explicitly.
TabCtrl_DeleteAllItems	Removes all items from a tab control. You can use this macro or send the TCM_DELETEALLITEMS message explicitly.
TabCtrl_DeleteItem	Removes an item from a tab control. You can use this macro or send the TCM_DELETEITEM message explicitly.
TabCtrl_DeselectAll	Resets items in a tab control, clearing any that were set to the TCIS_BUTTONPRESSED state. You can use this macro or send the TCM_DESELECTALL message explicitly.
TabCtrl_GetCurFocus	Returns the index of the item that has the focus in a tab control. You can use this macro or send the TCM_GETCURFOCUS message explicitly.
TabCtrl_GetCurSel	Determines the currently selected tab in a tab control. You can use this macro or send the TCM_GETCURSEL message explicitly.
TabCtrl_GetExtendedStyle	Retrieves the extended styles that are currently in use for the tab control. You can use this macro or send the TCM_GETEXTENDEDSTYLE message explicitly.
TabCtrl_GetImageList	Retrieves the image list associated with a tab control. You can use this macro or send the TCM_GETIMAGELIST message explicitly.
TabCtrl_GetItem	Retrieves information about a tab in a tab control. You can use this macro or send the TCM_GETITEM message explicitly.
TabCtrl_GetItemCount	Retrieves the number of tabs in the tab control. You can use this macro or send the TCM_GETITEMCOUNT message explicitly.
TabCtrl_GetItemRect	Retrieves the bounding rectangle for a tab in a tab control. You can use this macro or send the TCM_GETITEMRECT message explicitly.
TabCtrl_GetRowCount	Retrieves the current number of rows of tabs in a tab control. You can use this macro or send the TCM_GETROWCOUNT message explicitly.

TabCtrl_GetToolTips

Retrieves the handle to the tooltip control associated with a tab control. You can use this macro or send the TCM_GETTOOLTIPS message explicitly.

TabCtrl_GetUnicodeFormat

Retrieves the UNICODE character format flag for the control. You can use this macro or send the TCM_GETUNICODEFORMAT message explicitly.

TabCtrl_HighlightItem

Sets the highlight state of a tab item. You can use this macro or send the TCM_HIGHLIGHTITEM message explicitly.

TabCtrl_HitTest

Determines which tab, if any, is at a specified screen position. You can use this macro or send the TCM_HITTEST message explicitly.

TabCtrl_InsertItem

Inserts a new tab in a tab control. You can use this macro or send the TCM_INSERTITEM message explicitly.

TabCtrl_RemoveImage

Removes an image from a tab control's image list. You can use this macro or send the TCM_REMOVEIMAGE message explicitly.

TabCtrl_SetCurFocus

Sets the focus to a specified tab in a tab control. You can use this macro or send the TCM_SETCURFOCUS message explicitly.

TabCtrl_SetCurSel

Selects a tab in a tab control. You can use this macro or send the TCM_SETCURSEL message explicitly.

TabCtrl_SetExtendedStyle

Sets the extended styles that the tab control will use. You can use this macro or send the TCM_SETEXTENDEDSTYLE message explicitly.

TabCtrl_SetImageList

Assigns an image list to a tab control. You can use this macro or send the TCM_SETIMAGELIST message explicitly.

TabCtrl_SetItem

Sets some or all of a tab's attributes. You can use this macro or send the TCM_SETITEM message explicitly.

TabCtrl_SetItemExtra

Sets the number of bytes per tab reserved for application-defined data in a tab control. You can use this macro or send the TCM_SETITEMEXTRA message explicitly.

TabCtrl_SetItemSize	Sets the width and height of tabs in a fixed-width or owner-drawn tab control. You can use this macro or send the TCM_SETITEMSIZE message explicitly.
TabCtrl_SetMinTabWidth	Sets the minimum width of items in a tab control. You can use this macro or send the TCM_SETMINTABWIDTH message explicitly.
TabCtrl_SetPadding	Sets the amount of space (padding) around each tab's icon and label in a tab control. You can use this macro or send the TCM_SETPADDING message explicitly.
TabCtrl_SetToolTips	Assigns a tooltip control to a tab control. You can use this macro or send the TCM_SETTOOLTIPS message explicitly.
TabCtrl_SetUnicodeFormat	Sets the Unicode character format flag for the control.
TaskDialog	The TaskDialog function creates, displays, and operates a task dialog.
TaskDialogIndirect	The TaskDialogIndirect function creates, displays, and operates a task dialog.
THIRD_IPADDRESS	Extracts the field 2 value from a packed IP address retrieved with the IPM_GETADDRESS message.
TreeView_CreateDragImage	Creates a dragging bitmap for the specified item in a tree-view control.
TreeView_DeleteAllItems	Deletes all items from a tree-view control.
TreeView_DeleteItem	Removes an item and all its children from a tree-view control. You can also send the TVM_DELETEITEM message explicitly.
TreeView_EditLabel	Begins in-place editing of the specified item's text, replacing the text of the item with a single-line edit control containing the text.
TreeView_EndEditLabelNow	Ends the editing of a tree-view item's label. You can use this macro or send the TVM_ENDEDITLABELNOW message explicitly.

TreeView_EnsureVisible	Ensures that a tree-view item is visible, expanding the parent item or scrolling the tree-view control, if necessary. You can use this macro or send the TVM_ENSUREVISIBLE message explicitly.
TreeView_Expand	The TreeView_Expand macro expands or collapses the list of child items associated with the specified parent item, if any. You can use this macro or send the TVM_EXPAND message explicitly.
TreeView_GetBkColor	Retrieves the current background color of the control. You can use this macro or send the TVM_GETBKCOLOR message explicitly.
TreeView_GetCheckState	Gets the check state of the specified item. You can also use the TVM_GETITEMSTATE message directly.
TreeView_GetChild	Retrieves the first child item of the specified tree-view item. You can use this macro, or you can explicitly send the TVM_GETNEXTITEM message with the TVGN_CHILD flag.
TreeView_GetCount	Retrieves a count of the items in a tree-view control. You can use this macro or send the TVM_GETCOUNT message explicitly.
TreeView_GetDropHilight	Retrieves the tree-view item that is the target of a drag-and-drop operation. You can use this macro, or you can explicitly send the TVM_GETNEXTITEM message with the TVGN_DROPHILITE flag.
TreeView_GetEditControl	Retrieves the handle to the edit control being used to edit a tree-view item's text. You can use this macro or send the TVM_GETEDITCONTROL message explicitly.
TreeView_GetExtendedStyle	Retrieves the extended style for a specified tree-view control. Use this macro or send the TVM_GETEXTENDEDSTYLE message explicitly.
TreeView_GetFirstVisible	Retrieves the first visible item in a tree-view control window. You can use this macro, or you can explicitly send the TVM_GETNEXTITEM message with the TVGN_FIRSTVISIBLE flag.
TreeView_GetImageList	Retrieves the handle to the normal or state image list associated with a tree-view control. You can use this macro or send the TVM_GETIMAGELIST message explicitly.

TreeView_GetIndent

Retrieves the amount, in pixels, that child items are indented relative to their parent items. You can use this macro or send the TVM_GETINDENT message explicitly.

TreeView_GetInsertMarkColor

Retrieves the color used to draw the insertion mark for the tree view. You can use this macro or send the TVM_GETINSERTMARKCOLOR message explicitly.

TreeView_GetISearchString

Retrieves the incremental search string for a tree-view control. The tree-view control uses the incremental search string to select an item based on characters typed by the user. You can use this macro or send the TVM_GETISEARCHSTRING message explicitly.

TreeView_GetItem

Retrieves some or all of a tree-view item's attributes. You can use this macro or send the TVM_GETITEM message explicitly.

TreeView_GetItemHeight

Retrieves the current height of the tree-view items. You can use this macro or send the TVM_GETITEMHEIGHT message explicitly.

TreeView_GetItemPartRect

Retrieves the largest possible bounding rectangle that constitutes the "hit zone" for a specified part of an item. Use this macro or send the TVM_GETITEMPARTRECT message explicitly.

TreeView_GetItemRect

Retrieves the bounding rectangle for a tree-view item and indicates whether the item is visible. You can use this macro or send the TVM_GETITEMRECT message explicitly.

TreeView_GetItemState

Retrieves some or all of a tree-view item's state attributes. You can use this macro or send the TVM_GETITEMSTATE message explicitly.

TreeView_GetLastVisible

Retrieves the last expanded item in a tree-view control. This does not retrieve the last item visible in the tree-view window. You can use this macro, or you can explicitly send the TVM_GETNEXTITEM message with the TVGN_LASTVISIBLE flag.

TreeView_GetLineColor

Gets the current line color. You can also use the TVM_GETLINECOLOR message directly.

TreeView_GetNextItem

Retrieves the tree-view item that bears the specified relationship to a specified item. You can use this macro, use one of the TreeView_Get macros described below, or send the TVM_GETNEXTITEM message explicitly.

TreeView_GetNextSelected

Retrieves the tree-view item that bears the TVGN_NEXTSELECTED relationship to a specified tree item.

TreeView_GetNextSibling

Retrieves the next sibling item of a specified item in a tree-view control. You can use this macro, or you can explicitly send the TVM_GETNEXTITEM message with the TVGN_NEXT flag.

TreeView_GetNextVisible

Retrieves the next visible item that follows a specified item in a tree-view control. You can use this macro, or you can explicitly send the TVM_GETNEXTITEM message with the TVGN_NEXTVISIBLE flag.

TreeView_GetParent

Retrieves the parent item of the specified tree-view item. You can use this macro, or you can explicitly send the TVM_GETNEXTITEM message with the TVGN_PARENT flag.

TreeView_GetPrevSibling

Retrieves the previous sibling item of a specified item in a tree-view control. You can use this macro, or you can explicitly send the TVM_GETNEXTITEM message with the TVGN_PREVIOUS flag.

TreeView_GetPrevVisible

Retrieves the first visible item that precedes a specified item in a tree-view control. You can use this macro, or you can explicitly send the TVM_GETNEXTITEM message with the TVGN_PREVIOUSVISIBLE flag.

TreeView_GetRoot

Retrieves the topmost or very first item of the tree-view control. You can use this macro, or you can explicitly send the TVM_GETNEXTITEM message with the TVGN_ROOT flag.

TreeView_GetScrollTime

Retrieves the maximum scroll time for the tree-view control. You can use this macro or send the TVM_GETSCROLLTIME message explicitly.

TreeView_GetSelectedCount

TreeView_GetSelectedCount macro

TreeView_GetSelection

Retrieves the currently selected item in a tree-view control. You can use this macro, or you can explicitly send the TVM_GETNEXTITEM message with the TVGN_CARET flag.

TreeView_GetTextColor

Retrieves the current text color of the control. You can use this macro or send the TVM_GETTEXTCOLOR message explicitly.

TreeView_GetToolTips

Retrieves the handle to the child tooltip control used by a tree-view control. You can use this macro or send the TVM_GETTOOLTIPS message explicitly.

TreeView_GetUnicodeFormat	Retrieves the Unicode character format flag for the control. You can use this macro or send the TVM_GETUNICODEFORMAT message explicitly.
TreeView_GetVisibleCount	Obtains the number of items that can be fully visible in the client window of a tree-view control. You can use this macro or send the TVM_GETVISIBLECOUNT message explicitly.
TreeView_HitTest	Determines the location of the specified point relative to the client area of a tree-view control. You can use this macro or send the TVM_HITTEST message explicitly.
TreeView_InsertItem	Inserts a new item in a tree-view control. You can use this macro or send the TVM_INSERTITEM message explicitly.
TreeView_MapAccIDToHTREEITEM	Maps an accessibility ID to an HTREEITEM. You can use this macro or send the TVM_MAPACCIDTOHTREEITEM message explicitly.
TreeView_MapHTREEITEMToAccID	Maps an HTREEITEM to an accessibility ID. You can use this macro or send the TVM_MAPHTREEITEMTOACCID message explicitly.
TreeView_Select	Selects the specified tree-view item, scrolls the item into view, or redraws the item in the style used to indicate the target of a drag-and-drop operation.
TreeView_SelectDropTarget	Redraws a specified tree-view control item in the style used to indicate the target of a drag-and-drop operation. You can use this macro or the TreeView_Select macro, or you can send the TVM_SELECTITEM message explicitly.
TreeView_SelectItem	Selects the specified tree-view item. You can use this macro or the TreeView_Select macro, or you can send the TVM_SELECTITEM message explicitly.
TreeView_SelectSetFirstVisible	Scrolls the tree-view control vertically to ensure that the specified item is visible.
TreeView_SetAutoScrollInfo	Sets information used to determine auto-scroll characteristics. Use this macro or send the TVM_SETAUTOSCROLLINFO message explicitly.
TreeView_SetBkColor	Sets the background color of the control. You can use this macro or send the TVM_SETBKCOLOR message explicitly.

TreeView_SetBorder	Sets the size of the border for the items in a tree-view control. You can use this macro or send the TVM_SETBORDER message explicitly.
TreeView_SetCheckState	Sets the item's state image to "checked" or "unchecked." You can also use the TVM_SETITEM message directly.
TreeView_SetExtendedStyle	Sets the extended style for a specified TreeView control. Use this macro or send the TVM_SETEXTENDEDSTYLE message explicitly.
TreeView_SetHot	Sets the hot item for a tree-view control. You can use this macro or send the TVM_SETHOT message explicitly.
TreeView_SetImageList	Sets the normal or state image list for a tree-view control and redraws the control using the new images. You can use this macro or send the TVM_SETIMAGELIST message explicitly.
TreeView_SetIndent	Sets the width of indentation for a tree-view control and redraws the control to reflect the new width. You can use this macro or send the TVM_SETINDENT message explicitly.
TreeView_SetInsertMark	Sets the insertion mark in a tree-view control. You can use this macro or send the TVM_SETINSERTMARK message explicitly.
TreeView_SetInsertMarkColor	Sets the color used to draw the insertion mark for the tree view. You can use this macro or send the TVM_SETINSERTMARKCOLOR message explicitly.
TreeView_SetItem	The TreeView_SetItem macro sets some or all of a tree-view item's attributes. You can use this macro or send the TVM_SETITEM message explicitly.
TreeView_SetItemHeight	Sets the height of the tree-view items. You can use this macro or send the TVM_SETITEMHEIGHT message explicitly.
TreeView_SetItemState	Sets a tree-view item's state attributes. You can use this macro or send the TVM_SETITEM message explicitly.
TreeView_SetLineColor	Sets the current line color. You can also use the TVM_SETLINECOLOR message directly.

TreeView_SetScrollTime Sets the maximum scroll time for the tree-view control. You can use this macro or send the TVM_SETSCROLLTIME message explicitly.
TreeView_SetTextColor Sets the text color of the control. You can use this macro or send the TVM_SETTEXTCOLOR message explicitly.
TreeView_SetToolTips Sets a tree-view control's child tooltip control. You can use this macro or send the TVM_SETTOOLTIPS message explicitly.
TreeView_SetUnicodeFormat Sets the Unicode character format flag for the control.
TreeView_ShowInfoTip Shows the infotip for a specified item in a tree-view control. Use this macro or send the TVM_SHOWINFOTIP message explicitly.
TreeView_SortChildren Sorts the child items of the specified parent item in a tree-view control. You can use this macro or send the TVM_SORTCHILDREN message explicitly.
TreeView_SortChildrenCB Sorts tree-view items using an application-defined callback function that compares the items. You can use this macro or send the TVM_SORTCHILDRENCB message explicitly.
UninitializeFlatSB Uninitializes flat scroll bars for a particular window. The specified window will revert to standard scroll bars.

Callback functions

PFNLVGROUPCOMPARE The LVGroupCompare function is an application-defined callback function used with the LVM_INSERTGROUPSORTED and LVM_SORTGROUPS messages.
PFTASKDIALOGCALLBACK The TaskDialogCallbackProc function is an application-defined function used with the TaskDialogIndirect function.
SUBCLASSPROC Defines the prototype for the callback function used by RemoveWindowSubclass and SetWindowSubclass.

Structures

BUTTON_IMAGELIST	Contains information about an image list that is used with a button control.
BUTTON_SPLITINFO	Contains information that defines a split button (BS_SPLITBUTTON and BS_DEFSPLITBUTTON styles). Used with the BCM_GETSPLITINFO and BCM_SETSPLITINFO messages.
COLORMAP	Contains information used by the CreateMappedBitmap function to map the colors of the bitmap.
COLORSCHEME	Contains information for the drawing of buttons in a toolbar or rebar.
COMBOBOXEXITEMA	Contains information about an item in a ComboBoxEx control.
COMBOBOXEXITEMW	Contains information about an item in a ComboBoxEx control.
DATETIMEPICKERINFO	Contains information about a date and time picker (DTP) control.
DRAGLISTINFO	Contains information about a drag event. The pointer to DRAGLISTINFO is passed as the lParam parameter of the drag list message.
EDITBALLOONTIP	Contains information about a balloon tip associated with a button control.
HD_TEXTFILTERA	Contains information about header control text filters.
HD_TEXTFILTERW	Contains information about header control text filters.
HDHITTESTINFO	Contains information about a hit test. This structure is used with the HDM_HITTEST message and it supersedes the HD_HITTESTINFO structure.
HDITEMA	Contains information about an item in a header control. This structure supersedes the HD_ITEM structure.

HDITEMW	Contains information about an item in a header control. This structure supersedes the HD_ITEM structure.
HDLAYOUT	Contains information used to set the size and position of a header control. HDLAYOUT is used with the HDM_LAYOUT message. This structure supersedes the HD_LAYOUT structure.
IMAGEINFO	Contains information about an image in an image list. This structure is used with the IImageList::GetImageInfo function.
IMAGELISTDRAWPARAMS	Contains information about an image list draw operation and is used with the IImageList::Draw function.
INITCOMMONCONTROLSEX	Carries information used to load common control classes from the dynamic-link library (DLL). This structure is used with the InitCommonControlsEx function.
LHITTESTINFO	Used to get information about the link corresponding to a given location.
LITEM	Used to set and retrieve information about a link item.
LVBKIMAGEA	Contains information about the background image of a list-view control. This structure is used for both setting and retrieving background image information.
LVBKIMAGEW	Contains information about the background image of a list-view control. This structure is used for both setting and retrieving background image information.
LVCOLUMNA	Contains information about a column in report view. This structure is used both for creating and manipulating columns. This structure supersedes the LV_COLUMN structure.
LVCOLUMNW	Contains information about a column in report view. This structure is used both for creating and manipulating columns. This structure supersedes the LV_COLUMN structure.
LVFINDINFOA	Contains information used when searching for a list-view item. This structure is identical to LV_FINDINFO but has been renamed to fit standard naming conventions.

LVFINDINFOW	Contains information used when searching for a list-view item. This structure is identical to LV_FINDINFO but has been renamed to fit standard naming conventions.
LVFOOTERINFO	Contains information on a footer in a list-view control.
LVFOOTERITEM	Contains information on a footer item.
LVGROUP	Used to set and retrieve groups.
LVGROUPMETRICS	Contains information about the display of groups in a list-view control.
LVHITTESTINFO	Contains information about a hit test.
LVINSERTGROUPSORTED	Used to sort groups. It is used with LVM_INSERTGROUPSORTED.
LVINSERTMARK	Used to describe insertion points.
LVITEMA	Specifies or receives the attributes of a list-view item. This structure has been updated to support a new mask value (LVIF_INDENT) that enables item indenting. This structure supersedes the LV_ITEM structure.
LVITEMINDEX	Contains index information about a list-view item.
LVITEMW	Specifies or receives the attributes of a list-view item. This structure has been updated to support a new mask value (LVIF_INDENT) that enables item indenting. This structure supersedes the LV_ITEM structure.
LVSETINFOTIP	Provides information about tooltip text that is to be set.
LVTILEINFO	Provides information about an item in a list-view control when it is displayed in tile view.

LVTILEVIEWINFO	Provides information about a list-view control when it is displayed in tile view.
MCGRIDINFO	Contains information about part of a calendar control.
MCHITTESTINFO	Carries information specific to hit-testing points for a month calendar control. This structure is used with the MCM_HITTEST message and the corresponding MonthCal_HitTest macro.
NMBCDROPDOWN	Contains information about a BCN_DROPDOWN notification.
NMBCHOTITEM	Contains information about the movement of the mouse over a button control.
NMCBEDRAGBEGINA	Contains information used with the CBEN_DRAGBEGIN notification code.
NMCBEDRAGBEGINW	Contains information used with the CBEN_DRAGBEGIN notification code.
NMCBEENDEDITA	Contains information about the conclusion of an edit operation within a ComboBoxEx control. This structure is used with the CBEN_ENDEDIT notification code.
NMCBEENDEDITW	Contains information about the conclusion of an edit operation within a ComboBoxEx control. This structure is used with the CBEN_ENDEDIT notification code.
NMCHAR	Contains information used with character notification messages.
NMCOMBOBOXEXA	Contains information specific to ComboBoxEx items for use with notification codes.
NMCOMBOBOXEXW	Contains information specific to ComboBoxEx items for use with notification codes.
NMCUSTOMDRAW	Contains information specific to an NM_CUSTOMDRAW notification code.

NMCUSTOMSPLITRECTINFO	Contains information about the two rectangles of a split button. Sent with the NM_GETCUSTOMSPLITRECT notification.
NMCUSTOMTEXT	Contains information used with custom text notification.
NMDATETIMECHANGE	Contains information about a change that has taken place in a date and time picker (DTP) control. This structure is used with the DTN_DATETIMECHANGE notification code.
NMDATETIMEFORMATA	Contains information about a portion of the format string that defines a callback field within a date and time picker (DTP) control.
NMDATETIMEFORMATQUERYA	Contains information about a date and time picker (DTP) control callback field.
NMDATETIMEFORMATQUERYW	Contains information about a date and time picker (DTP) control callback field.
NMDATETIMEFORMATW	Contains information about a portion of the format string that defines a callback field within a date and time picker (DTP) control.
NMDATETIMESTRINGA	Contains information specific to an edit operation that has taken place in a date and time picker (DTP) control. This message is used with the DTN_USERSTRING notification code.
NMDATETIMESTRINGW	Contains information specific to an edit operation that has taken place in a date and time picker (DTP) control. This message is used with the DTN_USERSTRING notification code.
NMDATETIMEWMKEYDOWNA	Carries information used to describe and handle a DTN_WMKEYDOWN notification code.
NMDATETIMEWMKEYDOWNW	Carries information used to describe and handle a DTN_WMKEYDOWN notification code.
NMDAYSTATE	Carries information required to process the MCN_GETDAYSTATE notification code. All members of this structure are for input, except prgDayState, which the receiving application must set when processing MCN_GETDAYSTATE.

NMHDDISPINFOA	Contains information used in handling HDN_GETDISPINFO notification codes.
NMHDDISPINFOW	Contains information used in handling HDN_GETDISPINFO notification codes.
NMHDFILTERBTNCLICK	Specifies or receives the attributes of a filter button click.
NMHEADERA	Contains information about header control notification messages. This structure supersedes the HD_NOTIFY structure.
NMHEADERW	Contains information about header control notification messages. This structure supersedes the HD_NOTIFY structure.
NMIPADDRESS	Contains information for the IPN_FIELDCHANGED notification code.
NMITEMACTIVATE	Contains information about an LVN_ITEMACTIVATE notification code.
NMKEY	Contains information used with key notification messages.
NMLINK	The NMLINK Contains notification information. Send this structure with the NM_CLICK or NM_RETURN messages.
NMLISTVIEW	Contains information about a list-view notification message. This structure is the same as the NM_LISTVIEW structure but has been renamed to fit standard naming conventions.
NMLVCACHEHINT	Contains information used to update the cached item information for use with a virtual list view.
NMLVCUSTOMDRAW	Contains information specific to an NM_CUSTOMDRAW (list view) notification code sent by a list-view control.
NMLVDISPINFOA	Contains information about an LVN_GETDISPINFO or LVN_SETDISPINFO notification code. This structure is the same as the LV_DISPINFO structure, but has been renamed to fit standard naming conventions.

NMLVDISPINFOW	Contains information about an LVN_GETDISPINFO or LVN_SETDISPINFO notification code. This structure is the same as the LV_DISPINFO structure, but has been renamed to fit standard naming conventions.
NMLVEMPTYMARKUP	Contains information used with the LVN_GETEMPTYMARKUP notification code.
NMLVFINDITEMA	Contains information the owner needs to find items requested by a virtual list-view control. This structure is used with the LVN_ODFINDITEM notification code.
NMLVFINDITEMW	Contains information the owner needs to find items requested by a virtual list-view control. This structure is used with the LVN_ODFINDITEM notification code.
NMLVGETINFOTIPA	Contains and receives list-view item information needed to display a tooltip for an item. This structure is used with the LVN_GETINFOTIP notification code.
NMLVGETINFOTIPW	Contains and receives list-view item information needed to display a tooltip for an item. This structure is used with the LVN_GETINFOTIP notification code.
NMLVKEYDOWN	Contains information used in processing the LVN_KEYDOWN notification code. This structure is the same as the NMLVKEYDOWN structure but has been renamed to fit standard naming conventions.
NMLVLINK	Contains information about an LVN_LINKCLICK notification code.
NMLVODSTATECHANGE	Structure that contains information for use in processing the LVN_ODSTATECHANGED notification code.
NMLVSCROLL	Provides information about a scrolling operation.
NMMOUSE	Contains information used with mouse notification messages.
NMOBJECTNOTIFY	Contains information used with the TBN_GETOBJECT, TCN_GETOBJECT, and PSN_GETOBJECT notification codes.

NMPGCALCSIZE

Contains and receives information that the pager control uses to calculate the scrollable area of the contained window. It is used with the PGN_CALCSIZE notification.

NMPGHOTITEM

Contains information used with the PGN_HOTITEMCHANGE notification code.

NMPGSCROLL

Contains and receives information that the pager control uses when scrolling the contained window. It is used with the PGN_SCROLL notification.

NMRBAUTOSIZE

Contains information used in handling the RBN_AUTOSIZE notification codes.

NMREBAR

Contains information used in handling various rebar notifications.

NMREBARAUTOBREAK

Contains information used with the RBN_AUTOBREAK notification code.

NMREBARCHEVRON

Contains information used in handling the RBN_CHEVRONPUSHED notification code.

NMREBARCHILDSIZE

Contains information used in handling the RBN_CHILDSIZE notification code.

NMREBARSPLITTER

Contains information used to handle an RBN_SPLITTERDRAG notification code.

NMSEARCHWEB

Contains information used to handle an EN_SEARCHWEB notification code.

NMSELCHANGE

Carries information required to process the MCN_SELCHANGE notification code.

NMTBCUSTOMDRAW

Contains information specific to an NM_CUSTOMDRAW notification code sent by a toolbar control.

NMTBDISPINFOA

Contains and receives display information for a toolbar item. This structure is used with the TBN_GETDISPINFO notification code.

NMTBDISPINFOW	Contains and receives display information for a toolbar item. This structure is used with the TBN_GETDISPINFO notification code.
NMTBGETINFOTIPA	Contains and receives infotip information for a toolbar item. This structure is used with the TBN_GETINFOTIP notification code.
NMTBGETINFOTIPW	Contains and receives infotip information for a toolbar item. This structure is used with the TBN_GETINFOTIP notification code.
NMTBHOTITEM	Contains information used with the TBN_HOTITEMCHANGE notification code.
NMTBRESTORE	Allows applications to extract the information that was placed in NMTBSAVE when the toolbar state was saved. This structure is passed to applications when they receive a TBN_RESTORE notification code.
NMTBSAVE	This structure is passed to applications when they receive a TBN_SAVE notification code. It contains information about the button currently being saved. Applications can modify the values of the members to save additional information.
NMTCKEYDOWN	Contains information about a key press in a tab control. It is used with the TCN_KEYDOWN notification code. This structure supersedes the TC_KEYDOWN structure.
NMTOOLBARA	Contains information used to process toolbar notification codes. This structure supersedes the TBNOTIFY structure.
NMTOOLBARW	Contains information used to process toolbar notification codes. This structure supersedes the TBNOTIFY structure.
NMTOOLTIPSCREATED	Contains information used with NM_TOOLTIPSCREATED notification codes.
NMTRBTHUMBPOSCHANGING	Contains information about a trackbar change notification. This message is sent with the TRBN_THUMBPOSCHANGING notification.
NMTREEVIEWA	Contains information about a tree-view notification message. This structure is identical to the NM_TREEVIEW structure, but it has been renamed to follow current naming conventions.

NMTREEVIEWW

Contains information about a tree-view notification message. This structure is identical to the NM_TREEVIEW structure, but it has been renamed to follow current naming conventions.

NMTTCUSTOMDRAW

Contains information specific to an NM_CUSTOMDRAW notification code sent by a tooltip control.

NMTTDISPINFOA

Contains information used in handling the TTN_GETDISPINFO notification code. This structure supersedes the TOOLTIPTEXT structure.

NMTTDISPINFOW

Contains information used in handling the TTN_GETDISPINFO notification code. This structure supersedes the TOOLTIPTEXT structure.

NMTVASYNCDRAW

Contains an explanation of why the draw of an icon or overlay tree item failed.

NMTVCUSTOMDRAW

Contains information specific to an NM_CUSTOMDRAW (tree view) notification code sent by a tree-view control.

NMTVDISPINFOA

Contains and receives display information for a tree-view item. This structure is identical to the TV_DISPINFO structure, but it has been renamed to follow current naming conventions.

NMTVDISPINFOEXA

Contains information pertaining to extended TreeView notification information.

NMTVDISPINFOEXW

Contains information pertaining to extended TreeView notification information.

NMTVDISPINFOW

Contains and receives display information for a tree-view item. This structure is identical to the TV_DISPINFO structure, but it has been renamed to follow current naming conventions.

NMTVGETINFOTIPA

Contains and receives tree-view item information needed to display a tooltip for an item. This structure is used with the TVN_GETINFOTIP notification code.

NMTVGETINFOTIPW

Contains and receives tree-view item information needed to display a tooltip for an item. This structure is used with the TVN_GETINFOTIP notification code.

NMTVITEMCHANGE

Contains information on a tree-view item change. This structure is sent with the TVN_ITEMCHANGED and TVN_ITEMCHANGING notifications.

NMTVKEYDOWN

Contains information about a keyboard event in a tree-view control. This structure is used with the TVN_KEYDOWN notification code. The structure is identical to the TV_KEYDOWN structure, but it has been renamed to follow current naming conventions.

NMTVSTATEIMAGECHANGING

Contains information about an NM_TVSTATEIMAGECHANGING notification code.

NMUPDOWN

Contains information specific to up-down control notification messages. It is identical to and replaces the NM_UPDOWN structure.

NMVIEWCHANGE

Stores information required to process the MCN_VIEWCHANGE notification code.

PBRANGE

Contains information about the high and low limits of a progress bar control. This structure is used with the PBM_GETRANGE message.

RBHITTESTINFO

Contains information specific to a hit test operation. This structure is used with the RB_HITTEST message.

REBARBANDINFOA

Contains information that defines a band in a rebar control.

REBARBANDINFOW

Contains information that defines a band in a rebar control.

REBARINFO

Contains information that describes rebar control characteristics.

TASKDIALOG_BUTTON

The TASKDIALOG_BUTTON structure contains information used to display a button in a task dialog. The TASKDIALOGCONFIG structure uses this structure.

TASKDIALOGCONFIG

The TASKDIALOGCONFIG structure contains information used to display a task dialog. The TaskDialogIndirect function uses this structure.

TBADDBITMAP

Adds a bitmap that contains button images to a toolbar.

TBBUTTON

Contains information about a button in a toolbar.

TBBUTTONINFOA

Contains or receives information for a specific button in a toolbar.

TBBUTTONINFOW

Contains or receives information for a specific button in a toolbar.

TBINSERTMARK

Contains information on the insertion mark in a toolbar control.

TBMETRICS

Defines the metrics of a toolbar that are used to shrink or expand toolbar items.

TBREPLACEBITMAP

Used with the TB_REPLACEBITMAP message to replace one toolbar bitmap with another.

TBSAVEPARAMSA

Specifies the location in the registry where the TB_SAVERESTORE message stores and retrieves information about the state of a toolbar.

TBSAVEPARAMSW

Specifies the location in the registry where the TB_SAVERESTORE message stores and retrieves information about the state of a toolbar.

TCHITTESTINFO

Contains information about a hit test. This structure supersedes the TC_HITTESTINFO structure.

TCITEMA

Specifies or receives the attributes of a tab item. It is used with the TCM_INSERTITEM, TCM_GETITEM, and TCM_SETITEM messages. This structure supersedes the TC_ITEM structure.

TCITEMHEADERA

Specifies or receives the attributes of a tab. It is used with the TCM_INSERTITEM, TCM_GETITEM, and TCM_SETITEM messages. This structure supersedes the TC_ITEMHEADER structure.

TCITEMHEADERW

Specifies or receives the attributes of a tab. It is used with the TCM_INSERTITEM, TCM_GETITEM, and TCM_SETITEM messages. This structure supersedes the TC_ITEMHEADER structure.

TCITEMW

Specifies or receives the attributes of a tab item. It is used with the TCM_INSERTITEM, TCM_GETITEM, and TCM_SETITEM messages. This structure supersedes the TC_ITEM structure.

TTGETTITLE

Provides information about the title of a tooltip control.

TTHITTESTINFOA

Contains information that a tooltip control uses to determine whether a point is in the bounding rectangle of the specified tool. If the point is in the rectangle, the structure receives information about the tool.

TTHITTESTINFOW

Contains information that a tooltip control uses to determine whether a point is in the bounding rectangle of the specified tool. If the point is in the rectangle, the structure receives information about the tool.

TTTOOLINFOA

The TOOLINFO structure contains information about a tool in a tooltip control.

TTTOOLINFOW

The TOOLINFO structure contains information about a tool in a tooltip control.

TVGETITEMPARTRECTINFO

Contains information for identifying the "hit zone" for a specified part of a tree item. The structure is used with the TVM_GETITEMPARTRECT message and the TreeView_GetItemPartRect macro.

TVHITTESTINFO

Contains information used to determine the location of a point relative to a tree-view control.

TVINSERTSTRUCTA

Contains information used to add a new item to a tree-view control. This structure is used with the TVM_INSERTITEM message. The structure is identical to the TV_INSERTSTRUCT structure, but it has been renamed to follow current naming conventions.

TVINSERTSTRUCTW

Contains information used to add a new item to a tree-view control. This structure is used with the TVM_INSERTITEM message. The structure is identical to the TV_INSERTSTRUCT structure, but it has been renamed to follow current naming conventions.

TVITEMA

Specifies or receives attributes of a tree-view item. This structure is identical to the TV_ITEM structure, but it has been renamed to follow current naming conventions. New applications should use this structure.

TVITEMEXA Specifies or receives attributes of a tree-view item. This structure is an enhancement to the TVITEM structure. New applications should use this structure where appropriate.
TVITEMEXW Specifies or receives attributes of a tree-view item. This structure is an enhancement to the TVITEM structure. New applications should use this structure where appropriate.
TVITEMW Specifies or receives attributes of a tree-view item. This structure is identical to the TV_ITEM structure, but it has been renamed to follow current naming conventions. New applications should use this structure.
TVSORTCB Contains information used to sort child items in a tree-view control. This structure is used with the TVM_SORTCHILDRENCB message. This structure is identical to the TV_SORTCB structure, but it has been renamed to follow current naming conventions.
UDACCEL Contains acceleration information for an up-down control.

Enumerations

EC_ENDOFLINE Indicates the end of line character used by an edit control.
EC_SEARCHWEB_ENTRYPOINT Defines constants that indicate the entry point of a web search.

_TrackMouseEvent function (commctrl.h)

7/6/2022 • 2 minutes to read • Edit Online

Posts messages when the mouse pointer leaves a window or hovers over a window for a specified amount of time. This function calls TrackMouseEvent if it exists, otherwise it emulates it.

Syntax

```
BOOL _TrackMouseEvent(
  [in, out] LPTRACKMOUSEEVENT lpEventTrack
);
```

Parameters

`[in, out] lpEventTrack`

Type: **LPTRACKMOUSEEVENT**

A pointer to a TRACKMOUSEEVENT structure that contains tracking information.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, return value is zero.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	commctrl.h
Library	Comctl32.lib
DLL	Comctl32.dll

See also

Conceptual

Mouse Input

Other Resources

Reference

SystemParametersInfo

TRACKMOUSEEVENT

TrackMouseEvent

winuser.h header

7/6/2022 • 79 minutes to read • Edit Online

This header is used by multiple technologies. For more information, see:

- Data Exchange
- Desktop Window Manager (DWM)
- Developer Notes
- Dialog Boxes
- Display Devices Reference
- High DPI
- Input Feedback Configuration
- Input Source Identification
- Internationalization for Windows Applications
- Keyboard and Mouse Input
- Menus and Other Resources
- Mobile Device Management Settings Provider
- Pointer Device Input Stack
- Pointer Input Messages and Notifications
- Remote Desktop Services
- Security and Identity
- System Services
- The Windows Shell
- Touch Hit Testing
- Touch Injection
- Touch Input
- Window Stations and Desktops
- Windows Accessibility Features
- Windows and Messages
- Windows Controls
- Windows GDI

winuser.h contains the following programming interfaces:

Functions

ActivateKeyboardLayout Sets the input locale identifier (formerly called the keyboard layout handle) for the calling thread or the current process. The input locale identifier specifies a locale as well as the physical layout of the keyboard.
AddClipboardFormatListener Places the given window in the system-maintained clipboard format listener list.

AdjustWindowRect	Calculates the required size of the window rectangle, based on the desired client-rectangle size. The window rectangle can then be passed to the CreateWindow function to create a window whose client area is the desired size.
AdjustWindowRectEx	Calculates the required size of the window rectangle, based on the desired size of the client rectangle. The window rectangle can then be passed to the CreateWindowEx function to create a window whose client area is the desired size.
AdjustWindowRectExForDpi	Calculates the required size of the window rectangle, based on the desired size of the client rectangle and the provided DPI.
AllowSetForegroundWindow	Enables the specified process to set the foreground window using the SetForegroundWindow function. The calling process must already be able to set the foreground window. For more information, see Remarks later in this topic.
AnimateWindow	Enables you to produce special effects when showing or hiding windows. There are four types of animation:_roll, slide, collapse or expand, and alpha-blended fade.
AnyPopup	Indicates whether an owned, visible, top-level pop-up, or overlapped window exists on the screen. The function searches the entire screen, not just the calling application's client area.
AppendMenuA	Appends a new item to the end of the specified menu bar, drop-down menu, submenu, or shortcut menu. You can use this function to specify the content, appearance, and behavior of the menu item.
AppendMenuW	Appends a new item to the end of the specified menu bar, drop-down menu, submenu, or shortcut menu. You can use this function to specify the content, appearance, and behavior of the menu item.
AreDpiAwarenessContextsEqual	Determines whether two DPI_AWARENESS_CONTEXT values are identical.
ArrangeIconicWindows	Arranges all the minimized (iconic) child windows of the specified parent window.
AttachThreadInput	Attaches or detaches the input processing mechanism of one thread to that of another thread.
BeginDeferWindowPos	Allocates memory for a multiple-window- position structure and returns the handle to the structure.

BeginPaint	The BeginPaint function prepares the specified window for painting and fills a PAINTSTRUCT structure with information about the painting.
BlockInput	Blocks keyboard and mouse input events from reaching applications.
BringWindowToTop	Brings the specified window to the top of the Z order. If the window is a top-level window, it is activated. If the window is a child window, the top-level parent window associated with the child window is activated.
BroadcastSystemMessage	Sends a message to the specified recipients.
BroadcastSystemMessageA	Sends a message to the specified recipients.
BroadcastSystemMessageExA	Sends a message to the specified recipients.
BroadcastSystemMessageExW	Sends a message to the specified recipients.
BroadcastSystemMessageW	Sends a message to the specified recipients.
CalculatePopupWindowPosition	Calculates an appropriate pop-up window position using the specified anchor point, pop-up window size, flags, and the optional exclude rectangle.
CallMsgFilterA	Passes the specified message and hook code to the hook procedures associated with the WH_SYSMSGFILTER and WH_MSGFILTER hooks.
CallMsgFilterW	Passes the specified message and hook code to the hook procedures associated with the WH_SYSMSGFILTER and WH_MSGFILTER hooks.
CallNextHookEx	Passes the hook information to the next hook procedure in the current hook chain. A hook procedure can call this function either before or after processing the hook information.

CallWindowProcA	Passes message information to the specified window procedure.
CallWindowProcW	Passes message information to the specified window procedure.
CascadeWindows	Cascades the specified child windows of the specified parent window.
ChangeClipboardChain	Removes a specified window from the chain of clipboard viewers.
ChangeDisplaySettingsA	The ChangeDisplaySettings function changes the settings of the default display device to the specified graphics mode.
ChangeDisplaySettingsExA	The ChangeDisplaySettingsEx function changes the settings of the specified display device to the specified graphics mode.
ChangeDisplaySettingsExW	The ChangeDisplaySettingsEx function changes the settings of the specified display device to the specified graphics mode.
ChangeDisplaySettingsW	The ChangeDisplaySettings function changes the settings of the default display device to the specified graphics mode.
ChangeWindowMessageFilter	Adds or removes a message from the User Interface Privilege Isolation (UIPI) message filter.
ChangeWindowMessageFilterEx	Modifies the User Interface Privilege Isolation (UIPI) message filter for a specified window.
CharLowerA	Converts a character string or a single character to lowercase. If the operand is a character string, the function converts the characters in place.
CharLowerBuffA	Converts uppercase characters in a buffer to lowercase characters. The function converts the characters in place.
CharLowerBuffW	Converts uppercase characters in a buffer to lowercase characters. The function converts the characters in place.

CharLowerW	Converts a character string or a single character to lowercase. If the operand is a character string, the function converts the characters in place.
CharNextA	Retrieves a pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.
CharNextExA	Retrieves the pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.
CharNextW	Retrieves a pointer to the next character in a string. This function can handle strings consisting of either single- or multi-byte characters.
CharPrevA	Retrieves a pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.
CharPrevExA	Retrieves the pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.
CharPrevW	Retrieves a pointer to the preceding character in a string. This function can handle strings consisting of either single- or multi-byte characters.
CharToOemA	Translates a string into the OEM-defined character set.Warning Do not use.
CharToOemBuffA	Translates a specified number of characters in a string into the OEM-defined character set.
CharToOemBuffW	Translates a specified number of characters in a string into the OEM-defined character set.
CharToOemW	Translates a string into the OEM-defined character set.Warning Do not use.
CharUpperA	Converts a character string or a single character to uppercase. If the operand is a character string, the function converts the characters in place.

CharUpperBuffA	Converts lowercase characters in a buffer to uppercase characters. The function converts the characters in place.
CharUpperBuffW	Converts lowercase characters in a buffer to uppercase characters. The function converts the characters in place.
CharUpperW	Converts a character string or a single character to uppercase. If the operand is a character string, the function converts the characters in place.
CheckDlgButton	Changes the check state of a button control.
CheckMenuItem	Sets the state of the specified menu item's check-mark attribute to either selected or clear.
CheckMenuRadioItem	Checks a specified menu item and makes it a radio item. At the same time, the function clears all other menu items in the associated group and clears the radio-item type flag for those items.
CheckRadioButton	Adds a check mark to (checks) a specified radio button in a group and removes a check mark from (clears) all other radio buttons in the group.
ChildWindowFromPoint	Determines which, if any, of the child windows belonging to a parent window contains the specified point. The search is restricted to immediate child windows. Grandchildren, and deeper descendant windows are not searched.
ChildWindowFromPointEx	Determines which, if any, of the child windows belonging to the specified parent window contains the specified point.
ClientToScreen	The ClientToScreen function converts the client-area coordinates of a specified point to screen coordinates.
ClipCursor	Confines the cursor to a rectangular area on the screen.
CloseClipboard	Closes the clipboard.
CloseDesktop	Closes an open handle to a desktop object.

CloseGestureInfoHandle	Closes resources associated with a gesture information handle.
CloseTouchInputHandle	Closes a touch input handle, frees process memory associated with it, and invalidates the handle.
CloseWindow	Minimizes (but does not destroy) the specified window.
CloseWindowStation	Closes an open window station handle.
CopyAcceleratorTableA	Copies the specified accelerator table. This function is used to obtain the accelerator-table data that corresponds to an accelerator-table handle, or to determine the size of the accelerator-table data.
CopyAcceleratorTableW	Copies the specified accelerator table. This function is used to obtain the accelerator-table data that corresponds to an accelerator-table handle, or to determine the size of the accelerator-table data.
CopyCursor	Copies the specified cursor.
CopyIcon	Copies the specified icon from another module to the current module.
CopyImage	Creates a new image (icon, cursor, or bitmap) and copies the attributes of the specified image to the new one. If necessary, the function stretches the bits to fit the desired size of the new image.
CopyRect	The CopyRect function copies the coordinates of one rectangle to another.
CountClipboardFormats	Retrieves the number of different data formats currently on the clipboard.
CreateAcceleratorTableA	Creates an accelerator table.
CreateAcceleratorTableW	Creates an accelerator table.

CreateCaret	Creates a new shape for the system caret and assigns ownership of the caret to the specified window. The caret shape can be a line, a block, or a bitmap.
CreateCursor	Creates a cursor having the specified size, bit patterns, and hot spot.
CreateDesktopA	Creates a new desktop, associates it with the current window station of the calling process, and assigns it to the calling thread.
CreateDesktopExA	Creates a new desktop with the specified heap, associates it with the current window station of the calling process, and assigns it to the calling thread.
CreateDesktopExW	Creates a new desktop with the specified heap, associates it with the current window station of the calling process, and assigns it to the calling thread.
CreateDesktopW	Creates a new desktop, associates it with the current window station of the calling process, and assigns it to the calling thread.
CreateDialogA	Creates a modeless dialog box from a dialog box template resource. The CreateDialog macro uses the CreateDialogParam function.
CreateDialogIndirectA	Creates a modeless dialog box from a dialog box template in memory. The CreateDialogIndirect macro uses the CreateDialogIndirectParam function.
CreateDialogIndirectParamA	Creates a modeless dialog box from a dialog box template in memory.
CreateDialogIndirectParamW	Creates a modeless dialog box from a dialog box template in memory.
CreateDialogIndirectW	Creates a modeless dialog box from a dialog box template in memory. The CreateDialogIndirect macro uses the CreateDialogIndirectParam function.
CreateDialogParamA	Creates a modeless dialog box from a dialog box template resource.

CreateDialogParamW	Creates a modeless dialog box from a dialog box template resource.
CreateDialogW	Creates a modeless dialog box from a dialog box template resource. The CreateDialog macro uses the CreateDialogParam function.
CreateIcon	Creates an icon that has the specified size, colors, and bit patterns.
CreateIconFromResource	Creates an icon or cursor from resource bits describing the icon.
CreateIconFromResourceEx	Creates an icon or cursor from resource bits describing the icon.
CreateIconIndirect	Creates an icon or cursor from an ICONINFO structure.
CreateMDIWindowA	Creates a multiple-document interface (MDI) child window.
CreateMDIWindowW	Creates a multiple-document interface (MDI) child window.
CreateMenu	Creates a menu. The menu is initially empty, but it can be filled with menu items by using the InsertMenuItem, AppendMenu, and InsertMenu functions.
CreatePopupMenu	Creates a drop-down menu, submenu, or shortcut menu.
CreateSyntheticPointerDevice	Configures the pointer injection device for the calling application, and initializes the maximum number of simultaneous pointers that the app can inject.
CreateWindowA	Creates an overlapped, pop-up, or child window.
CreateWindowExA	Creates an overlapped, pop-up, or child window with an extended window style; otherwise, this function is identical to the CreateWindow function.

CreateWindowExW	Creates an overlapped, pop-up, or child window with an extended window style; otherwise, this function is identical to the CreateWindow function.
CreateWindowStationA	Creates a window station object, associates it with the calling process, and assigns it to the current session.
CreateWindowStationW	Creates a window station object, associates it with the calling process, and assigns it to the current session.
CreateWindowW	Creates an overlapped, pop-up, or child window.
DefDlgProcA	Calls the default dialog box window procedure to provide default processing for any window messages that a dialog box with a private window class does not process.
DefDlgProcW	Calls the default dialog box window procedure to provide default processing for any window messages that a dialog box with a private window class does not process.
DeferWindowPos	Updates the specified multiple-window � position structure for the specified window.
DefFrameProcA	Provides default processing for any window messages that the window procedure of a multiple-document interface (MDI) frame window does not process.
DefFrameProcW	Provides default processing for any window messages that the window procedure of a multiple-document interface (MDI) frame window does not process.
DefMDIChildProcA	Provides default processing for any window message that the window procedure of a multiple-document interface (MDI) child window does not process.
DefMDIChildProcW	Provides default processing for any window message that the window procedure of a multiple-document interface (MDI) child window does not process.
DefRawInputProc	Verifies that the size of the RAWINPUTHEADER structure is correct.

DefWindowProcA	Calls the default window procedure to provide default processing for any window messages that an application does not process.
DefWindowProcW	Calls the default window procedure to provide default processing for any window messages that an application does not process.
DeleteMenu	Deletes an item from the specified menu. If the menu item opens a menu or submenu, this function destroys the handle to the menu or submenu and frees the memory used by the menu or submenu.
DeregisterShellHookWindow	Unregisters a specified Shell window that is registered to receive Shell hook messages.
DestroyAcceleratorTable	Destroys an accelerator table.
DestroyCaret	Destroys the caret's current shape, frees the caret from the window, and removes the caret from the screen.
DestroyCursor	Destroys a cursor and frees any memory the cursor occupied. Do not use this function to destroy a shared cursor.
DestroyIcon	Destroys an icon and frees any memory the icon occupied.
DestroyMenu	Destroys the specified menu and frees any memory that the menu occupies.
DestroySyntheticPointerDevice	Destroys the specified pointer injection device.
DestroyWindow	Destroys the specified window.
DialogBoxA	Creates a modal dialog box from a dialog box template resource. DialogBox does not return control until the specified callback function terminates the modal dialog box by calling the EndDialog function.
DialogBoxIndirectA	Creates a modal dialog box from a dialog box template in memory. DialogBoxIndirect does not return control until the specified callback function terminates the modal dialog box by calling the EndDialog function.

DialogBoxIndirectParamA

Creates a modal dialog box from a dialog box template in memory.

DialogBoxIndirectParamW

Creates a modal dialog box from a dialog box template in memory.

DialogBoxIndirectW

Creates a modal dialog box from a dialog box template in memory. DialogBoxIndirect does not return control until the specified callback function terminates the modal dialog box by calling the EndDialog function.

DialogBoxParamA

Creates a modal dialog box from a dialog box template resource.

DialogBoxParamW

Creates a modal dialog box from a dialog box template resource.

DialogBoxW

Creates a modal dialog box from a dialog box template resource. DialogBox does not return control until the specified callback function terminates the modal dialog box by calling the EndDialog function.

DisableProcessWindowsGhosting

Disables the window ghosting feature for the calling GUI process. Window ghosting is a Windows Manager feature that lets the user minimize, move, or close the main window of an application that is not responding.

DispatchMessage

Dispatches a message to a window procedure. It is typically used to dispatch a message retrieved by the GetMessage function.

DispatchMessageA

Dispatches a message to a window procedure. It is typically used to dispatch a message retrieved by the GetMessage function.

DispatchMessageW

Dispatches a message to a window procedure. It is typically used to dispatch a message retrieved by the GetMessage function.

DisplayConfigGetDeviceInfo

The DisplayConfigGetDeviceInfo function retrieves display configuration information about the device.

DisplayConfigSetDeviceInfo

The DisplayConfigSetDeviceInfo function sets the properties of a target.

DlgDirListA

Replaces the contents of a list box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list can optionally include mapped drives.

DlgDirListComboBoxA	Replaces the contents of a combo box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list of names can include mapped drive letters.
DlgDirListComboBoxW	Replaces the contents of a combo box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list of names can include mapped drive letters.
DlgDirListW	Replaces the contents of a list box with the names of the subdirectories and files in a specified directory. You can filter the list of names by specifying a set of file attributes. The list can optionally include mapped drives.
DlgDirSelectComboBoxExA	Retrieves the current selection from a combo box filled by using the DlgDirListComboBox function. The selection is interpreted as a drive letter, a file, or a directory name.
DlgDirSelectComboBoxExW	Retrieves the current selection from a combo box filled by using the DlgDirListComboBox function. The selection is interpreted as a drive letter, a file, or a directory name.
DlgDirSelectExA	Retrieves the current selection from a single-selection list box. It assumes that the list box has been filled by the DlgDirList function and that the selection is a drive letter, filename, or directory name.
DlgDirSelectExW	Retrieves the current selection from a single-selection list box. It assumes that the list box has been filled by the DlgDirList function and that the selection is a drive letter, filename, or directory name.
DragDetect	Captures the mouse and tracks its movement until the user releases the left button, presses the ESC key, or moves the mouse outside the drag rectangle around the specified point.
DrawAnimatedRects	Animates the caption of a window to indicate the opening of an icon or the minimizing or maximizing of a window.
DrawCaption	The DrawCaption function draws a window caption.
DrawEdge	The DrawEdge function draws one or more edges of rectangle.
DrawFocusRect	The DrawFocusRect function draws a rectangle in the style used to indicate that the rectangle has the focus.

DrawFrameControl	The DrawFrameControl function draws a frame control of the specified type and style.
DrawIcon	Draws an icon or cursor into the specified device context.
DrawIconEx	Draws an icon or cursor into the specified device context, performing the specified raster operations, and stretching or compressing the icon or cursor as specified.
DrawMenuBar	Redraws the menu bar of the specified window. If the menu bar changes after the system has created the window, this function must be called to draw the changed menu bar.
DrawStateA	The DrawState function displays an image and applies a visual effect to indicate a state, such as a disabled or default state.
DrawStateW	The DrawState function displays an image and applies a visual effect to indicate a state, such as a disabled or default state.
DrawText	The DrawText function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).
DrawTextA	The DrawText function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).
DrawTextExA	The DrawTextEx function draws formatted text in the specified rectangle.
DrawTextExW	The DrawTextEx function draws formatted text in the specified rectangle.
DrawTextW	The DrawText function draws formatted text in the specified rectangle. It formats the text according to the specified method (expanding tabs, justifying characters, breaking lines, and so forth).
EmptyClipboard	Empties the clipboard and frees handles to data in the clipboard. The function then assigns ownership of the clipboard to the window that currently has the clipboard open.

EnableMenuItem	Enables, disables, or grays the specified menu item.
EnableMouseInPointer	Enables the mouse to act as a pointer input device and send WM_POINTER messages.
EnableNonClientDpiScaling	In high-DPI displays, enables automatic display scaling of the non-client area portions of the specified top-level window. Must be called during the initialization of that window.
EnableScrollBar	The EnableScrollBar function enables or disables one or both scroll bar arrows.
EnableWindow	Enables or disables mouse and keyboard input to the specified window or control. When input is disabled, the window does not receive input such as mouse clicks and key presses. When input is enabled, the window receives all input.
EndDeferWindowPos	Simultaneously updates the position and size of one or more windows in a single screen-refreshing cycle.
EndDialog	Destroys a modal dialog box, causing the system to end any processing for the dialog box.
EndMenu	Ends the calling thread's active menu.
EndPaint	The EndPaint function marks the end of painting in the specified window. This function is required for each call to the BeginPaint function, but only after painting is complete.
EndTask	Forcibly closes the specified window.
EnumChildWindows	Enumerates the child windows that belong to the specified parent window by passing the handle to each child window, in turn, to an application-defined callback function.
EnumClipboardFormats	Enumerates the data formats currently available on the clipboard.
EnumDesktopsA	Enumerates all desktops associated with the specified window station of the calling process. The function passes the name of each desktop, in turn, to an application-defined callback function.

EnumDesktopsW

Enumerates all desktops associated with the specified window station of the calling process. The function passes the name of each desktop, in turn, to an application-defined callback function.

EnumDesktopWindows

Enumerates all top-level windows associated with the specified desktop. It passes the handle to each window, in turn, to an application-defined callback function.

EnumDisplayDevicesA

The EnumDisplayDevices function lets you obtain information about the display devices in the current session.

EnumDisplayDevicesW

The EnumDisplayDevices function lets you obtain information about the display devices in the current session.

EnumDisplayMonitors

The EnumDisplayMonitors function enumerates display monitors (including invisible pseudo-monitors associated with the mirroring drivers) that intersect a region formed by the intersection of a specified clipping rectangle and the visible region of a device context. EnumDisplayMonitors calls an application-defined MonitorEnumProc callback function once for each monitor that is enumerated. Note that GetSystemMetrics (SM_CMONITORS) counts only the display monitors.

EnumDisplaySettingsA

The EnumDisplaySettings function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes of a display device, make a series of calls to this function.

EnumDisplaySettingsExA

The EnumDisplaySettingsEx function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes for a display device, make a series of calls to this function.

EnumDisplaySettingsExW

The EnumDisplaySettingsEx function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes for a display device, make a series of calls to this function.

EnumDisplaySettingsW

The EnumDisplaySettings function retrieves information about one of the graphics modes for a display device. To retrieve information for all the graphics modes of a display device, make a series of calls to this function.

EnumPropsA

Enumerates all entries in the property list of a window by passing them, one by one, to the specified callback function. EnumProps continues until the last entry is enumerated or the callback function returns FALSE.

EnumPropsExA

Enumerates all entries in the property list of a window by passing them, one by one, to the specified callback function. EnumPropsEx continues until the last entry is enumerated or the callback function returns FALSE.

EnumPropsExW	Enumerates all entries in the property list of a window by passing them, one by one, to the specified callback function. EnumPropsEx continues until the last entry is enumerated or the callback function returns FALSE.
EnumPropsW	Enumerates all entries in the property list of a window by passing them, one by one, to the specified callback function. EnumProps continues until the last entry is enumerated or the callback function returns FALSE.
EnumThreadWindows	Enumerates all nonchild windows associated with a thread by passing the handle to each window, in turn, to an application-defined callback function.
EnumWindows	Enumerates all top-level windows on the screen by passing the handle to each window, in turn, to an application-defined callback function. EnumWindows continues until the last top-level window is enumerated or the callback function returns FALSE.
EnumWindowStationsA	Enumerates all window stations in the current session. The function passes the name of each window station, in turn, to an application-defined callback function.
EnumWindowStationsW	Enumerates all window stations in the current session. The function passes the name of each window station, in turn, to an application-defined callback function.
EqualRect	The EqualRect function determines whether the two specified rectangles are equal by comparing the coordinates of their upper-left and lower-right corners.
EvaluateProximityToPolygon	Returns the score of a polygon as the probable touch target (compared to all other polygons that intersect the touch contact area) and an adjusted touch point within the polygon.
EvaluateProximityToRect	Returns the score of a rectangle as the probable touch target, compared to all other rectangles that intersect the touch contact area, and an adjusted touch point within the rectangle.
ExcludeUpdateRgn	The ExcludeUpdateRgn function prevents drawing within invalid areas of a window by excluding an updated region in the window from a clipping region.
ExitWindows	Calls the ExitWindowsEx function to log off the interactive user.

ExitWindowsEx	Logs off the interactive user, shuts down the system, or shuts down and restarts the system.
FillRect	The FillRect function fills a rectangle by using the specified brush. This function includes the left and top borders, but excludes the right and bottom borders of the rectangle.
FindWindowA	Retrieves a handle to the top-level window whose class name and window name match the specified strings. This function does not search child windows. This function does not perform a case-sensitive search.
FindWindowExA	Retrieves a handle to a window whose class name and window name match the specified strings. The function searches child windows, beginning with the one following the specified child window. This function does not perform a case-sensitive search.
FindWindowExW	Retrieves a handle to a window whose class name and window name match the specified strings. The function searches child windows, beginning with the one following the specified child window. This function does not perform a case-sensitive search.
FindWindowW	Retrieves a handle to the top-level window whose class name and window name match the specified strings. This function does not search child windows. This function does not perform a case-sensitive search.
FlashWindow	Flashes the specified window one time. It does not change the active state of the window.
FlashWindowEx	Flashes the specified window. It does not change the active state of the window.
FrameRect	The FrameRect function draws a border around the specified rectangle by using the specified brush. The width and height of the border are always one logical unit.
GET_APPCOMMAND_LPARAM	Retrieves the application command from the specified LPARAM value.
GET_DEVICE_LPARAM	Retrieves the input device type from the specified LPARAM value.
GET_FLAGS_LPARAM	Retrieves the state of certain virtual keys from the specified LPARAM value.

GET_KEYSTATE_LPARAM	Retrieves the state of certain virtual keys from the specified LPARAM value.
GET_KEYSTATE_WPARAM	Retrieves the state of certain virtual keys from the specified WPARAM value.
GET_NCHITTEST_WPARAM	Retrieves the hit-test value from the specified WPARAM value.
GET_POINTERID_WPARAM	Retrieves the pointer ID using the specified value.
GET_RAWINPUT_CODE_WPARAM	Retrieves the input code from wParam in WM_INPUT.
GET_WHEEL_DELTA_WPARAM	Retrieves the wheel-delta value from the specified WPARAM value.
GET_XBUTTON_WPARAM	Retrieves the state of certain buttons from the specified WPARAM value.
GetActiveWindow	Retrieves the window handle to the active window attached to the calling thread's message queue.
GetAltTabInfoA	Retrieves status information for the specified window if it is the application-switching (ALT+TAB) window.
GetAltTabInfoW	Retrieves status information for the specified window if it is the application-switching (ALT+TAB) window.
GetAncestor	Retrieves the handle to the ancestor of the specified window.
GetAsyncKeyState	Determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to GetAsyncKeyState.
GetAutoRotationState	Retrieves an AR_STATE value containing the state of screen auto-rotation for the system, for example whether auto-rotation is supported, and whether it is enabled by the user.

GetAwarenessFromDpiAwarenessContext	Retrieves the DPI_AWARENESS value from a DPI_AWARENESS_CONTEXT.
GetCapture	Retrieves a handle to the window (if any) that has captured the mouse. Only one window at a time can capture the mouse; this window receives mouse input whether or not the cursor is within its borders.
GetCaretBlinkTime	Retrieves the time required to invert the caret's pixels. The user can set this value.
GetCaretPos	Copies the caret's position to the specified POINT structure.
GetCIMSSM	Retrieves the source of the input message (GetCurrentInputMessageSourceInSendMessage).
GetClassInfoA	Retrieves information about a window class.
GetClassInfoExA	Retrieves information about a window class, including a handle to the small icon associated with the window class. The GetClassInfo function does not retrieve a handle to the small icon.
GetClassInfoExW	Retrieves information about a window class, including a handle to the small icon associated with the window class. The GetClassInfo function does not retrieve a handle to the small icon.
GetClassInfoW	Retrieves information about a window class.
GetClassLongA	Retrieves the specified 32-bit (DWORD) value from the WNDCLASSEX structure associated with the specified window.
GetClassLongPtrA	Retrieves the specified value from the WNDCLASSEX structure associated with the specified window.
GetClassLongPtrW	Retrieves the specified value from the WNDCLASSEX structure associated with the specified window.
GetClassLongW	Retrieves the specified 32-bit (DWORD) value from the WNDCLASSEX structure associated with the specified window.

GetClassName	Retrieves the name of the class to which the specified window belongs.
GetClassNameA	Retrieves the name of the class to which the specified window belongs.
GetClassNameW	Retrieves the name of the class to which the specified window belongs.
GetClassWord	Retrieves the 16-bit (WORD) value at the specified offset into the extra class memory for the window class to which the specified window belongs.
GetClientRect	Retrieves the coordinates of a window's client area.
GetClipboardData	Retrieves data from the clipboard in a specified format. The clipboard must have been opened previously.
GetClipboardFormatNameA	Retrieves from the clipboard the name of the specified registered format. The function copies the name to the specified buffer.
GetClipboardFormatNameW	Retrieves from the clipboard the name of the specified registered format. The function copies the name to the specified buffer.
GetClipboardOwner	Retrieves the window handle of the current owner of the clipboard.
GetClipboardSequenceNumber	Retrieves the clipboard sequence number for the current window station.
GetClipboardViewer	Retrieves the handle to the first window in the clipboard viewer chain.
GetClipCursor	Retrieves the screen coordinates of the rectangular area to which the cursor is confined.
GetComboBoxInfo	Retrieves information about the specified combo box.

GetCurrentInputMessageSource	Retrieves the source of the input message.
GetCursor	Retrieves a handle to the current cursor.
GetCursorInfo	Retrieves information about the global cursor.
GetCursorPos	Retrieves the position of the mouse cursor, in screen coordinates.
GetDC	The GetDC function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen.
GetDCEx	The GetDCEx function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen.
GetDesktopWindow	Retrieves a handle to the desktop window. The desktop window covers the entire screen. The desktop window is the area on top of which other windows are painted.
GetDialogBaseUnits	Retrieves the system's dialog base units, which are the average width and height of characters in the system font.
GetDialogControlDpiChangeBehavior	Retrieves and per-monitor DPI scaling behavior overrides of a child window in a dialog.
GetDialogDpiChangeBehavior	Returns the flags that might have been set on a given dialog by an earlier call to SetDialogDpiChangeBehavior.
GetDisplayAutoRotationPreferences	Retrieves the screen auto-rotation preferences for the current process.
GetDisplayAutoRotationPreferencesByProcessId	Retrieves the screen auto-rotation preferences for the process indicated by the dwProcessId parameter.
GetDisplayConfigBufferSizes	The GetDisplayConfigBufferSizes function retrieves the size of the buffers that are required to call the QueryDisplayConfig function.

GetDlgCtrlID	Retrieves the identifier of the specified control.
GetDlgItem	Retrieves a handle to a control in the specified dialog box.
GetDlgItemInt	Translates the text of a specified control in a dialog box into an integer value.
GetDlgItemTextA	Retrieves the title or text associated with a control in a dialog box.
GetDlgItemTextW	Retrieves the title or text associated with a control in a dialog box.
GetDoubleClickTime	Retrieves the current double-click time for the mouse.
GetDpiForSystem	Returns the system DPI.
GetDpiForWindow	Returns the dots per inch (dpi) value for the specified window.
GetDpiFromDpiAwarenessContext	Retrieves the DPI from a given DPI_AWARENESS_CONTEXT handle. This enables you to determine the DPI of a thread without needed to examine a window created within that thread.
GetFocus	Retrieves the handle to the window that has the keyboard focus, if the window is attached to the calling thread's message queue.
GetForegroundWindow	Retrieves a handle to the foreground window (the window with which the user is currently working). The system assigns a slightly higher priority to the thread that creates the foreground window than it does to other threads.
GetGestureConfig	Retrieves the configuration for which Windows Touch gesture messages are sent from a window.
GetGestureExtraArgs	Retrieves additional information about a gesture from its GESTUREINFO handle.

GetGestureInfo	Retrieves a GESTUREINFO structure given a handle to the gesture information.
GetGuiResources	Retrieves the count of handles to graphical user interface (GUI) objects in use by the specified process.
GetGUIThreadInfo	Retrieves information about the active window or a specified GUI thread.
GetIconInfo	Retrieves information about the specified icon or cursor.
GetIconInfoExA	Retrieves information about the specified icon or cursor. GetIconInfoEx extends GetIconInfo by using the newer ICONINFOEX structure.
GetIconInfoExW	Retrieves information about the specified icon or cursor. GetIconInfoEx extends GetIconInfo by using the newer ICONINFOEX structure.
GetInputState	Determines whether there are mouse-button or keyboard messages in the calling thread's message queue.
GetKBCodePage	Retrieves the current code page.
GetKeyboardLayout	Retrieves the active input locale identifier (formerly called the keyboard layout).
GetKeyboardLayoutList	Retrieves the input locale identifiers (formerly called keyboard layout handles) corresponding to the current set of input locales in the system. The function copies the identifiers to the specified buffer.
GetKeyboardLayoutNameA	Retrieves the name of the active input locale identifier (formerly called the keyboard layout) for the system.
GetKeyboardLayoutNameW	Retrieves the name of the active input locale identifier (formerly called the keyboard layout) for the system.
GetKeyboardState	Copies the status of the 256 virtual keys to the specified buffer.

GetKeyboardType	Retrieves information about the current keyboard.
GetKeyNameTextA	Retrieves a string that represents the name of a key.
GetKeyNameTextW	Retrieves a string that represents the name of a key.
GetKeyState	Retrieves the status of the specified virtual key. The status specifies whether the key is up, down, or toggled (on, off�alternating each time the key is pressed).
GetLastActivePopup	Determines which pop-up window owned by the specified window was most recently active.
GetLastInputInfo	Retrieves the time of the last input event.
GetLayeredWindowAttributes	Retrieves the opacity and transparency color key of a layered window.
GetListBoxInfo	Retrieves the number of items per column in a specified list box.
GetMenu	Retrieves a handle to the menu assigned to the specified window.
GetMenuBarInfo	Retrieves information about the specified menu bar.
GetMenuCheckMarkDimensions	Retrieves the dimensions of the default check-mark bitmap.
GetMenuContextHelpId	Retrieves the Help context identifier associated with the specified menu.
GetMenuDefaultItem	Determines the default menu item on the specified menu.

GetMenuInfo	Retrieves information about a specified menu.
GetMenuItemCount	Determines the number of items in the specified menu.
GetMenuItemID	Retrieves the menu item identifier of a menu item located at the specified position in a menu.
GetMenuItemInfoA	Retrieves information about a menu item.
GetMenuItemInfoW	Retrieves information about a menu item.
GetMenuItemRect	Retrieves the bounding rectangle for the specified menu item.
GetMenuState	Retrieves the menu flags associated with the specified menu item.
GetMenuStringA	Copies the text string of the specified menu item into the specified buffer.
GetMenuStringW	Copies the text string of the specified menu item into the specified buffer.
GetMessage	Retrieves a message from the calling thread's message queue. The function dispatches incoming sent messages until a posted message is available for retrieval.
GetMessageA	Retrieves a message from the calling thread's message queue. The function dispatches incoming sent messages until a posted message is available for retrieval.
GetMessageExtraInfo	Retrieves the extra message information for the current thread. Extra message information is an application- or driver-defined value associated with the current thread's message queue.
GetMessagePos	Retrieves the cursor position for the last message retrieved by the GetMessage function.

GetMessageTime	Retrieves the message time for the last message retrieved by the GetMessage function.
GetMessageW	Retrieves a message from the calling thread's message queue. The function dispatches incoming sent messages until a posted message is available for retrieval.
GetMonitorInfoA	The GetMonitorInfo function retrieves information about a display monitor.
GetMonitorInfoW	The GetMonitorInfo function retrieves information about a display monitor.
GetMouseMovePointsEx	Retrieves a history of up to 64 previous coordinates of the mouse or pen.
GetNextDlgGroupItem	Retrieves a handle to the first control in a group of controls that precedes (or follows) the specified control in a dialog box.
GetNextDlgTabItem	Retrieves a handle to the first control that has the WS_TABSTOP style that precedes (or follows) the specified control.
GetNextWindow	Retrieves a handle to the next or previous window in the Z-Order. The next window is below the specified window; the previous window is above.
GetOpenClipboardWindow	Retrieves the handle to the window that currently has the clipboard open.
GetParent	Retrieves a handle to the specified window's parent or owner.
GetPhysicalCursorPos	Retrieves the position of the cursor in physical coordinates.
GetPointerCursorId	Retrieves the cursor identifier associated with the specified pointer.
GetPointerDevice	Gets information about the pointer device.

GetPointerDeviceCursors Gets the cursor IDs that are mapped to the cursors associated with a pointer device.
GetPointerDeviceProperties Gets device properties that aren't included in the POINTER_DEVICE_INFO structure.
GetPointerDeviceRects Gets the x and y range for the pointer device (in himetric) and the x and y range (current resolution) for the display that the pointer device is mapped to.
GetPointerDevices Gets information about the pointer devices attached to the system.
GetPointerFrameInfo Gets the entire frame of information for the specified pointers associated with the current message.
GetPointerFrameInfoHistory Gets the entire frame of information (including coalesced input frames) for the specified pointers associated with the current message.
GetPointerFramePenInfo Gets the entire frame of pen-based information for the specified pointers (of type PT_PEN) associated with the current message.
GetPointerFramePenInfoHistory Gets the entire frame of pen-based information (including coalesced input frames) for the specified pointers (of type PT_PEN) associated with the current message.
GetPointerFrameTouchInfo Gets the entire frame of touch-based information for the specified pointers (of type PT_TOUCH) associated with the current message.
GetPointerFrameTouchInfoHistory Gets the entire frame of touch-based information (including coalesced input frames) for the specified pointers (of type PT_TOUCH) associated with the current message.
GetPointerInfo Gets the information for the specified pointer associated with the current message.
GetPointerInfoHistory Gets the information associated with the individual inputs, if any, that were coalesced into the current message for the specified pointer.

GetPointerInputTransform	Gets one or more transforms for the pointer information coordinates associated with the current message.
GetPointerPenInfo	Gets the pen-based information for the specified pointer (of type PT_PEN) associated with the current message.
GetPointerPenInfoHistory	Gets the pen-based information associated with the individual inputs, if any, that were coalesced into the current message for the specified pointer (of type PT_PEN).
GetPointerTouchInfo	Gets the touch-based information for the specified pointer (of type PT_TOUCH) associated with the current message.
GetPointerTouchInfoHistory	Gets the touch-based information associated with the individual inputs, if any, that were coalesced into the current message for the specified pointer (of type PT_TOUCH).
GetPointerType	Retrieves the pointer type for a specified pointer.
GetPriorityClipboardFormat	Retrieves the first available clipboard format in the specified list.
GetProcessDefaultLayout	Retrieves the default layout that is used when windows are created with no parent or owner.
GetProcessWindowStation	Retrieves a handle to the current window station for the calling process.
GetPropA	Retrieves a data handle from the property list of the specified window. The character string identifies the handle to be retrieved. The string and handle must have been added to the property list by a previous call to the SetProp function.
GetPropW	Retrieves a data handle from the property list of the specified window. The character string identifies the handle to be retrieved. The string and handle must have been added to the property list by a previous call to the SetProp function.
GetQueueStatus	Retrieves the type of messages found in the calling thread's message queue.
GetRawInputBuffer	Performs a buffered read of the raw input data.

GetRawInputData	Retrieves the raw input from the specified device.
GetRawInputDeviceInfoA	Retrieves information about the raw input device.
GetRawInputDeviceInfoW	Retrieves information about the raw input device.
GetRawInputDeviceList	Enumerates the raw input devices attached to the system.
GetRawPointerDeviceData	Gets the raw input data from the pointer device.
GetRegisteredRawInputDevices	Retrieves the information about the raw input devices for the current application.
GetScrollBarInfo	The GetScrollBarInfo function retrieves information about the specified scroll bar.
GetScrollInfo	The GetScrollInfo function retrieves the parameters of a scroll bar, including the minimum and maximum scrolling positions, the page size, and the position of the scroll box (thumb).
GetScrollPos	The GetScrollPos function retrieves the current position of the scroll box (thumb) in the specified scroll bar.
GetScrollRange	The GetScrollRange function retrieves the current minimum and maximum scroll box (thumb) positions for the specified scroll bar.
GetShellWindow	Retrieves a handle to the Shell's desktop window.
GetSubMenu	Retrieves a handle to the drop-down menu or submenu activated by the specified menu item.
GetSysColor	Retrieves the current color of the specified display element.

GetSysColorBrush	The GetSysColorBrush function retrieves a handle identifying a logical brush that corresponds to the specified color index.
GetSystemDpiForProcess	Retrieves the system DPI associated with a given process. This is useful for avoiding compatibility issues that arise from sharing DPI-sensitive information between multiple system-aware processes with different system DPI values.
GetSystemMenu	Enables the application to access the window menu (also known as the system menu or the control menu) for copying and modifying.
GetSystemMetrics	Retrieves the specified system metric or system configuration setting.
GetSystemMetricsForDpi	Retrieves the specified system metric or system configuration setting taking into account a provided DPI.
GetTabbedTextExtentA	The GetTabbedTextExtent function computes the width and height of a character string.
GetTabbedTextExtentW	The GetTabbedTextExtent function computes the width and height of a character string.
GetThreadDesktop	Retrieves a handle to the desktop assigned to the specified thread.
GetThreadDpiAwarenessContext	Gets the DPI_AWARENESS_CONTEXT for the current thread.
GetThreadDpiHostingBehavior	Retrieves the DPI_HOSTING_BEHAVIOR from the current thread.
GetTitleBarInfo	Retrieves information about the specified title bar.
GetTopWindow	Examines the Z order of the child windows associated with the specified parent window and retrieves a handle to the child window at the top of the Z order.
GetTouchInputInfo	Retrieves detailed information about touch inputs associated with a particular touch input handle.

GetUnpredictedMessagePos

Gets pointer data before it has gone through touch prediction processing.

GetUpdatedClipboardFormats

Retrieves the currently supported clipboard formats.

GetUpdateRect

The GetUpdateRect function retrieves the coordinates of the smallest rectangle that completely encloses the update region of the specified window.

GetUpdateRgn

The GetUpdateRgn function retrieves the update region of a window by copying it into the specified region. The coordinates of the update region are relative to the upper-left corner of the window (that is, they are client coordinates).

GetUserObjectInformationA

Retrieves information about the specified window station or desktop object.

GetUserObjectInformationW

Retrieves information about the specified window station or desktop object.

GetUserObjectSecurity

Retrieves security information for the specified user object.

GetWindow

Retrieves a handle to a window that has the specified relationship (Z-Order or owner) to the specified window.

GetWindowContextHelpId

Retrieves the Help context identifier, if any, associated with the specified window.

GetWindowDC

The GetWindowDC function retrieves the device context (DC) for the entire window, including title bar, menus, and scroll bars.

GetWindowDisplayAffinity

Retrieves the current display affinity setting, from any process, for a given window.

GetWindowDpiAwarenessContext

Returns the DPI_AWARENESS_CONTEXT associated with a window.

GetWindowDpiHostingBehavior

Returns the DPI_HOSTING_BEHAVIOR of the specified window.

GetWindowFeedbackSetting

Retrieves the feedback configuration for a window.

GetWindowInfo

Retrieves information about the specified window.

GetWindowLongA

Retrieves information about the specified window.

GetWindowLongPtrA

Retrieves information about the specified window. The function also retrieves the value at a specified offset into the extra window memory.

GetWindowLongPtrW

Retrieves information about the specified window. The function also retrieves the value at a specified offset into the extra window memory.

GetWindowLongW

Retrieves information about the specified window.

GetWindowModuleFileNameA

Retrieves the full path and file name of the module associated with the specified window handle.

GetWindowModuleFileNameW

Retrieves the full path and file name of the module associated with the specified window handle.

GetWindowPlacement

Retrieves the show state and the restored, minimized, and maximized positions of the specified window.

GetWindowRect

Retrieves the dimensions of the bounding rectangle of the specified window. The dimensions are given in screen coordinates that are relative to the upper-left corner of the screen.

GetWindowRgn

The GetWindowRgn function obtains a copy of the window region of a window.

GetWindowRgnBox

The GetWindowRgnBox function retrieves the dimensions of the tightest bounding rectangle for the window region of a window.

GetWindowTextA

Copies the text of the specified window's title bar (if it has one) into a buffer. If the specified window is a control, the text of the control is copied. However, GetWindowText cannot retrieve the text of a control in another application.

GetWindowTextLengthA

Retrieves the length, in characters, of the specified window's title bar text (if the window has a title bar).

GetWindowTextLengthW

Retrieves the length, in characters, of the specified window's title bar text (if the window has a title bar).

GetWindowTextW

Copies the text of the specified window's title bar (if it has one) into a buffer. If the specified window is a control, the text of the control is copied. However, GetWindowText cannot retrieve the text of a control in another application.

GetWindowThreadProcessId

Retrieves the identifier of the thread that created the specified window and, optionally, the identifier of the process that created the window.

GID_ROTATE_ANGLE_FROM_ARGUMENT

The GID_ROTATE_ANGLE_FROM_ARGUMENT macro is used to interpret the GID_ROTATE ullArgument value when receiving the value in the WM_GESTURE structure.

GID_ROTATE_ANGLE_TO_ARGUMENT

Converts a radian value to an argument for rotation gesture messages.

GrayStringA

The GrayString function draws gray text at the specified location.

GrayStringW

The GrayString function draws gray text at the specified location.

HAS_POINTER_CONFIDENCE_WPARAM

Checks whether the specified pointer message is considered intentional rather than accidental.

HideCaret

Removes the caret from the screen. Hiding a caret does not destroy its current shape or invalidate the insertion point.

HiliteMenuItem

Adds or removes highlighting from an item in a menu bar.

InflateRect

The InflateRect function increases or decreases the width and height of the specified rectangle.

InitializeTouchInjection

Configures the touch injection context for the calling application and initializes the maximum number of simultaneous contacts that the app can inject.

InjectSyntheticPointerInput	Simulates pointer input (pen or touch).
InjectTouchInput	Simulates touch input.
InSendMessage	Determines whether the current window procedure is processing a message that was sent from another thread (in the same process or a different process) by a call to the SendMessage function.
InSendMessageEx	Determines whether the current window procedure is processing a message that was sent from another thread (in the same process or a different process).
InsertMenuA	Inserts a new menu item into a menu, moving other items down the menu.
InsertMenuItemA	Inserts a new menu item at the specified position in a menu.
InsertMenuItemW	Inserts a new menu item at the specified position in a menu.
InsertMenuW	Inserts a new menu item into a menu, moving other items down the menu.
InternalGetWindowText	Copies the text of the specified window's title bar (if it has one) into a buffer.
IntersectRect	The IntersectRect function calculates the intersection of two source rectangles and places the coordinates of the intersection rectangle into the destination rectangle.
InvalidateRect	The InvalidateRect function adds a rectangle to the specified window's update region. The update region represents the portion of the window's client area that must be redrawn.
InvalidateRgn	The InvalidateRgn function invalidates the client area within the specified region by adding it to the current update region of a window.

InvertRect	The InvertRect function inverts a rectangle in a window by performing a logical NOT operation on the color values for each pixel in the rectangle's interior.
IS_INTRESOURCE	Determines whether a value is an integer identifier for a resource.
IS_POINTER_CANCELED_WPARAM	Checks whether the specified pointer input ended abruptly, or was invalid, indicating the interaction was not completed.
IS_POINTER_FIFTHBUTTON_WPARAM	Checks whether the specified pointer took fifth action.
IS_POINTER_FIRSTBUTTON_WPARAM	Checks whether the specified pointer took first action.
IS_POINTER_FLAG_SET_WPARAM	Checks whether a pointer macro sets the specified flag.
IS_POINTER_FOURTHBUTTON_WPARAM	Checks whether the specified pointer took fourth action.
IS_POINTER_INCONTACT_WPARAM	Checks whether the specified pointer is in contact.
IS_POINTER_INRANGE_WPARAM	Checks whether the specified pointer is in range.
IS_POINTER_NEW_WPARAM	Checks whether the specified pointer is a new pointer.
IS_POINTER_SECONDBUTTON_WPARAM	Checks whether the specified pointer took second action.
IS_POINTER_THIRDBUTTON_WPARAM	Checks whether the specified pointer took third action.
IsCharAlphaA	Determines whether a character is an alphabetical character. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

IsCharAlphaNumericA

Determines whether a character is either an alphabetical or a numeric character. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

IsCharAlphaNumericW

Determines whether a character is either an alphabetical or a numeric character. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

IsCharAlphaW

Determines whether a character is an alphabetical character. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

IsCharLowerA

Determines whether a character is lowercase. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

IsCharLowerW

IsCharUpperA

Determines whether a character is uppercase. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

IsCharUpperW

Determines whether a character is uppercase. This determination is based on the semantics of the language selected by the user during setup or through Control Panel.

IsChild

Determines whether a window is a child window or descendant window of a specified parent window.

IsClipboardFormatAvailable

Determines whether the clipboard contains data in the specified format.

IsDialogMessageA

Determines whether a message is intended for the specified dialog box and, if it is, processes the message.

IsDialogMessageW

Determines whether a message is intended for the specified dialog box and, if it is, processes the message.

IsDlgButtonChecked

The IsDlgButtonChecked function determines whether a button control is checked or whether a three-state button control is checked, unchecked, or indeterminate.

IsGUIThread

Determines whether the calling thread is already a GUI thread. It can also optionally convert the thread to a GUI thread.

IsHungAppWindow

Determines whether the system considers that a specified application is not responding.

IsIconic

Determines whether the specified window is minimized (iconic).

IsImmersiveProcess

Determines whether the process belongs to a Windows Store app.

IsMenu

Determines whether a handle is a menu handle.

IsMouseInPointerEnabled

Indicates whether EnableMouseInPointer is set for the mouse to act as a pointer input device and send WM_POINTER messages.

IsProcessDPIAware

IsProcessDPIAware may be altered or unavailable. Instead, use GetProcessDPIAwareness.

IsRectEmpty

The IsRectEmpty function determines whether the specified rectangle is empty.

IsTouchWindow

Checks whether a specified window is touch-capable and, optionally, retrieves the modifier flags set for the window's touch capability.

IsValidDpiAwarenessContext

Determines if a specified DPI_AWARENESS_CONTEXT is valid and supported by the current system.

IsWindow

Determines whether the specified window handle identifies an existing window.

IsWindowEnabled

Determines whether the specified window is enabled for mouse and keyboard input.

IsWindowUnicode

Determines whether the specified window is a native Unicode window.

IsWindowVisible	Determines the visibility state of the specified window.
IsWinEventHookInstalled	Determines whether there is an installed WinEvent hook that might be notified of a specified event.
IsWow64Message	Determines whether the last message read from the current thread's queue originated from a WOW64 process.
IsZoomed	Determines whether a window is maximized.
keybd_event	Synthesizes a keystroke.
KillTimer	Destroys the specified timer.
LoadAcceleratorsA	Loads the specified accelerator table.
LoadAcceleratorsW	Loads the specified accelerator table.
LoadBitmapA	The LoadBitmap function loads the specified bitmap resource from a module's executable file.
LoadBitmapW	The LoadBitmap function loads the specified bitmap resource from a module's executable file.
LoadCursorA	Loads the specified cursor resource from the executable (.EXE) file associated with an application instance.
LoadCursorFromFileA	Creates a cursor based on data contained in a file.
LoadCursorFromFileW	Creates a cursor based on data contained in a file.
LoadCursorW	Loads the specified cursor resource from the executable (.EXE) file associated with an application instance.

LoadIconA	Loads the specified icon resource from the executable (.exe) file associated with an application instance.
LoadIconW	Loads the specified icon resource from the executable (.exe) file associated with an application instance.
LoadImageA	Loads an icon, cursor, animated cursor, or bitmap.
LoadImageW	Loads an icon, cursor, animated cursor, or bitmap.
LoadKeyboardLayoutA	Loads a new input locale identifier (formerly called the keyboard layout) into the system.
LoadKeyboardLayoutW	Loads a new input locale identifier (formerly called the keyboard layout) into the system.
LoadMenuA	Loads the specified menu resource from the executable (.exe) file associated with an application instance.
LoadMenuIndirectA	Loads the specified menu template in memory.
LoadMenuIndirectW	Loads the specified menu template in memory.
LoadMenuW	Loads the specified menu resource from the executable (.exe) file associated with an application instance.
LoadStringA	Loads a string resource from the executable file associated with a specified module, copies the string into a buffer, and appends a terminating null character.
LoadStringW	Loads a string resource from the executable file associated with a specified module, copies the string into a buffer, and appends a terminating null character.
LockSetForegroundWindow	The foreground process can call the LockSetForegroundWindow function to disable calls to the SetForegroundWindow function.

LockWindowUpdate	The LockWindowUpdate function disables or enables drawing in the specified window. Only one window can be locked at a time.
LockWorkStation	Locks the workstation's display.
LogicalToPhysicalPoint	Converts the logical coordinates of a point in a window to physical coordinates.
LogicalToPhysicalPointForPerMonitorDPI	Converts a point in a window from logical coordinates into physical coordinates, regardless of the dots per inch (dpi) awareness of the caller.
LookupIconIdFromDirectory	Searches through icon or cursor data for the icon or cursor that best fits the current display device.
LookupIconIdFromDirectoryEx	Searches through icon or cursor data for the icon or cursor that best fits the current display device.
MAKEINTRESOURCEA	Converts an integer value to a resource type compatible with the resource-management functions. This macro is used in place of a string containing the name of the resource.
MAKEINTRESOURCEW	Converts an integer value to a resource type compatible with the resource-management functions. This macro is used in place of a string containing the name of the resource.
MAKELPARAM	Creates a value for use as an lParam parameter in a message. The macro concatenates the specified values.
MAKELRESULT	Creates a value for use as a return value from a window procedure. The macro concatenates the specified values.
MAKEWPARAM	Creates a value for use as a wParam parameter in a message. The macro concatenates the specified values.
MapDialogRect	Converts the specified dialog box units to screen units (pixels).
MapVirtualKeyA	Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code.

MapVirtualKeyExA	Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code. The function translates the codes using the input language and an input locale identifier.
MapVirtualKeyExW	Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code. The function translates the codes using the input language and an input locale identifier.
MapVirtualKeyW	Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code.
MapWindowPoints	The MapWindowPoints function converts (maps) a set of points from a coordinate space relative to one window to a coordinate space relative to another window.
MenuItemFromPoint	Determines which menu item, if any, is at the specified location.
MessageBeep	Plays a waveform sound. The waveform sound for each sound type is identified by an entry in the registry.
MessageBox	Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.
MessageBoxA	Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.
MessageBoxExA	Creates, displays, and operates a message box.
MessageBoxExW	Creates, displays, and operates a message box.
MessageBoxIndirectA	Creates, displays, and operates a message box. The message box contains application-defined message text and title, any icon, and any combination of predefined push buttons.
MessageBoxIndirectW	Creates, displays, and operates a message box. The message box contains application-defined message text and title, any icon, and any combination of predefined push buttons.

MessageBoxW	Displays a modal dialog box that contains a system icon, a set of buttons, and a brief application-specific message, such as status or error information. The message box returns an integer value that indicates which button the user clicked.
ModifyMenuA	Changes an existing menu item.
ModifyMenuW	Changes an existing menu item.
MonitorFromPoint	The MonitorFromPoint function retrieves a handle to the display monitor that contains a specified point.
MonitorFromRect	The MonitorFromRect function retrieves a handle to the display monitor that has the largest area of intersection with a specified rectangle.
MonitorFromWindow	The MonitorFromWindow function retrieves a handle to the display monitor that has the largest area of intersection with the bounding rectangle of a specified window.
mouse_event	The mouse_event function synthesizes mouse motion and button clicks.
MoveWindow	Changes the position and dimensions of the specified window.
MsgWaitForMultipleObjects	Waits until one or all of the specified objects are in the signaled state or the time-out interval elapses. The objects can include input event objects.
MsgWaitForMultipleObjectsEx	Waits until one or all of the specified objects are in the signaled state, an I/O completion routine or asynchronous procedure call (APC) is queued to the thread, or the time-out interval elapses. The array of objects can include input event objects.
NEXTRAWINPUTBLOCK	Retrieves the location of the next structure in an array of RAWINPUT structures.
NotifyWinEvent	Signals the system that a predefined event occurred. If any client applications have registered a hook function for the event, the system calls the client's hook function.

OemKeyScan	Maps OEMASCII codes 0 through 0x0FF into the OEM scan codes and shift states. The function provides information that allows a program to send OEM text to another program by simulating keyboard input.
OemToCharA	Translates a string from the OEM-defined character set into either an ANSI or a wide-character string.Warning Do not use.
OemToCharBuffA	Translates a specified number of characters in a string from the OEM-defined character set into either an ANSI or a wide-character string.
OemToCharBuffW	Translates a specified number of characters in a string from the OEM-defined character set into either an ANSI or a wide-character string.
OemToCharW	Translates a string from the OEM-defined character set into either an ANSI or a wide-character string.Warning Do not use.
OffsetRect	The OffsetRect function moves the specified rectangle by the specified offsets.
OpenClipboard	Opens the clipboard for examination and prevents other applications from modifying the clipboard content.
OpenDesktopA	Opens the specified desktop object.
OpenDesktopW	Opens the specified desktop object.
OpenIcon	Restores a minimized (iconic) window to its previous size and position; it then activates the window.
OpenInputDesktop	Opens the desktop that receives user input.
OpenWindowStationA	Opens the specified window station.
OpenWindowStationW	Opens the specified window station.

PackTouchHitTestingProximityEvaluation	Returns the proximity evaluation score and the adjusted touch-point coordinates as a packed value for the WM_TOUCHHITTESTING callback.
PaintDesktop	The PaintDesktop function fills the clipping region in the specified device context with the desktop pattern or wallpaper. The function is provided primarily for shell desktops.
PeekMessageA	Dispatches incoming nonqueued messages, checks the thread message queue for a posted message, and retrieves the message (if any exist).
PeekMessageW	Dispatches incoming nonqueued messages, checks the thread message queue for a posted message, and retrieves the message (if any exist).
PhysicalToLogicalPoint	Converts the physical coordinates of a point in a window to logical coordinates.
PhysicalToLogicalPointForPerMonitorDPI	Converts a point in a window from physical coordinates into logical coordinates, regardless of the dots per inch (dpi) awareness of the caller.
POINTSTOPOINT	The POINTSTOPOINT macro copies the contents of a POINTS structure into a POINT structure.
POINTTOPOINTS	The POINTTOPOINTS macro converts a POINT structure to a POINTS structure.
PostMessageA	Places (posts) a message in the message queue associated with the thread that created the specified window and returns without waiting for the thread to process the message.
PostMessageW	Places (posts) a message in the message queue associated with the thread that created the specified window and returns without waiting for the thread to process the message.
PostQuitMessage	Indicates to the system that a thread has made a request to terminate (quit). It is typically used in response to a WM_DESTROY message.
PostThreadMessageA	Posts a message to the message queue of the specified thread. It returns without waiting for the thread to process the message.

PostThreadMessageW	Posts a message to the message queue of the specified thread. It returns without waiting for the thread to process the message.
PrintWindow	The PrintWindow function copies a visual window into the specified device context (DC), typically a printer DC.
PrivateExtractIconsA	Creates an array of handles to icons that are extracted from a specified file.
PrivateExtractIconsW	Creates an array of handles to icons that are extracted from a specified file.
PtInRect	The PtInRect function determines whether the specified point lies within the specified rectangle.
QueryDisplayConfig	The QueryDisplayConfig function retrieves information about all possible display paths for all display devices, or views, in the current setting.
RealChildWindowFromPoint	Retrieves a handle to the child window at the specified point. The search is restricted to immediate child windows; grandchildren and deeper descendant windows are not searched.
RealGetWindowClassA	Retrieves a string that specifies the window type.
RealGetWindowClassW	Retrieves a string that specifies the window type.
RedrawWindow	The RedrawWindow function updates the specified rectangle or region in a window's client area.
RegisterClassA	Registers a window class for subsequent use in calls to the CreateWindow or CreateWindowEx function.
RegisterClassExA	Registers a window class for subsequent use in calls to the CreateWindow or CreateWindowEx function.
RegisterClassExW	Registers a window class for subsequent use in calls to the CreateWindow or CreateWindowEx function.

RegisterClassW	Registers a window class for subsequent use in calls to the CreateWindow or CreateWindowEx function.
RegisterClipboardFormatA	Registers a new clipboard format. This format can then be used as a valid clipboard format.
RegisterClipboardFormatW	Registers a new clipboard format. This format can then be used as a valid clipboard format.
RegisterDeviceNotificationA	Registers the device or type of device for which a window will receive notifications.
RegisterDeviceNotificationW	Registers the device or type of device for which a window will receive notifications.
RegisterForTooltipDismissNotification	Lets apps or UI frameworks register and unregister windows to receive notification to dismiss their tooltip windows.
RegisterHotKey	Defines a system-wide hot key.
RegisterPointerDeviceNotifications	Registers a window to process the WM_POINTERDEVICECHANGE, WM_POINTERDEVICEINRANGE, and WM_POINTERDEVICEOUTOFRANGE pointer device notifications.
RegisterPointerInputTarget	Allows the caller to register a target window to which all pointer input of the specified type is redirected.
RegisterPointerInputTargetEx	RegisterPointerInputTargetEx may be altered or unavailable. Instead, use RegisterPointerInputTarget.
RegisterPowerSettingNotification	Registers the application to receive power setting notifications for the specific power setting event.
RegisterRawInputDevices	Registers the devices that supply the raw input data.
RegisterShellHookWindow	Registers a specified Shell window to receive certain messages for events or notifications that are useful to Shell applications.

RegisterSuspendResumeNotification	Registers to receive notification when the system is suspended or resumed. Similar to PowerRegisterSuspendResumeNotification, but operates in user mode and can take a window handle.
RegisterTouchHitTestingWindow	Registers a window to process the WM_TOUCHHITTESTING notification.
RegisterTouchWindow	Registers a window as being touch-capable.
RegisterWindowMessageA	Defines a new window message that is guaranteed to be unique throughout the system. The message value can be used when sending or posting messages.
RegisterWindowMessageW	Defines a new window message that is guaranteed to be unique throughout the system. The message value can be used when sending or posting messages.
ReleaseCapture	Releases the mouse capture from a window in the current thread and restores normal mouse input processing.
ReleaseDC	The ReleaseDC function releases a device context (DC), freeing it for use by other applications. The effect of the ReleaseDC function depends on the type of DC. It frees only common and window DCs. It has no effect on class or private DCs.
RemoveClipboardFormatListener	Removes the given window from the system-maintained clipboard format listener list.
RemoveMenu	Deletes a menu item or detaches a submenu from the specified menu.
RemovePropA	Removes an entry from the property list of the specified window. The specified character string identifies the entry to be removed.
RemovePropW	Removes an entry from the property list of the specified window. The specified character string identifies the entry to be removed.
ReplyMessage	Replies to a message sent from another thread by the SendMessage function.

ScreenToClient	The ScreenToClient function converts the screen coordinates of a specified point on the screen to client-area coordinates.
ScrollDC	The ScrollDC function scrolls a rectangle of bits horizontally and vertically.
ScrollWindow	The ScrollWindow function scrolls the contents of the specified window's client area.
ScrollWindowEx	The ScrollWindowEx function scrolls the contents of the specified window's client area.
SendDlgItemMessageA	Sends a message to the specified control in a dialog box.
SendDlgItemMessageW	Sends a message to the specified control in a dialog box.
SendInput	Synthesizes keystrokes, mouse motions, and button clicks.
SendMessage	Sends the specified message to a window or windows. The SendMessage function calls the window procedure for the specified window and does not return until the window procedure has processed the message.
SendMessageA	Sends the specified message to a window or windows. The SendMessage function calls the window procedure for the specified window and does not return until the window procedure has processed the message.
SendMessageCallbackA	Sends the specified message to a window or windows.
SendMessageCallbackW	Sends the specified message to a window or windows.
SendMessageTimeoutA	Sends the specified message to one or more windows.
SendMessageTimeoutW	Sends the specified message to one or more windows.

SendMessageW

Sends the specified message to a window or windows. The SendMessage function calls the window procedure for the specified window and does not return until the window procedure has processed the message.

SendNotifyMessageA

Sends the specified message to a window or windows.

SendNotifyMessageW

Sends the specified message to a window or windows.

SetActiveWindow

Activates a window. The window must be attached to the calling thread's message queue.

SetAdditionalForegroundBoostProcesses

SetAdditionalForegroundBoostProcesses is a performance assist API to help applications with a multi-process application model where multiple processes contribute to a foreground experience, either as data or rendering.

SetCapture

Sets the mouse capture to the specified window belonging to the current thread.

SetCaretBlinkTime

Sets the caret blink time to the specified number of milliseconds. The blink time is the elapsed time, in milliseconds, required to invert the caret's pixels.

SetCaretPos

Moves the caret to the specified coordinates. If the window that owns the caret was created with the CS_OWNDC class style, then the specified coordinates are subject to the mapping mode of the device context associated with that window.

SetClassLongA

Replaces the specified 32-bit (long) value at the specified offset into the extra class memory or the WNDCLASSEX structure for the class to which the specified window belongs.

SetClassLongPtrA

Replaces the specified value at the specified offset in the extra class memory or the WNDCLASSEX structure for the class to which the specified window belongs.

SetClassLongPtrW

Replaces the specified value at the specified offset in the extra class memory or the WNDCLASSEX structure for the class to which the specified window belongs.

SetClassLongW

Replaces the specified 32-bit (long) value at the specified offset into the extra class memory or the WNDCLASSEX structure for the class to which the specified window belongs.

SetClassWord	Replaces the 16-bit (WORD) value at the specified offset into the extra class memory for the window class to which the specified window belongs.
SetClipboardData	Places data on the clipboard in a specified clipboard format.
SetClipboardViewer	Adds the specified window to the chain of clipboard viewers. Clipboard viewer windows receive a WM_DRAWCLIPBOARD message whenever the content of the clipboard changes. This function is used for backward compatibility with earlier versions of Windows.
SetCoalescableTimer	Creates a timer with the specified time-out value and coalescing tolerance delay.
SetCursor	Sets the cursor shape.
SetCursorPos	Moves the cursor to the specified screen coordinates.
SetDialogControlDpiChangeBehavior	Overrides the default per-monitor DPI scaling behavior of a child window in a dialog.
SetDialogDpiChangeBehavior	Dialogs in Per-Monitor v2 contexts are automatically DPI scaled. This method lets you customize their DPI change behavior.
SetDisplayAutoRotationPreferences	Sets the screen auto-rotation preferences for the current process.
SetDisplayConfig	The SetDisplayConfig function modifies the display topology, source, and target modes by exclusively enabling the specified paths in the current session.
SetDlgItemInt	Sets the text of a control in a dialog box to the string representation of a specified integer value.
SetDlgItemTextA	Sets the title or text of a control in a dialog box.
SetDlgItemTextW	Sets the title or text of a control in a dialog box.

SetDoubleClickTime	Sets the double-click time for the mouse.
SetFocus	Sets the keyboard focus to the specified window. The window must be attached to the calling thread's message queue.
SetForegroundWindow	Brings the thread that created the specified window into the foreground and activates the window.
SetGestureConfig	Configures the messages that are sent from a window for Windows Touch gestures.
SetKeyboardState	Copies an array of keyboard key states into the calling thread's keyboard input-state table. This is the same table accessed by the GetKeyboardState and GetKeyState functions. Changes made to this table do not affect keyboard input to any other thread.
SetLastErrorEx	Sets the last-error code.
SetLayeredWindowAttributes	Sets the opacity and transparency color key of a layered window.
SetMenu	Assigns a new menu to the specified window.
SetMenuContextHelpId	Associates a Help context identifier with a menu.
SetMenuDefaultItem	Sets the default menu item for the specified menu.
SetMenuInfo	Sets information for a specified menu.
SetMenuItemBitmaps	Associates the specified bitmap with a menu item. Whether the menu item is selected or clear, the system displays the appropriate bitmap next to the menu item.
SetMenuItemInfoA	Changes information about a menu item.

SetMenuItemInfoW	Changes information about a menu item.
SetMessageExtraInfo	Sets the extra message information for the current thread.
SetParent	Changes the parent window of the specified child window.
SetPhysicalCursorPos	Sets the position of the cursor in physical coordinates.
SetProcessDefaultLayout	Changes the default layout when windows are created with no parent or owner only for the currently running process.
SetProcessDPIAware	SetProcessDPIAware may be altered or unavailable. Instead, use SetProcessDPIAwareness.
SetProcessDpiAwarenessContext	Sets the current process to a specified dots per inch (dpi) awareness context. The DPI awareness contexts are from the DPI_AWARENESS_CONTEXT value.
SetProcessRestrictionExemption	Exempts the calling process from restrictions preventing desktop processes from interacting with the Windows Store app environment. This function is used by development and debugging tools.
SetProcessWindowStation	Assigns the specified window station to the calling process.
SetPropA	Adds a new entry or changes an existing entry in the property list of the specified window.
SetPropW	Adds a new entry or changes an existing entry in the property list of the specified window.
SetRect	The SetRect function sets the coordinates of the specified rectangle. This is equivalent to assigning the left, top, right, and bottom arguments to the appropriate members of the RECT structure.
SetRectEmpty	The SetRectEmpty function creates an empty rectangle in which all coordinates are set to zero.

SetScrollInfo

The SetScrollInfo function sets the parameters of a scroll bar, including the minimum and maximum scrolling positions, the page size, and the position of the scroll box (thumb). The function also redraws the scroll bar, if requested.

SetScrollPos

The SetScrollPos function sets the position of the scroll box (thumb) in the specified scroll bar and, if requested, redraws the scroll bar to reflect the new position of the scroll box.

SetScrollRange

The SetScrollRange function sets the minimum and maximum scroll box positions for the specified scroll bar.

SetSysColors

Sets the colors for the specified display elements.

SetSystemCursor

Enables an application to customize the system cursors. It replaces the contents of the system cursor specified by the id parameter with the contents of the cursor specified by the hcur parameter and then destroys hcur.

SetThreadCursorCreationScaling

Sets the DPI scale for which the cursors being created on this thread are intended. This value is taken into account when scaling the cursor for the specific monitor on which it is being shown.

SetThreadDesktop

Assigns the specified desktop to the calling thread. All subsequent operations on the desktop use the access rights granted to the desktop.

SetThreadDpiAwarenessContext

Set the DPI awareness for the current thread to the provided value.

SetThreadDpiHostingBehavior

Sets the thread's DPI_HOSTING_BEHAVIOR. This behavior allows windows created in the thread to host child windows with a different DPI_AWARENESS_CONTEXT.

SetTimer

Creates a timer with the specified time-out value.

SetUserObjectInformationA

Sets information about the specified window station or desktop object.

SetUserObjectInformationW

Sets information about the specified window station or desktop object.

SetUserObjectSecurity	Sets the security of a user object. This can be, for example, a window or a DDE conversation.
SetWindowContextHelpId	Associates a Help context identifier with the specified window.
SetWindowDisplayAffinity	Stores the display affinity setting in kernel mode on the hWnd associated with the window.
SetWindowFeedbackSetting	Sets the feedback configuration for a window.
SetWindowLongA	Changes an attribute of the specified window. The function also sets the 32-bit (long) value at the specified offset into the extra window memory.
SetWindowLongPtrA	Changes an attribute of the specified window.
SetWindowLongPtrW	Changes an attribute of the specified window.
SetWindowLongW	Changes an attribute of the specified window. The function also sets the 32-bit (long) value at the specified offset into the extra window memory.
SetWindowPlacement	Sets the show state and the restored, minimized, and maximized positions of the specified window.
SetWindowPos	Changes the size, position, and Z order of a child, pop-up, or top-level window. These windows are ordered according to their appearance on the screen. The topmost window receives the highest rank and is the first window in the Z order.
SetWindowRgn	The SetWindowRgn function sets the window region of a window.
SetWindowsHookExA	Installs an application-defined hook procedure into a hook chain.
SetWindowsHookExW	Installs an application-defined hook procedure into a hook chain.

SetWindowTextA Changes the text of the specified window's title bar (if it has one). If the specified window is a control, the text of the control is changed. However, SetWindowText cannot change the text of a control in another application.
SetWindowTextW Changes the text of the specified window's title bar (if it has one). If the specified window is a control, the text of the control is changed. However, SetWindowText cannot change the text of a control in another application.
SetWinEventHook Sets an event hook function for a range of events.
ShowCaret Makes the caret visible on the screen at the caret's current position. When the caret becomes visible, it begins flashing automatically.
ShowCursor Displays or hides the cursor.
ShowOwnedPopups Shows or hides all pop-up windows owned by the specified window.
ShowScrollBar The ShowScrollBar function shows or hides the specified scroll bar.
ShowWindow Sets the specified window's show state.
ShowWindowAsync Sets the show state of a window without waiting for the operation to complete.
ShutdownBlockReasonCreate Indicates that the system cannot be shut down and sets a reason string to be displayed to the user if system shutdown is initiated.
ShutdownBlockReasonDestroy Indicates that the system can be shut down and frees the reason string.
ShutdownBlockReasonQuery Retrieves the reason string set by the ShutdownBlockReasonCreate function.
SkipPointerFrameMessages Determines which pointer input frame generated the most recently retrieved message for the specified pointer and discards any queued (unretrieved) pointer input messages generated from the same pointer input frame.

SoundSentry

Triggers a visual signal to indicate that a sound is playing.

SubtractRect

The SubtractRect function determines the coordinates of a rectangle formed by subtracting one rectangle from another.

SwapMouseButton

Reverses or restores the meaning of the left and right mouse buttons.

SwitchDesktop

Makes the specified desktop visible and activates it. This enables the desktop to receive input from the user.

SwitchToThisWindow

Switches focus to the specified window and brings it to the foreground.

SystemParametersInfoA

Retrieves or sets the value of one of the system-wide parameters.

SystemParametersInfoForDpi

Retrieves the value of one of the system-wide parameters, taking into account the provided DPI value.

SystemParametersInfoW

Retrieves or sets the value of one of the system-wide parameters.

TabbedTextOutA

The TabbedTextOut function writes a character string at a specified location, expanding tabs to the values specified in an array of tab-stop positions. Text is written in the currently selected font, background color, and text color.

TabbedTextOutW

The TabbedTextOut function writes a character string at a specified location, expanding tabs to the values specified in an array of tab-stop positions. Text is written in the currently selected font, background color, and text color.

TileWindows

Tiles the specified child windows of the specified parent window.

ToAscii

Translates the specified virtual-key code and keyboard state to the corresponding character or characters.

ToAsciiEx

Translates the specified virtual-key code and keyboard state to the corresponding character or characters. The function translates the code using the input language and physical keyboard layout identified by the input locale identifier.

TOUCH_COORD_TO_PIXEL	Converts touch coordinates to pixels.
ToUnicode	Translates the specified virtual-key code and keyboard state to the corresponding Unicode character or characters.
ToUnicodeEx	Translates the specified virtual-key code and keyboard state to the corresponding Unicode character or characters.
TrackMouseEvent	Posts messages when the mouse pointer leaves a window or hovers over a window for a specified amount of time.
TrackPopupMenu	Displays a shortcut menu at the specified location and tracks the selection of items on the menu. The shortcut menu can appear anywhere on the screen.
TrackPopupMenuEx	Displays a shortcut menu at the specified location and tracks the selection of items on the shortcut menu. The shortcut menu can appear anywhere on the screen.
TranslateAcceleratorA	Processes accelerator keys for menu commands.
TranslateAcceleratorW	Processes accelerator keys for menu commands.
TranslateMDISysAccel	Processes accelerator keystrokes for window menu commands of the multiple-document interface (MDI) child windows associated with the specified MDI client window.
TranslateMessage	Translates virtual-key messages into character messages. The character messages are posted to the calling thread's message queue, to be read the next time the thread calls the GetMessage or PeekMessage function.
UnhookWindowsHookEx	Removes a hook procedure installed in a hook chain by the SetWindowsHookEx function.
UnhookWinEvent	Removes an event hook function created by a previous call to SetWinEventHook.
UnionRect	The UnionRect function creates the union of two rectangles. The union is the smallest rectangle that contains both source rectangles.

UnloadKeyboardLayout	Unloads an input locale identifier (formerly called a keyboard layout).
UnregisterClassA	Unregisters a window class, freeing the memory required for the class.
UnregisterClassW	Unregisters a window class, freeing the memory required for the class.
UnregisterDeviceNotification	Closes the specified device notification handle.
UnregisterHotKey	Frees a hot key previously registered by the calling thread.
UnregisterPointerInputTarget	Allows the caller to unregister a target window to which all pointer input of the specified type is redirected.
UnregisterPointerInputTargetEx	UnregisterPointerInputTargetEx may be altered or unavailable. Instead, use UnregisterPointerInputTarget.
UnregisterPowerSettingNotification	Unregisters the power setting notification.
UnregisterSuspendResumeNotification	Cancels a registration to receive notification when the system is suspended or resumed. Similar to PowerUnregisterSuspendResumeNotification but operates in user mode.
UnregisterTouchWindow	Registers a window as no longer being touch-capable.
UpdateLayeredWindow	Updates the position, size, shape, content, and translucency of a layered window.
UpdateWindow	The UpdateWindow function updates the client area of the specified window by sending a WM_PAINT message to the window if the window's update region is not empty.
UserHandleGrantAccess	Grants or denies access to a handle to a User object to a job that has a user-interface restriction.

ValidateRect	The ValidateRect function validates the client area within a rectangle by removing the rectangle from the update region of the specified window.
ValidateRgn	The ValidateRgn function validates the client area within a region by removing the region from the current update region of the specified window.
VkKeyScanA	Translates a character to the corresponding virtual-key code and shift state for the current keyboard.
VkKeyScanExA	Translates a character to the corresponding virtual-key code and shift state. The function translates the character using the input language and physical keyboard layout identified by the input locale identifier.
VkKeyScanExW	Translates a character to the corresponding virtual-key code and shift state. The function translates the character using the input language and physical keyboard layout identified by the input locale identifier.
VkKeyScanW	Translates a character to the corresponding virtual-key code and shift state for the current keyboard.
WaitForInputIdle	Waits until the specified process has finished processing its initial input and is waiting for user input with no input pending, or until the time-out interval has elapsed.
WaitMessage	Yields control to other threads when a thread has no other messages in its message queue. The WaitMessage function suspends the thread and does not return until a new message is placed in the thread's message queue.
WindowFromDC	The WindowFromDC function returns a handle to the window associated with the specified display device context (DC). Output functions that use the specified device context draw into this window.
WindowFromPhysicalPoint	Retrieves a handle to the window that contains the specified physical point.
WindowFromPoint	Retrieves a handle to the window that contains the specified point.
WinHelpA	Launches Windows Help (Winhelp.exe) and passes additional data that indicates the nature of the help requested by the application.

WinHelpW	Launches Windows Help (Winhelp.exe) and passes additional data that indicates the nature of the help requested by the application.
wsprintfA	Writes formatted data to the specified buffer.
wsprintfW	Writes formatted data to the specified buffer.
wvsprintfA	Writes formatted data to the specified buffer using a pointer to a list of arguments.
wvsprintfW	Writes formatted data to the specified buffer using a pointer to a list of arguments.

Callback functions

DLGPROC	Application-defined callback function used with the CreateDialog and DialogBox families of functions.
DRAWSTATEPROC	The DrawStateProc function is an application-defined callback function that renders a complex image for the DrawState function.
EDITWORDBREAKPROCA	An application-defined callback function used with the EM_SETWORDBREAKPROC message.
EDITWORDBREAKPROCW	An application-defined callback function used with the EM_SETWORDBREAKPROC message.
GRAYSTRINGPROC	The OutputProc function is an application-defined callback function used with the GrayString function.
HOOKPROC	An application-defined or library-defined callback function used with the SetWindowsHookEx function. The system calls this function after the SendMessage function is called. The hook procedure can examine the message; it cannot modify it.
MONITORENUMPROC	A MonitorEnumProc function is an application-defined callback function that is called by the EnumDisplayMonitors function.

MSGBOXCALLBACK

A callback function, which you define in your application, that processes help events for the message box.

PROPENUMPROCA

An application-defined callback function used with the EnumProps function.

PROPENUMPROCEXA

Application-defined callback function used with the EnumPropsEx function.

PROPENUMPROCEXW

Application-defined callback function used with the EnumPropsEx function.

PROPENUMPROCW

An application-defined callback function used with the EnumProps function.

SENDASYNCPROC

An application-defined callback function used with the SendMessageCallback function.

TIMERPROC

An application-defined callback function that processes WM_TIMER messages. The TIMERPROC type defines a pointer to this callback function. TimerProc is a placeholder for the application-defined function name.

WINEVENTPROC

An application-defined callback (or hook) function that the system calls in response to events generated by an accessible object.

WNDPROC

A callback function, which you define in your application, that processes messages sent to a window.

Structures

ACCEL

Defines an accelerator key used in an accelerator table.

ACCESSTIMEOUT

Contains information about the time-out period associated with the accessibility features.

ALTTABINFO

Contains status information for the application-switching (ALT+TAB) window.

ANIMATIONINFO	Describes the animation effects associated with user actions.
AUDIODESCRIPTION	Contains information associated with audio descriptions. This structure is used with the SystemParametersInfo function when the SPI_GETAUDIODESCRIPTION or SPI_SETAUDIODESCRIPTION action value is specified.
BSMINFO	Contains information about a window that denied a request from BroadcastSystemMessageEx.
CBT_CREATEWNDA	Contains information passed to a WH_CBT hook procedure, CBTProc, before a window is created.
CBT_CREATEWNDW	Contains information passed to a WH_CBT hook procedure, CBTProc, before a window is created.
CBTACTIVATESTRUCT	Contains information passed to a WH_CBT hook procedure, CBTProc, before a window is activated.
CHANGEFILTERSTRUCT	Contains extended result information obtained by calling the ChangeWindowMessageFilterEx function.
CLIENTCREATESTRUCT	Contains information about the menu and first multiple-document interface (MDI) child window of an MDI client window.
COMBOBOXINFO	Contains combo box status information.
COMPAREITEMSTRUCT	Supplies the identifiers and application-supplied data for two items in a sorted, owner-drawn list box or combo box.
COPYDATASTRUCT	Contains data to be passed to another application by the WM_COPYDATA message.
CREATESTRUCTA	Defines the initialization parameters passed to the window procedure of an application. These members are identical to the parameters of the CreateWindowEx function.
CREATESTRUCTW	Defines the initialization parameters passed to the window procedure of an application. These members are identical to the parameters of the CreateWindowEx function.

CURSORINFO

Contains global cursor information.

CURSORSHAPE

Contains information about a cursor.

CWPRETSTRUCT

Defines the message parameters passed to a WH_CALLWNDPROCRET hook procedure, CallWndRetProc.

CWPSTRUCT

Defines the message parameters passed to a WH_CALLWNDPROC hook procedure, CallWndProc.

DEBUGHOOKINFO

Contains debugging information passed to a WH_DEBUG hook procedure, DebugProc.

DELETEITEMSTRUCT

Describes a deleted list box or combo box item.

DLGITEMTEMPLATE

Defines the dimensions and style of a control in a dialog box. One or more of these structures are combined with a DLGTEMPLATE structure to form a standard template for a dialog box.

DLGTEMPLATE

Defines the dimensions and style of a dialog box.

DRAWITEMSTRUCT

Provides information that the owner window uses to determine how to paint an owner-drawn control or menu item.

DRAWTEXTPARAMS

The DRAWTEXTPARAMS structure contains extended formatting options for the DrawTextEx function.

EVENTMSG

Contains information about a hardware message sent to the system message queue. This structure is used to store message information for the JournalPlaybackProc callback function.

FILTERKEYS

Contains information about the FilterKeys accessibility feature, which enables a user with disabilities to set the keyboard repeat rate (RepeatKeys), acceptance delay (SlowKeys), and bounce rate (BounceKeys).

FLASHWINFO

Contains the flash status for a window and the number of times the system should flash the window.

GESTURECONFIG

Gets and sets the configuration for enabling gesture messages and the type of this configuration.

GESTUREINFO

Stores information about a gesture.

GESTURENOTIFYSTRUCT

When transmitted with WM_GESTURENOTIFY messages, passes information about a gesture.

GUITHREADINFO

Contains information about a GUI thread.

HARDWAREINPUT

Contains information about a simulated message generated by an input device other than a keyboard or mouse.

HELPINFO

Contains information about an item for which context-sensitive help has been requested.

HELPWININFOA

Contains the size and position of either a primary or secondary Help window. An application can set this information by calling the WinHelp function with the HELP_SETWINPOS value.

HELPWININFOW

Contains the size and position of either a primary or secondary Help window. An application can set this information by calling the WinHelp function with the HELP_SETWINPOS value.

HIGHCONTRASTA

Contains information about the high contrast accessibility feature.

HIGHCONTRASTW

Contains information about the high contrast accessibility feature.

ICONINFO

Contains information about an icon or a cursor.

ICONINFOEXA

Contains information about an icon or a cursor. Extends ICONINFO. Used by GetIconInfoEx.

ICONINFOEXW

Contains information about an icon or a cursor. Extends ICONINFO. Used by GetIconInfoEx.

ICONMETRICSA

Contains the scalable metrics associated with icons. This structure is used with the SystemParametersInfo function when the SPI_GETICONMETRICS or SPI_SETICONMETRICS action is specified.

ICONMETRICSW

Contains the scalable metrics associated with icons. This structure is used with the SystemParametersInfo function when the SPI_GETICONMETRICS or SPI_SETICONMETRICS action is specified.

INPUT

Used by SendInput to store information for synthesizing input events such as keystrokes, mouse movement, and mouse clicks.

INPUT_INJECTION_VALUE

Contains the input injection details.

INPUT_MESSAGE_SOURCE

Contains information about the source of the input message.

INPUT_TRANSFORM

Defines the matrix that represents a transform on a message consumer.

KBDLLHOOKSTRUCT

Contains information about a low-level keyboard input event.

KEYBDINPUT

Contains information about a simulated keyboard event.

LASTINPUTINFO

Contains the time of the last input.

MDICREATESTRUCTA

Contains information about the class, title, owner, location, and size of a multiple-document interface (MDI) child window.

MDICREATESTRUCTW

Contains information about the class, title, owner, location, and size of a multiple-document interface (MDI) child window.

MDINEXTMENU

Contains information about the menu to be activated.

MEASUREITEMSTRUCT

Informs the system of the dimensions of an owner-drawn control or menu item. This allows the system to process user interaction with the control correctly.

MENUBARINFO	Contains menu bar information.
MENUGETOBJECTINFO	Contains information about the menu that the mouse cursor is on.
MENUINFO	Contains information about a menu.
MENUITEMINFOA	Contains information about a menu item.
MENUITEMINFOW	Contains information about a menu item.
MENUITEMTEMPLATE	Defines a menu item in a menu template.
MENUITEMTEMPLATEHEADER	Defines the header for a menu template. A complete menu template consists of a header and one or more menu item lists.
MINIMIZEDMETRICS	Contains the scalable metrics associated with minimized windows.
MINMAXINFO	Contains information about a window's maximized size and position and its minimum and maximum tracking size.
MONITORINFO	The MONITORINFO structure contains information about a display monitor.The GetMonitorInfo function stores information in a MONITORINFO structure or a MONITORINFOEX structure.The MONITORINFO structure is a subset of the MONITORINFOEX structure.
MONITORINFOEXA	The MONITORINFOEX structure contains information about a display monitor.The GetMonitorInfo function stores information into a MONITORINFOEX structure or a MONITORINFO structure.The MONITORINFOEX structure is a superset of the MONITORINFO structure.
MONITORINFOEXW	The MONITORINFOEX structure contains information about a display monitor.The GetMonitorInfo function stores information into a MONITORINFOEX structure or a MONITORINFO structure.The MONITORINFOEX structure is a superset of the MONITORINFO structure.

MOUSEHOOKSTRUCT	Contains information about a mouse event passed to a WH_MOUSE hook procedure, MouseProc.
MOUSEHOOKSTRUCTEX	Contains information about a mouse event passed to a WH_MOUSE hook procedure, MouseProc. This is an extension of the MOUSEHOOKSTRUCT structure that includes information about wheel movement or the use of the X button.
MOUSEINPUT	Contains information about a simulated mouse event.
MOUSEKEYS	Contains information about the MouseKeys accessibility feature.
MOUSEMOVEPOINT	Contains information about the mouse's location in screen coordinates.
MSG	Contains message information from a thread's message queue.
MSGBOXPARAMSA	Contains information used to display a message box. The MessageBoxIndirect function uses this structure.
MSGBOXPARAMSW	Contains information used to display a message box. The MessageBoxIndirect function uses this structure.
MSLLHOOKSTRUCT	Contains information about a low-level mouse input event.
MULTIKEYHELPA	Specifies a keyword to search for and the keyword table to be searched by Windows Help.
MULTIKEYHELPW	Specifies a keyword to search for and the keyword table to be searched by Windows Help.
NCCALCSIZE_PARAMS	Contains information that an application can use while processing the WM_NCCALCSIZE message to calculate the size, position, and valid contents of the client area of a window.
NMHDR	Contains information about a notification message.

NONCLIENTMETRICSA	Contains the scalable metrics associated with the nonclient area of a nonminimized window.
NONCLIENTMETRICSW	Contains the scalable metrics associated with the nonclient area of a nonminimized window.
PAINTSTRUCT	The PAINTSTRUCT structure contains information for an application. This information can be used to paint the client area of a window owned by that application.
POINTER_DEVICE_CURSOR_INFO	Contains cursor ID mappings for pointer devices.
POINTER_DEVICE_INFO	Contains information about a pointer device. An array of these structures is returned from the GetPointerDevices function. A single structure is returned from a call to the GetPointerDevice function.
POINTER_DEVICE_PROPERTY	Contains pointer-based device properties (Human Interface Device (HID) global items that correspond to HID usages).
POINTER_INFO	Contains basic pointer information common to all pointer types. Applications can retrieve this information using the GetPointerInfo, GetPointerFrameInfo, GetPointerInfoHistory and GetPointerFrameInfoHistory functions.
POINTER_PEN_INFO	Defines basic pen information common to all pointer types.
POINTER_TOUCH_INFO	Defines basic touch information common to all pointer types.
POINTER_TYPE_INFO	Contains information about the pointer input type.
POWERBROADCAST_SETTING	Sent with a power setting event and contains data about the specific change.
RAWHID	Describes the format of the raw input from a Human Interface Device (HID).
RAWINPUT	Contains the raw input from a device.

RAWINPUTDEVICE	Defines information for the raw input devices.
RAWINPUTDEVICELIST	Contains information about a raw input device.
RAWINPUTHEADER	Contains the header information that is part of the raw input data.
RAWKEYBOARD	Contains information about the state of the keyboard.
RAWMOUSE	Contains information about the state of the mouse.
RID_DEVICE_INFO	Defines the raw input data coming from any device.
RID_DEVICE_INFO_HID	Defines the raw input data coming from the specified Human Interface Device (HID).
RID_DEVICE_INFO_KEYBOARD	Defines the raw input data coming from the specified keyboard.
RID_DEVICE_INFO_MOUSE	Defines the raw input data coming from the specified mouse.
SCROLLBARINFO	The SCROLLBARINFO structure contains scroll bar information.
SCROLLINFO	The SCROLLINFO structure contains scroll bar parameters to be set by the SetScrollInfo function (or SBM_SETSCROLLINFO message), or retrieved by the GetScrollInfo function (or SBM_GETSCROLLINFO message).
SERIALKEYSA	Contains information about the SerialKeys accessibility feature, which interprets data from a communication aid attached to a serial port as commands causing the system to simulate keyboard and mouse input.
SERIALKEYSW	Contains information about the SerialKeys accessibility feature, which interprets data from a communication aid attached to a serial port as commands causing the system to simulate keyboard and mouse input.

SOUNDSENTRYA	Contains information about the SoundSentry accessibility feature. When the SoundSentry feature is on, the computer displays a visual indication only when a sound is generated.
SOUNDSENTRYW	Contains information about the SoundSentry accessibility feature. When the SoundSentry feature is on, the computer displays a visual indication only when a sound is generated.
STICKYKEYS	Contains information about the StickyKeys accessibility feature.
STYLESTRUCT	Contains the styles for a window.
TITLEBARINFO	Contains title bar information.
TITLEBARINFOEX	Expands on the information described in the TITLEBARINFO structure by including the coordinates of each element of the title bar.
TOGGLEKEYS	Contains information about the ToggleKeys accessibility feature.
TOUCH_HIT_TESTING_INPUT	Contains information about the touch contact area reported by the touch digitizer.
TOUCH_HIT_TESTING_PROXIMITY_EVALUATION	Contains the hit test score that indicates whether the object is the likely target of the touch contact area, relative to other objects that intersect the touch contact area.
TOUCHINPUT	Encapsulates data for touch input.
TOUCHPREDICTIONPARAMETERS	Contains hardware input details that can be used to predict touch targets and help compensate for hardware latency when processing touch and gesture input that contains distance and velocity data.
TPMPARAMS	Contains extended parameters for the TrackPopupMenuEx function.

TRACKMOUSEEVENT	Used by the TrackMouseEvent function to track when the mouse pointer leaves a window or hovers over a window for a specified amount of time.
UPDATELAYEREDWINDOWINFO	Used by UpdateLayeredWindowIndirect to provide position, size, shape, content, and translucency information for a layered window.
USAGE_PROPERTIES	Contains device properties (Human Interface Device (HID) global items that correspond to HID usages) for any type of HID input device.
USEROBJECTFLAGS	Contains information about a window station or desktop handle.
WINDOWINFO	Contains window information.
WINDOWPLACEMENT	Contains information about the placement of a window on the screen.
WINDOWPOS	Contains information about the size and position of a window.
WNDCLASSA	Contains the window class attributes that are registered by the RegisterClass function.
WNDCLASSEXA	Contains window class information.
WNDCLASSEXW	Contains window class information.
WNDCLASSW	Contains the window class attributes that are registered by the RegisterClass function.
WTSSESSION_NOTIFICATION	Provides information about the session change notification. A service receives this structure in its HandlerEx function in response to a session change event.

Enumerations

AR_STATE	Indicates the state of screen auto-rotation for the system. For example, whether auto-rotation is supported, and whether it is enabled by the user.
DIALOG_CONTROL_DPI_CHANGE_BEHAVIORS	Describes per-monitor DPI scaling behavior overrides for child windows within dialogs. The values in this enumeration are bitfields and can be combined.
DIALOG_DPI_CHANGE_BEHAVIORS	In Per Monitor v2 contexts, dialogs will automatically respond to DPI changes by resizing themselves and re-computing the positions of their child windows (here referred to as re-layouting).
FEEDBACK_TYPE	Specifies the visual feedback associated with an event.
INPUT_MESSAGE_DEVICE_TYPE	The type of device that sent the input message.
INPUT_MESSAGE_ORIGIN_ID	The ID of the input message source.
ORIENTATION_PREFERENCE	Indicates the screen orientation preference for a desktop app process.
POINTER_BUTTON_CHANGE_TYPE	Identifies a change in the state of a button associated with a pointer.
POINTER_DEVICE_CURSOR_TYPE	Identifies the pointer device cursor types.
POINTER_DEVICE_TYPE	Identifies the pointer device types.
POINTER_FEEDBACK_MODE	Identifies the visual feedback behaviors available to CreateSyntheticPointerDevice.
tagPOINTER_INPUT_TYPE	Identifies the pointer input types.
TOOLTIP_DISMISS_FLAGS	

ActivateKeyboardLayout function (winuser.h)

7/6/2022 • 3 minutes to read • Edit Online

Sets the input locale identifier (formerly called the keyboard layout handle) for the calling thread or the current process. The input locale identifier specifies a locale as well as the physical layout of the keyboard.

Syntax

```
HKL ActivateKeyboardLayout(
  [in] HKL  hkl,
  [in] UINT Flags
);
```

Parameters

`[in] hkl`

Type: **HKL**

Input locale identifier to be activated.

The input locale identifier must have been loaded by a previous call to the LoadKeyboardLayout function. This parameter must be either the handle to a keyboard layout or one of the following values.

VALUE	MEANING
HKL_NEXT 1	Selects the next locale identifier in the circular list of loaded locale identifiers maintained by the system.
HKL_PREV 0	Selects the previous locale identifier in the circular list of loaded locale identifiers maintained by the system.

`[in] Flags`

Type: **UINT**

Specifies how the input locale identifier is to be activated. This parameter can be one of the following values.

VALUE	MEANING

KLF_REORDER 0x00000008	If this bit is set, the system's circular list of loaded locale identifiers is reordered by moving the locale identifier to the head of the list. If this bit is not set, the list is rotated without a change of order. For example, if a user had an English locale identifier active, as well as having French, German, and Spanish locale identifiers loaded (in that order), then activating the German locale identifier with the **KLF_REORDER** bit set would produce the following order: German, English, French, Spanish. Activating the German locale identifier without the **KLF_REORDER** bit set would produce the following order: German, Spanish, English, French. If less than three locale identifiers are loaded, the value of this flag is irrelevant.
KLF_RESET 0x40000000	If set but **KLF_SHIFTLOCK** is not set, the Caps Lock state is turned off by pressing the Caps Lock key again. If set and **KLF_SHIFTLOCK** is also set, the Caps Lock state is turned off by pressing either SHIFT key. These two methods are mutually exclusive, and the setting persists as part of the User's profile in the registry.
KLF_SETFORPROCESS 0x00000100	Activates the specified locale identifier for the entire process and sends the WM_INPUTLANGCHANGE message to the current thread's focus or active window.
KLF_SHIFTLOCK 0x00010000	This is used with **KLF_RESET**. See **KLF_RESET** for an explanation.
KLF_UNLOADPREVIOUS	This flag is unsupported. Use the UnloadKeyboardLayout function instead.

Return value

Type: **HKL**

The return value is of type **HKL**. If the function succeeds, the return value is the previous input locale identifier. Otherwise, it is zero.

To get extended error information, use the GetLastError function.

Remarks

This function only affects the layout for the current process or thread.

This function is not restricted to keyboard layouts. The *hkl* parameter is actually an input locale identifier. This is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input. Several input locale identifiers can be loaded at any one time, but only one is active at a time. Loading multiple input locale identifiers makes it possible to rapidly switch between them.

When multiple IMEs are allowed for each locale, passing an input locale identifier in which the high word (the device handle) is zero activates the first IME in the list belonging to the locale.

The **KLF_RESET** and **KLF_SHIFTLOCK** flags alter the method by which the Caps Lock state is turned off. By default, the Caps Lock state is turned off by hitting the Caps Lock key again. If only **KLF_RESET** is set, the default state is reestablished. If **KLF_RESET** and **KLF_SHIFTLOCK** are set, the Caps Lock state is turned off by pressing either Caps Lock key. This feature is used to conform to local keyboard behavior standards as well as for personal preferences.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetKeyboardLayoutName

Keyboard Input

LoadKeyboardLayout

Reference

UnloadKeyboardLayout

BlockInput function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Blocks keyboard and mouse input events from reaching applications.

Syntax

```
BOOL BlockInput(
  [in] BOOL fBlockIt
);
```

Parameters

`[in] fBlockIt`

Type: **BOOL**

The function's purpose. If this parameter is **TRUE**, keyboard and mouse input events are blocked. If this parameter is **FALSE**, keyboard and mouse events are unblocked. Note that only the thread that blocked input can successfully unblock input.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If input is already blocked, the return value is zero. To get extended error information, call GetLastError.

Remarks

When input is blocked, real physical input from the mouse or keyboard will not affect the input queue's synchronous key state (reported by GetKeyState and GetKeyboardState), nor will it affect the asynchronous key state (reported by GetAsyncKeyState). However, the thread that is blocking input can affect both of these key states by calling SendInput. No other thread can do this.

The system will unblock input in the following cases:

- The thread that blocked input unexpectedly exits without calling **BlockInput** with *fBlock* set to **FALSE**. In this case, the system cleans up properly and re-enables input.
- The user presses CTRL+ALT+DEL or the system invokes the **Hard System Error** modal message box (for example, when a program faults or a device fails).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

GetKeyState

GetKeyboardState

Keyboard Input

Reference

SendInput

DefRawInputProc function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Unlike DefWindowProcA and DefWindowProcW, this function doesn't do any processing.

DefRawInputProc only checks whether **cbSizeHeader**'s value corresponds to the expected size of RAWINPUTHEADER.

Syntax

```
LRESULT DefRawInputProc(
  [in] PRAWINPUT *paRawInput,
  [in] INT       nInput,
  [in] UINT      cbSizeHeader
);
```

Parameters

`[in] paRawInput`

Type: **PRAWINPUT***

Ignored.

`[in] nInput`

Type: **INT**

Ignored.

`[in] cbSizeHeader`

Type: **UINT**

The size, in bytes, of the RAWINPUTHEADER structure.

Return value

Type: **LRESULT**

If successful, the function returns **0**. Otherwise it returns **-1**.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows

Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-rawinput-l1-1-0 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

RAWINPUT

RAWINPUTHEADER

Raw Input

DragDetect function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Captures the mouse and tracks its movement until the user releases the left button, presses the ESC key, or moves the mouse outside the drag rectangle around the specified point. The width and height of the drag rectangle are specified by the **SM_CXDRAG** and **SM_CYDRAG** values returned by the GetSystemMetrics function.

Syntax

```
BOOL DragDetect(
  [in] HWND  hwnd,
  [in] POINT pt
);
```

Parameters

`[in] hwnd`

Type: **HWND**

A handle to the window receiving mouse input.

`[in] pt`

Type: **POINT**

Initial position of the mouse, in screen coordinates. The function determines the coordinates of the drag rectangle by using this point.

Return value

Type: **BOOL**

If the user moved the mouse outside of the drag rectangle while holding down the left button, the return value is nonzero.

If the user did not move the mouse outside of the drag rectangle while holding down the left button, the return value is zero.

Remarks

The system metrics for the drag rectangle are configurable, allowing for larger or smaller drag rectangles.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetSystemMetrics

Mouse Input

POINT

Reference

EnableWindow function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Enables or disables mouse and keyboard input to the specified window or control. When input is disabled, the window does not receive input such as mouse clicks and key presses. When input is enabled, the window receives all input.

Syntax

```
BOOL EnableWindow(
  [in] HWND hWnd,
  [in] BOOL bEnable
);
```

Parameters

`[in] hWnd`

Type: **HWND**

A handle to the window to be enabled or disabled.

`[in] bEnable`

Type: **BOOL**

Indicates whether to enable or disable the window. If this parameter is **TRUE**, the window is enabled. If the parameter is **FALSE**, the window is disabled.

Return value

Type: **BOOL**

If the window was previously disabled, the return value is nonzero.

If the window was not previously disabled, the return value is zero.

Remarks

If the window is being disabled, the system sends a WM_CANCELMODE message. If the enabled state of a window is changing, the system sends a WM_ENABLE message after the **WM_CANCELMODE** message. (These messages are sent before **EnableWindow** returns.) If a window is already disabled, its child windows are implicitly disabled, although they are not sent a **WM_ENABLE** message.

A window must be enabled before it can be activated. For example, if an application is displaying a modeless dialog box and has disabled its main window, the application must enable the main window before destroying the dialog box. Otherwise, another window will receive the keyboard focus and be activated. If a child window is disabled, it is ignored when the system tries to determine which window should receive mouse messages.

By default, a window is enabled when it is created. To create a window that is initially disabled, an application can specify the **WS_DISABLED** style in the CreateWindow or CreateWindowEx function. After a window has been created, an application can use **EnableWindow** to enable or disable the window.

An application can use this function to enable or disable a control in a dialog box. A disabled control cannot receive the keyboard focus, nor can a user gain access to it.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-window-l1-1-4 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

CreateWindow

CreateWindowEx

IsWindowEnabled

Keyboard Input

Reference

WM_ENABLE

GET_APPCOMMAND_LPARAM macro (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the application command from the specified **LPARAM** value.

Syntax

```
void GET_APPCOMMAND_LPARAM(
   lParam
);
```

Parameters

`lParam`

The value to be converted.

Return value

None

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

Mouse Input Overview

GET_DEVICE_LPARAM macro (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the input device type from the specified **LPARAM** value.

Syntax

```
void GET_DEVICE_LPARAM(
   lParam
);
```

Parameters

`lParam`

The value to be converted.

Return value

None

Remarks

This macro is identical to the GET_MOUSEORKEY_LPARAM macro.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

Conceptual

GET_MOUSEORKEY_LPARAM

Mouse Input

Reference

GET_FLAGS_LPARAM macro (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the state of certain virtual keys from the specified **LPARAM** value.

Syntax

```
void GET_FLAGS_LPARAM(
   lParam
);
```

Parameters

`lParam`

The value to be converted.

Return value

None

Remarks

This macro is identical to the GET_KEYSTATE_LPARAM macro.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

Conceptual

GET_KEYSTATE_LPARAM

Mouse Input

Reference

GET_KEYSTATE_LPARAM macro (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the state of certain virtual keys from the specified **LPARAM** value.

Syntax

```
void GET_KEYSTATE_LPARAM(
   lParam
);
```

Parameters

`lParam`

The value to be converted.

Return value

None

Remarks

This macro is identical to the GET_FLAGS_LPARAM macro.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

Conceptual

GET_FLAGS_LPARAM

Mouse Input

Reference

GET_KEYSTATE_WPARAM macro (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the state of certain virtual keys from the specified **WPARAM** value.

Syntax

```
void GET_KEYSTATE_WPARAM(
   wParam
);
```

Parameters

`wParam`

The value to be converted.

Return value

None

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

Mouse Input Overview

GET_NCHITTEST_WPARAM macro (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the hit-test value from the specified **WPARAM** value.

Syntax

```
void GET_NCHITTEST_WPARAM(
   wParam
);
```

Parameters

`wParam`

The value to be converted.

Return value

None

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

Mouse Input Overview

GET_RAWINPUT_CODE_WPARAM macro (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the input code from *wParam* in WM_INPUT message.

Syntax

```
void GET_RAWINPUT_CODE_WPARAM(
   wParam
);
```

Parameters

`wParam`

wParam from WM_INPUT message.

Return value

Input code value. Can be one of the following:

VALUE	MEANING
RIM_INPUT 0	Input occurred while the application was in the foreground.
RIM_INPUTSINK 1	Input occurred while the application was not in the foreground.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

Conceptual

RAWINPUT

Raw Input

Reference

WM_INPUT

GET_WHEEL_DELTA_WPARAM macro (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the wheel-delta value from the specified **WPARAM** value.

Syntax

```
void GET_WHEEL_DELTA_WPARAM(
   wParam
);
```

Parameters

`wParam`

The value to be converted.

Return value

None

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

Mouse Input Overview

GET_XBUTTON_WPARAM macro (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the state of certain buttons from the specified **WPARAM** value.

Syntax

```
void GET_XBUTTON_WPARAM(
   wParam
);
```

Parameters

`wParam`

The value to be converted.

Return value

None

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

Mouse Input Overview

GetActiveWindow function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the window handle to the active window attached to the calling thread's message queue.

Syntax

```
HWND GetActiveWindow();
```

Return value

Type: **HWND**

The return value is the handle to the active window attached to the calling thread's message queue. Otherwise, the return value is **NULL**.

Remarks

To get the handle to the foreground window, you can use GetForegroundWindow.

To get the window handle to the active window in the message queue for another thread, use GetGUIThreadInfo.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-window-l1-1-4 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

GetForegroundWindow

GetGUIThreadInfo

Keyboard Input

Reference

SetActiveWindow

GetAsyncKeyState function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to **GetAsyncKeyState**.

Syntax

```
SHORT GetAsyncKeyState(
  [in] int vKey
);
```

Parameters

`[in] vKey`

Type: **int**

The virtual-key code. For more information, see Virtual Key Codes.

You can use left- and right-distinguishing constants to specify certain keys. See the Remarks section for further information.

Return value

Type: **SHORT**

If the function succeeds, the return value specifies whether the key was pressed since the last call to **GetAsyncKeyState**, and whether the key is currently up or down. If the most significant bit is set, the key is down, and if the least significant bit is set, the key was pressed after the previous call to **GetAsyncKeyState**. However, you should not rely on this last behavior; for more information, see the Remarks.

The return value is zero for the following cases:

- The current desktop is not the active desktop
- The foreground thread belongs to another process and the desktop does not allow the hook or the journal record.

Remarks

The **GetAsyncKeyState** function works with mouse buttons. However, it checks on the state of the physical mouse buttons, not on the logical mouse buttons that the physical buttons are mapped to. For example, the call **GetAsyncKeyState**(VK_LBUTTON) always returns the state of the left physical mouse button, regardless of whether it is mapped to the left or right logical mouse button. You can determine the system's current mapping of physical mouse buttons to logical mouse buttons by calling `GetSystemMetrics(SM_SWAPBUTTON)`.

which returns TRUE if the mouse buttons have been swapped.

Although the least significant bit of the return value indicates whether the key has been pressed since the last query, due to the preemptive multitasking nature of Windows, another application can call **GetAsyncKeyState** and receive the "recently pressed" bit instead of your application. The behavior of the least significant bit of the return value is retained strictly for compatibility with 16-bit Windows applications (which are non-preemptive) and should not be relied upon.

You can use the virtual-key code constants **VK_SHIFT**, **VK_CONTROL**, and **VK_MENU** as values for the *vKey* parameter. This gives the state of the SHIFT, CTRL, or ALT keys without distinguishing between left and right.

You can use the following virtual-key code constants as values for *vKey* to distinguish between the left and right instances of those keys.

CODE	MEANING
VK_LSHIFT	Left-shift key.
VK_RSHIFT	Right-shift key.
VK_LCONTROL	Left-control key.
VK_RCONTROL	Right-control key.
VK_LMENU	Left-menu key.
VK_RMENU	Right-menu key.

These left- and right-distinguishing constants are only available when you call the GetKeyboardState, SetKeyboardState, **GetAsyncKeyState**, GetKeyState, and MapVirtualKey functions.

Example

```
while (GetMessage(&msg, nullptr, 0, 0))
{
    if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    switch (msg.message)
    {
    case WM_KEYDOWN:
        if ((GetAsyncKeyState(VK_ESCAPE) & 0x01) && bRunning)
        {
            Stop();
        }
        break;
    }
}
```

Example from Windows Classic Samples on GitHub.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetKeyState

GetKeyboardState

GetSystemMetrics

Keyboard Input

MapVirtualKey

Other Resources

Reference

SetKeyboardState

GetCapture function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves a handle to the window (if any) that has captured the mouse. Only one window at a time can capture the mouse; this window receives mouse input whether or not the cursor is within its borders.

Syntax

```
HWND GetCapture();
```

Return value

Type: **HWND**

The return value is a handle to the capture window associated with the current thread. If no window in the thread has captured the mouse, the return value is **NULL**.

Remarks

A **NULL** return value means the current thread has not captured the mouse. However, it is possible that another thread or process has captured the mouse.

To get a handle to the capture window on another thread, use the GetGUIThreadInfo function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-mouse-l1-1-0 (introduced in Windows 8)

See also

Conceptual

GetGUIThreadInfo

Mouse Input

Reference

ReleaseCapture

SetCapture

GetDoubleClickTime function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the current double-click time for the mouse. A double-click is a series of two clicks of the mouse button, the second occurring within a specified time after the first. The double-click time is the maximum number of milliseconds that may occur between the first and second click of a double-click. The maximum double-click time is 5000 milliseconds.

Syntax

```
UINT GetDoubleClickTime();
```

Return value

Type: **UINT**

The return value specifies the current double-click time, in milliseconds. The maximum return value is 5000 milliseconds.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-mouse-l1-1-0 (introduced in Windows 8)

See also

Conceptual

Mouse Input

Reference

SetDoubleClickTime

GetFocus function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the handle to the window that has the keyboard focus, if the window is attached to the calling thread's message queue.

Syntax

```
HWND GetFocus();
```

Return value

Type: **HWND**

The return value is the handle to the window with the keyboard focus. If the calling thread's message queue does not have an associated window with the keyboard focus, the return value is **NULL**.

Remarks

GetFocus returns the window with the keyboard focus for the current thread's message queue. If **GetFocus** returns **NULL**, another thread's queue may be attached to a window that has the keyboard focus.

Use the GetForegroundWindow function to retrieve the handle to the window with which the user is currently working. You can associate your thread's message queue with the windows owned by another thread by using the AttachThreadInput function.

To get the window with the keyboard focus on the foreground queue or the queue of another thread, use the GetGUIThreadInfo function.

Examples

For an example, see "Creating a Combo Box Toolbar" in Using Combo Boxes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

API set	ext-ms-win-ntuser-window-l1-1-4 (introduced in Windows 10, version 10.0.14393)

See also

AttachThreadInput

Conceptual

GetForegroundWindow

GetGUIThreadInfo

Keyboard Input

Other Resources

Reference

SetFocus

WM_KILLFOCUS

WM_SETFOCUS

GetKBCodePage function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the current code page.

> **Note** This function is provided only for compatibility with 16-bit versions of Windows. Applications should use the GetOEMCP function to retrieve the OEM code-page identifier for the system.

Syntax

```
UINT GetKBCodePage();
```

Return value

Type: **UINT**

The return value is an OEM code-page identifier, or it is the default identifier if the registry value is not readable. For a list of OEM code-page identifiers, see Code Page Identifiers.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetACP

GetOEMCP

Keyboard Input

Reference

GetKeyboardLayout function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the active input locale identifier (formerly called the keyboard layout).

Syntax

```
HKL GetKeyboardLayout(
  [in] DWORD idThread
);
```

Parameters

`[in] idThread`

Type: **DWORD**

The identifier of the thread to query, or 0 for the current thread.

Return value

Type: **HKL**

The return value is the input locale identifier for the thread. The low word contains a Language Identifier for the input language and the high word contains a device handle to the physical layout of the keyboard.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

Since the keyboard layout can be dynamically changed, applications that cache information about the current keyboard layout should process the WM_INPUTLANGCHANGE message to be informed of changes in the input language.

To get the KLID (keyboard layout ID) of the currently active HKL, call the GetKeyboardLayoutName.

Beginning in Windows 8: The preferred method to retrieve the language associated with the current keyboard layout or input method is a call to Windows.Globalization.Language.CurrentInputMethodLanguageTag. If your app passes language tags from **CurrentInputMethodLanguageTag** to any National Language Support functions, it must first convert the tags by calling ResolveLocaleName.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

ActivateKeyboardLayout

Conceptual

CreateThread

Keyboard Input

LoadKeyboardLayout

Other Resources

Reference

WM_INPUTLANGCHANGE

GetKeyboardLayoutList function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the input locale identifiers (formerly called keyboard layout handles) corresponding to the current set of input locales in the system. The function copies the identifiers to the specified buffer.

Syntax

```
int GetKeyboardLayoutList(
  [in]  int nBuff,
  [out] HKL *lpList
);
```

Parameters

`[in] nBuff`

Type: **int**

The maximum number of handles that the buffer can hold.

`[out] lpList`

Type: **HKL***

A pointer to the buffer that receives the array of input locale identifiers.

Return value

Type: **int**

If the function succeeds, the return value is the number of input locale identifiers copied to the buffer or, if *nBuff* is zero, the return value is the size, in array elements, of the buffer needed to receive all current input locale identifiers.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

Beginning in Windows 8: The preferred method to retrieve the language associated with the current keyboard layout or input method is a call to Windows.Globalization.Language.CurrentInputMethodLanguageTag. If your app passes language tags from **CurrentInputMethodLanguageTag** to any National Language Support functions, it must first convert the tags by calling ResolveLocaleName.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetKeyboardLayout

Keyboard Input

Reference

GetKeyboardLayoutNameA function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the name of the active input locale identifier (formerly called the keyboard layout) for the system.

Syntax

```
BOOL GetKeyboardLayoutNameA(
  [out] LPSTR pwszKLID
);
```

Parameters

`[out] pwszKLID`

Type: **LPTSTR**

The buffer (of at least **KL_NAMELENGTH** characters in length) that receives the name of the input locale identifier, including the terminating null character. This will be a copy of the string provided to the LoadKeyboardLayout function, unless layout substitution took place.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

Beginning in Windows 8: The preferred method to retrieve the language associated with the current keyboard layout or input method is a call to Windows.Globalization.Language.CurrentInputMethodLanguageTag. If your app passes language tags from **CurrentInputMethodLanguageTag** to any National Language Support functions, it must first convert the tags by calling ResolveLocaleName.

NOTE

The winuser.h header defines GetKeyboardLayoutName as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

ActivateKeyboardLayout

Conceptual

Keyboard Input

LoadKeyboardLayout

Reference

UnloadKeyboardLayout

GetKeyboardLayoutNameW function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the name of the active input locale identifier (formerly called the keyboard layout) for the system.

Syntax

```
BOOL GetKeyboardLayoutNameW(
  [out] LPWSTR pwszKLID
);
```

Parameters

`[out] pwszKLID`

Type: **LPTSTR**

The buffer (of at least **KL_NAMELENGTH** characters in length) that receives the name of the input locale identifier, including the terminating null character. This will be a copy of the string provided to the LoadKeyboardLayout function, unless layout substitution took place.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

Beginning in Windows 8: The preferred method to retrieve the language associated with the current keyboard layout or input method is a call to Windows.Globalization.Language.CurrentInputMethodLanguageTag. If your app passes language tags from **CurrentInputMethodLanguageTag** to any National Language Support functions, it must first convert the tags by calling ResolveLocaleName.

NOTE

The winuser.h header defines GetKeyboardLayoutName as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

ActivateKeyboardLayout

Conceptual

Keyboard Input

LoadKeyboardLayout

Reference

UnloadKeyboardLayout

GetKeyboardState function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Copies the status of the 256 virtual keys to the specified buffer.

Syntax

```
BOOL GetKeyboardState(
  [out] PBYTE lpKeyState
);
```

Parameters

`[out] lpKeyState`

Type: **PBYTE**

The 256-byte array that receives the status data for each virtual key.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

An application can call this function to retrieve the current status of all the virtual keys. The status changes as a thread removes keyboard messages from its message queue. The status does not change as keyboard messages are posted to the thread's message queue, nor does it change as keyboard messages are posted to or retrieved from message queues of other threads. (Exception: Threads that are connected through AttachThreadInput share the same keyboard state.)

When the function returns, each member of the array pointed to by the *lpKeyState* parameter contains status data for a virtual key. If the high-order bit is 1, the key is down; otherwise, it is up. If the key is a toggle key, for example CAPS LOCK, then the low-order bit is 1 when the key is toggled and is 0 if the key is untoggled. The low-order bit is meaningless for non-toggle keys. A toggle key is said to be toggled when it is turned on. A toggle key's indicator light (if any) on the keyboard will be on when the key is toggled, and off when the key is untoggled.

To retrieve status information for an individual key, use the GetKeyState function. To retrieve the current state for an individual key regardless of whether the corresponding keyboard message has been retrieved from the message queue, use the GetAsyncKeyState function.

An application can use the virtual-key code constants **VK_SHIFT**, **VK_CONTROL** and **VK_MENU** as indices into the array pointed to by *lpKeyState*. This gives the status of the SHIFT, CTRL, or ALT keys without distinguishing between left and right. An application can also use the following virtual-key code constants as indices to distinguish between the left and right instances of those keys:

VK_LSHIFT
VK_RSHIFT
VK_LCONTROL
VK_RCONTROL
VK_LMENU
VK_RMENU

These left- and right-distinguishing constants are available to an application only through the **GetKeyboardState**, SetKeyboardState, GetAsyncKeyState, GetKeyState, and MapVirtualKey functions.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-rawinput-l1-1-0 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

GetAsyncKeyState

GetKeyState

Keyboard Input

MapVirtualKey

Reference

SetKeyboardState

GetKeyboardType function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves information about the current keyboard.

Syntax

```
int GetKeyboardType(
  [in] int nTypeFlag
);
```

Parameters

`[in] nTypeFlag`

Type: **int**

The type of keyboard information to be retrieved. This parameter can be one of the following values.

VALUE	MEANING
0	Keyboard type
1	Keyboard subtype
2	The number of function keys on the keyboard

Return value

Type: **int**

If the function succeeds, the return value specifies the requested information.

If the function fails and *nTypeFlag* is not 1, the return value is 0; 0 is a valid return value when *nTypeFlag* is 1 (keyboard subtype). To get extended error information, call GetLastError.

Remarks

Valid keyboard types are:

VALUE	DESCRIPTION
0x4	Enhanced 101- or 102-key keyboards (and compatibles)
0x7	Japanese Keyboard
0x8	Korean Keyboard
0x51	Unknown type or HID keyboard

Keyboard subtypes are original equipment manufacturer (OEM)-dependent values.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Keyboard Input Functions

GetKeyNameTextA function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves a string that represents the name of a key.

Syntax

```
int GetKeyNameTextA(
  [in]  LONG  lParam,
  [out] LPSTR lpString,
  [in]  int   cchSize
);
```

Parameters

`[in] lParam`

Type: **LONG**

The second parameter of the keyboard message (such as WM_KEYDOWN) to be processed. The function interprets the following bit positions in the *lParam*.

BITS	MEANING
16-23	Scan code.
24	Extended-key flag. Distinguishes some keys on an enhanced keyboard.
25	"Do not care" bit. The application calling this function sets this bit to indicate that the function should not distinguish between left and right CTRL and SHIFT keys, for example.

`[out] lpString`

Type: **LPTSTR**

The buffer that will receive the key name.

`[in] cchSize`

Type: **int**

The maximum length, in characters, of the key name, including the terminating null character. (This parameter should be equal to the size of the buffer pointed to by the *lpString* parameter.)

Return value

Type: **int**

If the function succeeds, a null-terminated string is copied into the specified buffer, and the return value is the length of the string, in characters, not counting the terminating null character.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The format of the key-name string depends on the current keyboard layout. The keyboard driver maintains a list of names in the form of character strings for keys with names longer than a single character. The key name is translated according to the layout of the currently installed keyboard, thus the function may give different results for different input locales. The name of a character key is the character itself. The names of dead keys are spelled out in full.

NOTE

The winuser.h header defines GetKeyNameText as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Keyboard Input

GetKeyNameTextW function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves a string that represents the name of a key.

Syntax

```
int GetKeyNameTextW(
  [in]  LONG   lParam,
  [out] LPWSTR lpString,
  [in]  int    cchSize
);
```

Parameters

`[in] lParam`

Type: **LONG**

The second parameter of the keyboard message (such as WM_KEYDOWN) to be processed. The function interprets the following bit positions in the *lParam*.

BITS	MEANING
16-23	Scan code.
24	Extended-key flag. Distinguishes some keys on an enhanced keyboard.
25	"Do not care" bit. The application calling this function sets this bit to indicate that the function should not distinguish between left and right CTRL and SHIFT keys, for example.

`[out] lpString`

Type: **LPTSTR**

The buffer that will receive the key name.

`[in] cchSize`

Type: **int**

The maximum length, in characters, of the key name, including the terminating null character. (This parameter should be equal to the size of the buffer pointed to by the *lpString* parameter.)

Return value

Type: **int**

If the function succeeds, a null-terminated string is copied into the specified buffer, and the return value is the length of the string, in characters, not counting the terminating null character.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The format of the key-name string depends on the current keyboard layout. The keyboard driver maintains a list of names in the form of character strings for keys with names longer than a single character. The key name is translated according to the layout of the currently installed keyboard, thus the function may give different results for different input locales. The name of a character key is the character itself. The names of dead keys are spelled out in full.

> **NOTE**
>
> The winuser.h header defines GetKeyNameText as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Keyboard Input

GetKeyState function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the status of the specified virtual key. The status specifies whether the key is up, down, or toggled (on, off—alternating each time the key is pressed).

Syntax

```
SHORT GetKeyState(
  [in] int nVirtKey
);
```

Parameters

`[in] nVirtKey`

Type: **int**

A virtual key. If the desired virtual key is a letter or digit (A through Z, a through z, or 0 through 9), *nVirtKey* must be set to the ASCII value of that character. For other keys, it must be a virtual-key code.

If a non-English keyboard layout is used, virtual keys with values in the range ASCII A through Z and 0 through 9 are used to specify most of the character keys. For example, for the German keyboard layout, the virtual key of value ASCII O (0x4F) refers to the "o" key, whereas VK_OEM_1 refers to the "o with umlaut" key.

Return value

Type: **SHORT**

The return value specifies the status of the specified virtual key, as follows:

- If the high-order bit is 1, the key is down; otherwise, it is up.
- If the low-order bit is 1, the key is toggled. A key, such as the CAPS LOCK key, is toggled if it is turned on. The key is off and untoggled if the low-order bit is 0. A toggle key's indicator light (if any) on the keyboard will be on when the key is toggled, and off when the key is untoggled.

Remarks

The key status returned from this function changes as a thread reads key messages from its message queue. The status does not reflect the interrupt-level state associated with the hardware. Use the GetAsyncKeyState function to retrieve that information.

An application calls **GetKeyState** in response to a keyboard-input message. This function retrieves the state of the key when the input message was generated.

To retrieve state information for all the virtual keys, use the GetKeyboardState function.

An application can use the virtual key code constants **VK_SHIFT**, **VK_CONTROL**, and **VK_MENU** as values for the *nVirtKey* parameter. This gives the status of the SHIFT, CTRL, or ALT keys without distinguishing between left and right. An application can also use the following virtual-key code constants as values for *nVirtKey* to distinguish between the left and right instances of those keys:

VK_LSHIFT VK_RSHIFT VK_LCONTROL VK_RCONTROL VK_LMENU VK_RMENU These left- and right-distinguishing constants are available to an application only through the GetKeyboardState, SetKeyboardState, GetAsyncKeyState, **GetKeyState**, and MapVirtualKey functions.

Examples

For an example, see Displaying Keyboard Input.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

GetKeyboardState

Keyboard Input

MapVirtualKey

Reference

SetKeyboardState

GetLastInputInfo function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the time of the last input event.

Syntax

```
BOOL GetLastInputInfo(
  [out] PLASTINPUTINFO plii
);
```

Parameters

`[out] plii`

Type: **PLASTINPUTINFO**

A pointer to a LASTINPUTINFO structure that receives the time of the last input event.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

This function is useful for input idle detection. However, **GetLastInputInfo** does not provide system-wide user input information across all running sessions. Rather, **GetLastInputInfo** provides session-specific user input information for only the session that invoked the function.

The tick count when the last input event was received (see LASTINPUTINFO) is not guaranteed to be incremental. In some cases, the value might be less than the tick count of a prior event. For example, this can be caused by a timing gap between the raw input thread and the desktop thread or an event raised by SendInput, which supplies its own tick count.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

Library	User32.lib
DLL	User32.dll

See also

Conceptual

Keyboard Input

LASTINPUTINFO

Reference

GetMouseMovePointsEx function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves a history of up to 64 previous coordinates of the mouse or pen.

Syntax

```
int GetMouseMovePointsEx(
  [in]  UINT             cbSize,
  [in]  LPMOUSEMOVEPOINT lppt,
  [out] LPMOUSEMOVEPOINT lpptBuf,
  [in]  int              nBufPoints,
  [in]  DWORD            resolution
);
```

Parameters

`[in] cbSize`

Type: **UINT**

The size, in bytes, of the MOUSEMOVEPOINT structure.

`[in] lppt`

Type: **LPMOUSEMOVEPOINT**

A pointer to a MOUSEMOVEPOINT structure containing valid mouse coordinates (in screen coordinates). It may also contain a time stamp.

The **GetMouseMovePointsEx** function searches for the point in the mouse coordinates history. If the function finds the point, it returns the last *nBufPoints* prior to and including the supplied point.

If your application supplies a time stamp, the **GetMouseMovePointsEx** function will use it to differentiate between two equal points that were recorded at different times.

An application should call this function using the mouse coordinates received from the WM_MOUSEMOVE message and convert them to screen coordinates.

`[out] lpptBuf`

Type: **LPMOUSEMOVEPOINT**

A pointer to a buffer that will receive the points. It should be at least *cbSize** *nBufPoints* in size.

`[in] nBufPoints`

Type: **int**

The number of points to be retrieved.

`[in] resolution`

Type: **DWORD**

The resolution desired. This parameter can be one of the following values.

VALUE	MEANING
GMMP_USE_DISPLAY_POINTS 1	Retrieves the points using the display resolution.
GMMP_USE_HIGH_RESOLUTION_POINTS 2	Retrieves high resolution points. Points can range from zero to 65,535 (0xFFFF) in both x- and y-coordinates. This is the resolution provided by absolute coordinate pointing devices such as drawing tablets.

Return value

Type: **int**

If the function succeeds, the return value is the number of points in the buffer. Otherwise, the function returns –1. For extended error information, your application can call GetLastError.

Remarks

The system retains the last 64 mouse coordinates and their time stamps. If your application supplies a mouse coordinate to **GetMouseMovePointsEx** and the coordinate exists in the system's mouse coordinate history, the function retrieves the specified number of coordinates from the systems' history. You can also supply a time stamp, which will be used to differentiate between identical points in the history.

The **GetMouseMovePointsEx** function will return points that eventually were dispatched not only to the calling thread but also to other threads.

GetMouseMovePointsEx may fail or return erroneous values in the following cases:

- If negative coordinates are passed in the MOUSEMOVEPOINT structure.
- If **GetMouseMovePointsEx** retrieves a coordinate with a negative value.

These situations can occur if multiple monitors are present. To correct this, first call GetSystemMetrics to get the following values:

- SM_XVIRTUALSCREEN,
- SM_YVIRTUALSCREEN,
- SM_CXVIRTUALSCREEN, and
- SM_CYVIRTUALSCREEN.

Then, for each point that is returned from **GetMouseMovePointsEx**, perform the following transform:

```
int nVirtualWidth = GetSystemMetrics(SM_CXVIRTUALSCREEN) ;
int nVirtualHeight = GetSystemMetrics(SM_CYVIRTUALSCREEN) ;
int nVirtualLeft = GetSystemMetrics(SM_XVIRTUALSCREEN) ;
int nVirtualTop = GetSystemMetrics(SM_YVIRTUALSCREEN) ;
int cpt = 0 ;
int mode = GMMP_USE_DISPLAY_POINTS ;

MOUSEMOVEPOINT mp_in ;
MOUSEMOVEPOINT mp_out[64] ;

ZeroMemory(&mp_in, sizeof(mp_in)) ;
mp_in.x = pt.x & 0x0000FFFF ;//Ensure that this number will pass through.
mp_in.y = pt.y & 0x0000FFFF ;
cpt = GetMouseMovePointsEx(&mp_in, &mp_out, 64, mode) ;

for (int i = 0; i < cpt; i++)
{
   switch(mode)
   {
   case GMMP_USE_DISPLAY_POINTS:
      if (mp_out[i].x > 32767)
         mp_out[i].x -= 65536 ;
      if (mp_out[i].y > 32767)
         mp_out[i].y -= 65536 ;
      break ;
   case GMMP_USE_HIGH_RESOLUTION_POINTS:
      mp_out[i].x = ((mp_out[i].x * (nVirtualWidth - 1)) - (nVirtualLeft * 65536)) / nVirtualWidth ;
      mp_out[i].y = ((mp_out[i].y * (nVirtualHeight - 1)) - (nVirtualTop * 65536)) / nVirtualHeight ;
      break ;
   }
}
```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

MOUSEMOVEPOINT

Mouse Input

Reference

GetRawInputBuffer function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Performs a buffered read of the raw input messages data found in the calling thread's message queue.

Syntax

```
UINT GetRawInputBuffer(
  [out, optional] PRAWINPUT pData,
  [in, out]       PUINT     pcbSize,
  [in]            UINT      cbSizeHeader
);
```

Parameters

`[out, optional] pData`

Type: **PRAWINPUT**

A pointer to a buffer of RAWINPUT structures that contain the raw input data. Buffer should be aligned on a pointer boundary, which is a **DWORD** on 32-bit architectures and a **QWORD** on 64-bit architectures.

If **NULL**, size of the first raw input message data (minimum required buffer), in bytes, is returned in **pcbSize*.

`[in, out] pcbSize`

Type: **PUINT**

The size, in bytes, of the provided RAWINPUT buffer.

`[in] cbSizeHeader`

Type: **UINT**

The size, in bytes, of the RAWINPUTHEADER structure.

Return value

Type: **UINT**

If *pData* is **NULL** and the function is successful, the return value is zero. If *pData* is not **NULL** and the function is successful, the return value is the number of RAWINPUT structures written to *pData*.

If an error occurs, the return value is (**UINT**)-1. Call GetLastError for the error code.

Remarks

When an application receives raw input, its message queue gets a WM_INPUT message and the queue status flag QS_RAWINPUT is set.

Using **GetRawInputBuffer**, the raw input data is read in the array of variable size RAWINPUT structures and corresponding WM_INPUT messages are removed from the calling thread's message queue. You can call this method several times with buffer that cannot fit all message's data until all raw input messages have been read.

The NEXTRAWINPUTBLOCK macro allows an application to traverse an array of RAWINPUT structures.

If all raw input messages have been successfully read from message queue then QS_RAWINPUT flag is cleared from the calling thread's message queue status.

NOTE

WOW64: To get the correct size of the raw input buffer, do not use **pcbSize*, use **pcbSize* * 8 instead. To ensure **GetRawInputBuffer** behaves properly on WOW64, you must align the RAWINPUT structure by 8 bytes. The following code shows how to align **RAWINPUT** for WOW64.

```
[StructLayout(LayoutKind.Explicit)]
internal struct RAWINPUT
{
    [FieldOffset(0)]
    public RAWINPUTHEADER header;

    [FieldOffset(16+8)]
    public RAWMOUSE mouse;

    [FieldOffset(16+8)]
    public RAWKEYBOARD keyboard;

    [FieldOffset(16+8)]
    public RAWHID hid;
}
```

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetMessage

NEXTRAWINPUTBLOCK

RAWINPUT

RAWINPUTHEADER

Raw Input

Reference

About Raw Input

GetRawInputData function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the raw input from the specified device.

Syntax

```
UINT GetRawInputData(
  [in]            HRAWINPUT hRawInput,
  [in]            UINT      uiCommand,
  [out, optional] LPVOID    pData,
  [in, out]       PUINT     pcbSize,
  [in]            UINT      cbSizeHeader
);
```

Parameters

`[in] hRawInput`

Type: **HRAWINPUT**

A handle to the RAWINPUT structure. This comes from the *lParam* in WM_INPUT.

`[in] uiCommand`

Type: **UINT**

The command flag. This parameter can be one of the following values.

VALUE	MEANING
RID_HEADER 0x10000005	Get the header information from the RAWINPUT structure.
RID_INPUT 0x10000003	Get the raw data from the RAWINPUT structure.

`[out, optional] pData`

Type: **LPVOID**

A pointer to the data that comes from the RAWINPUT structure. This depends on the value of *uiCommand*. If *pData* is **NULL**, the required size of the buffer is returned in **pcbSize*.

`[in, out] pcbSize`

Type: **PUINT**

The size, in bytes, of the data in *pData*.

`[in] cbSizeHeader`

Type: **UINT**

The size, in bytes, of the RAWINPUTHEADER structure.

Return value

Type: **UINT**

If *pData* is **NULL** and the function is successful, the return value is 0. If *pData* is not **NULL** and the function is successful, the return value is the number of bytes copied into pData.

If there is an error, the return value is (**UINT**)-1.

Remarks

GetRawInputData gets the raw input one RAWINPUT structure at a time. In contrast, GetRawInputBuffer gets an array of **RAWINPUT** structures.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-rawinput-l1-1-0 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

GetRawInputBuffer

RAWINPUT

RAWINPUTHEADER

Raw Input

Reference

GetRawInputDeviceInfoA function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves information about the raw input device.

Syntax

```
UINT GetRawInputDeviceInfoA(
  [in, optional]      HANDLE hDevice,
  [in]                UINT   uiCommand,
  [in, out, optional] LPVOID pData,
  [in, out]           PUINT  pcbSize
);
```

Parameters

`[in, optional] hDevice`

Type: **HANDLE**

A handle to the raw input device. This comes from the **hDevice** member of RAWINPUTHEADER or from GetRawInputDeviceList.

`[in] uiCommand`

Type: **UINT**

Specifies what data will be returned in *pData*. This parameter can be one of the following values.

VALUE	MEANING
RIDI_PREPARSEDDATA 0x20000005	*pData* is a PHIDP_PREPARSED_DATA pointer to a buffer for a top-level collection's preparsed data.
RIDI_DEVICENAME 0x20000007	*pData* points to a string that contains the device interface name. If this device is opened with Shared Access Mode then you can call CreateFile with this name to open a HID collection and use returned handle for calling ReadFile to read input reports and WriteFile to send output reports. For more information, see Opening HID Collections and Handling HID Reports. For this *uiCommand* only, the value in *pcbSize* is the character count (not the byte count).
RIDI_DEVICEINFO 0x2000000b	*pData* points to an RID_DEVICE_INFO structure.

`[in, out, optional] pData`

Type: **LPVOID**

A pointer to a buffer that contains the information specified by *uiCommand*.

If *uiCommand* is **RIDI_DEVICEINFO**, set the **cbSize** member of RID_DEVICE_INFO to `sizeof(RID_DEVICE_INFO)` before calling **GetRawInputDeviceInfo**.

`[in, out] pcbSize`

Type: **PUINT**

The size, in bytes, of the data in *pData*.

Return value

Type: **UINT**

If successful, this function returns a non-negative number indicating the number of bytes copied to *pData*.

If *pData* is not large enough for the data, the function returns -1. If *pData* is **NULL**, the function returns a value of zero. In both of these cases, *pcbSize* is set to the minimum size required for the *pData* buffer.

Call GetLastError to identify any other errors.

Remarks

NOTE

The winuser.h header defines GetRawInputDeviceInfo as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-rawinput-l1-1-0 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

RAWINPUTHEADER

RID_DEVICE_INFO

Raw Input

Reference

WM_INPUT

Top-Level Collections

Preparsed Data

PHIDP_PREPARSED_DATA

Opening HID collections

Handling HID Reports

GetRawInputDeviceInfoW function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves information about the raw input device.

Syntax

```
UINT GetRawInputDeviceInfoW(
  [in, optional]      HANDLE hDevice,
  [in]                UINT   uiCommand,
  [in, out, optional] LPVOID pData,
  [in, out]           PUINT  pcbSize
);
```

Parameters

`[in, optional] hDevice`

Type: **HANDLE**

A handle to the raw input device. This comes from the **hDevice** member of RAWINPUTHEADER or from GetRawInputDeviceList.

`[in] uiCommand`

Type: **UINT**

Specifies what data will be returned in *pData*. This parameter can be one of the following values.

VALUE	MEANING
RIDI_PREPARSEDDATA 0x20000005	*pData* is a PHIDP_PREPARSED_DATA pointer to a buffer for a top-level collection's preparsed data.
RIDI_DEVICENAME 0x20000007	*pData* points to a string that contains the device interface name. If this device is opened with Shared Access Mode then you can call CreateFile with this name to open a HID collection and use returned handle for calling ReadFile to read input reports and WriteFile to send output reports. For more information, see Opening HID Collections and Handling HID Reports. For this *uiCommand* only, the value in *pcbSize* is the character count (not the byte count).
RIDI_DEVICEINFO 0x2000000b	*pData* points to an RID_DEVICE_INFO structure.

`[in, out, optional] pData`

Type: **LPVOID**

A pointer to a buffer that contains the information specified by *uiCommand*.

If *uiCommand* is **RIDI_DEVICEINFO**, set the **cbSize** member of RID_DEVICE_INFO to `sizeof(RID_DEVICE_INFO)` before calling **GetRawInputDeviceInfo**.

`[in, out] pcbSize`

Type: **PUINT**

The size, in bytes, of the data in *pData*.

Return value

Type: **UINT**

If successful, this function returns a non-negative number indicating the number of bytes copied to *pData*.

If *pData* is not large enough for the data, the function returns -1. If *pData* is **NULL**, the function returns a value of zero. In both of these cases, *pcbSize* is set to the minimum size required for the *pData* buffer.

Call GetLastError to identify any other errors.

Remarks

NOTE

The winuser.h header defines GetRawInputDeviceInfo as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-rawinput-l1-1-0 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

RAWINPUTHEADER

RID_DEVICE_INFO

Raw Input

Reference

WM_INPUT

Top-Level Collections

Preparsed Data

PHIDP_PREPARSED_DATA

Opening HID collections

Handling HID Reports

GetRawInputDeviceList function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Enumerates the raw input devices attached to the system.

Syntax

```
UINT GetRawInputDeviceList(
  [out, optional] PRAWINPUTDEVICELIST pRawInputDeviceList,
  [in, out]       PUINT               puiNumDevices,
  [in]            UINT                cbSize
);
```

Parameters

`[out, optional] pRawInputDeviceList`

Type: **PRAWINPUTDEVICELIST**

An array of RAWINPUTDEVICELIST structures for the devices attached to the system. If **NULL**, the number of devices are returned in **puiNumDevices*.

`[in, out] puiNumDevices`

Type: **PUINT**

If *pRawInputDeviceList* is **NULL**, the function populates this variable with the number of devices attached to the system; otherwise, this variable specifies the number of RAWINPUTDEVICELIST structures that can be contained in the buffer to which *pRawInputDeviceList* points. If this value is less than the number of devices attached to the system, the function returns the actual number of devices in this variable and fails with **ERROR_INSUFFICIENT_BUFFER**. If this value is greater than or equal to the number of devices attached to the system, then the value is unchanged, and the number of devices is reported as the return value.

`[in] cbSize`

Type: **UINT**

The size of a RAWINPUTDEVICELIST structure, in bytes.

Return value

Type: **UINT**

If the function is successful, the return value is the number of devices stored in the buffer pointed to by *pRawInputDeviceList*.

On any other error, the function returns (**UINT**) -1 and GetLastError returns the error indication.

Remarks

The devices returned from this function are the mouse, the keyboard, and other Human Interface Device (HID) devices.

To get more detailed information about the attached devices, call GetRawInputDeviceInfo using the hDevice from RAWINPUTDEVICELIST.

Examples

The following sample code shows a typical call to **GetRawInputDeviceList**:

```
UINT nDevices;
PRAWINPUTDEVICELIST pRawInputDeviceList = NULL;
while (true) {
    if (GetRawInputDeviceList(NULL, &nDevices, sizeof(RAWINPUTDEVICELIST)) != 0) { Error();}
    if (nDevices == 0) { break; }
    if ((pRawInputDeviceList = malloc(sizeof(RAWINPUTDEVICELIST) * nDevices)) == NULL) {Error();}
    nDevices = GetRawInputDeviceList(pRawInputDeviceList, &nDevices, sizeof(RAWINPUTDEVICELIST));
    if (nDevices == (UINT)-1) {
        if (GetLastError() != ERROR_INSUFFICIENT_BUFFER) { Error(); }
        // Devices were added.
        free(pRawInputDeviceList);
        continue;
    }
    break;
}
// do the job...
// after the job, free the RAWINPUTDEVICELIST
free(pRawInputDeviceList);
```

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-rawinput-l1-1-0 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

GetRawInputDeviceInfo

RAWINPUTDEVICELIST

Raw Input

Reference

GetRegisteredRawInputDevices function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the information about the raw input devices for the current application.

Syntax

```
UINT GetRegisteredRawInputDevices(
  [out, optional] PRAWINPUTDEVICE pRawInputDevices,
  [in, out]       PUINT           puiNumDevices,
  [in]            UINT            cbSize
);
```

Parameters

`[out, optional] pRawInputDevices`

Type: **PRAWINPUTDEVICE**

An array of RAWINPUTDEVICE structures for the application.

`[in, out] puiNumDevices`

Type: **PUINT**

The number of RAWINPUTDEVICE structures in **pRawInputDevices*.

`[in] cbSize`

Type: **UINT**

The size, in bytes, of a RAWINPUTDEVICE structure.

Return value

Type: **UINT**

If successful, the function returns a non-negative number that is the number of RAWINPUTDEVICE structures written to the buffer.

If the *pRawInputDevices* buffer is too small or **NULL**, the function sets the last error as **ERROR_INSUFFICIENT_BUFFER**, returns -1, and sets *puiNumDevices* to the required number of devices. If the function fails for any other reason, it returns -1. For more details, call GetLastError.

Remarks

To receive raw input from a device, an application must register it by using RegisterRawInputDevices.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

RAWINPUTDEVICE

Raw Input

Reference

RegisterRawInputDevices

HARDWAREINPUT structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Contains information about a simulated message generated by an input device other than a keyboard or mouse.

Syntax

```
typedef struct tagHARDWAREINPUT {
  DWORD uMsg;
  WORD  wParamL;
  WORD  wParamH;
} HARDWAREINPUT, *PHARDWAREINPUT, *LPHARDWAREINPUT;
```

Members

`uMsg`

Type: **DWORD**

The message generated by the input hardware.

`wParamL`

Type: **WORD**

The low-order word of the *lParam* parameter for **uMsg**.

`wParamH`

Type: **WORD**

The high-order word of the *lParam* parameter for **uMsg**.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

INPUT

Keyboard Input

Reference

SendInput

INPUT structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Used by SendInput to store information for synthesizing input events such as keystrokes, mouse movement, and mouse clicks.

Syntax

```
typedef struct tagINPUT {
  DWORD type;
  union {
    MOUSEINPUT    mi;
    KEYBDINPUT    ki;
    HARDWAREINPUT hi;
  } DUMMYUNIONNAME;
} INPUT, *PINPUT, *LPINPUT;
```

Members

`type`

Type: **DWORD**

The type of the input event. This member can be one of the following values.

VALUE	MEANING
INPUT_MOUSE 0	The event is a mouse event. Use the **mi** structure of the union.
INPUT_KEYBOARD 1	The event is a keyboard event. Use the **ki** structure of the union.
INPUT_HARDWARE 2	The event is a hardware event. Use the **hi** structure of the union.

`DUMMYUNIONNAME`

`DUMMYUNIONNAME.mi`

Type: **MOUSEINPUT**

The information about a simulated mouse event.

`DUMMYUNIONNAME.ki`

Type: **KEYBDINPUT**

The information about a simulated keyboard event.

`DUMMYUNIONNAME.hi`

Type: **HARDWAREINPUT**

The information about a simulated hardware event.

Remarks

INPUT_KEYBOARD supports nonkeyboard input methods, such as handwriting recognition or voice recognition, as if it were text input by using the **KEYEVENTF_UNICODE** flag. For more information, see the remarks section of KEYBDINPUT.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetMessageExtraInfo

HARDWAREINPUT

KEYBDINPUT

Keyboard Input

MOUSEINPUT

Reference

SendInput

keybd_event

mouse_event

IsWindowEnabled function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Determines whether the specified window is enabled for mouse and keyboard input.

Syntax

```
BOOL IsWindowEnabled(
  [in] HWND hWnd
);
```

Parameters

`[in] hWnd`

Type: **HWND**

A handle to the window to be tested.

Return value

Type: **BOOL**

If the window is enabled, the return value is nonzero.

If the window is not enabled, the return value is zero.

Remarks

A child window receives input only if it is both enabled and visible.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-window-l1-1-4 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

EnableWindow

IsWindowVisible

Keyboard Input

Reference

keybd_event function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Synthesizes a keystroke. The system can use such a synthesized keystroke to generate a WM_KEYUP or WM_KEYDOWN message. The keyboard driver's interrupt handler calls the **keybd_event** function.

> **Note** This function has been superseded. Use SendInput instead.

Syntax

```
void keybd_event(
  [in] BYTE      bVk,
  [in] BYTE      bScan,
  [in] DWORD     dwFlags,
  [in] ULONG_PTR dwExtraInfo
);
```

Parameters

`[in] bVk`

Type: **BYTE**

A virtual-key code. The code must be a value in the range 1 to 254. For a complete list, see Virtual Key Codes.

`[in] bScan`

Type: **BYTE**

A hardware scan code for the key.

`[in] dwFlags`

Type: **DWORD**

Controls various aspects of function operation. This parameter can be one or more of the following values.

VALUE	MEANING
KEYEVENTF_EXTENDEDKEY 0x0001	If specified, the scan code was preceded by a prefix byte having the value 0xE0 (224).
KEYEVENTF_KEYUP 0x0002	If specified, the key is being released. If not specified, the key is being depressed.

`[in] dwExtraInfo`

Type: **ULONG_PTR**

An additional value associated with the key stroke.

Return value

None

Remarks

An application can simulate a press of the PRINTSCRN key in order to obtain a screen snapshot and save it to the clipboard. To do this, call **keybd_event** with the *bVk* parameter set to **VK_SNAPSHOT**.

Examples

The following sample program toggles the NUM LOCK light by using **keybd_event** with a virtual key of **VK_NUMLOCK**. It takes a Boolean value that indicates whether the light should be turned off (**FALSE**) or on (**TRUE**). The same technique can be used for the CAPS LOCK key (**VK_CAPITAL**) and the SCROLL LOCK key (**VK_SCROLL**).

```
#include <windows.h>

void SetNumLock( BOOL bState )
{
   BYTE keyState[256];

   GetKeyboardState((LPBYTE)&keyState);
   if( (bState && !(keyState[VK_NUMLOCK] & 1)) ||
       (!bState && (keyState[VK_NUMLOCK] & 1)) )
   {
   // Simulate a key press
      keybd_event( VK_NUMLOCK,
                   0x45,
                   KEYEVENTF_EXTENDEDKEY | 0,
                   0 );

   // Simulate a key release
      keybd_event( VK_NUMLOCK,
                   0x45,
                   KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP,
                   0);
   }
}

void main()
{
   SetNumLock( TRUE );
}
```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

GetKeyState

Keyboard Input

MapVirtualKey

Reference

SetKeyboardState

keybd_event

KEYBDINPUT structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Contains information about a simulated keyboard event.

Syntax

```
typedef struct tagKEYBDINPUT {
  WORD      wVk;
  WORD      wScan;
  DWORD     dwFlags;
  DWORD     time;
  ULONG_PTR dwExtraInfo;
} KEYBDINPUT, *PKEYBDINPUT, *LPKEYBDINPUT;
```

Members

`wVk`

Type: **WORD**

A virtual-key code. The code must be a value in the range 1 to 254. If the **dwFlags** member specifies **KEYEVENTF_UNICODE**, **wVk** must be 0.

`wScan`

Type: **WORD**

A hardware scan code for the key. If **dwFlags** specifies **KEYEVENTF_UNICODE**, **wScan** specifies a Unicode character which is to be sent to the foreground application.

`dwFlags`

Type: **DWORD**

Specifies various aspects of a keystroke. This member can be certain combinations of the following values.

VALUE	MEANING
KEYEVENTF_EXTENDEDKEY 0x0001	If specified, the scan code was preceded by a prefix byte that has the value 0xE0 (224).
KEYEVENTF_KEYUP 0x0002	If specified, the key is being released. If not specified, the key is being pressed.
KEYEVENTF_SCANCODE 0x0008	If specified, **wScan** identifies the key and **wVk** is ignored.

KEYEVENTF_UNICODE 0x0004	If specified, the system synthesizes a **VK_PACKET** keystroke. The **wVk** parameter must be zero. This flag can only be combined with the **KEYEVENTF_KEYUP** flag. For more information, see the Remarks section.

`time`

Type: **DWORD**

The time stamp for the event, in milliseconds. If this parameter is zero, the system will provide its own time stamp.

`dwExtraInfo`

Type: **ULONG_PTR**

An additional value associated with the keystroke. Use the GetMessageExtraInfo function to obtain this information.

Remarks

INPUT_KEYBOARD supports nonkeyboard-input methods—such as handwriting recognition or voice recognition—as if it were text input by using the **KEYEVENTF_UNICODE** flag. If **KEYEVENTF_UNICODE** is specified, SendInput sends a WM_KEYDOWN or WM_KEYUP message to the foreground thread's message queue with *wParam* equal to **VK_PACKET**. Once GetMessage or PeekMessage obtains this message, passing the message to TranslateMessage posts a WM_CHAR message with the Unicode character originally specified by **wScan**. This Unicode character will automatically be converted to the appropriate ANSI value if it is posted to an ANSI window.

Set the **KEYEVENTF_SCANCODE** flag to define keyboard input in terms of the scan code. This is useful to simulate a physical keystroke regardless of which keyboard is currently being used. The virtual key value of a key may alter depending on the current keyboard layout or what other keys were pressed, but the scan code will always be the same.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetMessageExtraInfo

INPUT

Keyboard Input

Reference

SendInput

LASTINPUTINFO structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Contains the time of the last input.

Syntax

```
typedef struct tagLASTINPUTINFO {
  UINT  cbSize;
  DWORD dwTime;
} LASTINPUTINFO, *PLASTINPUTINFO;
```

Members

`cbSize`

Type: **UINT**

The size of the structure, in bytes. This member must be set to `sizeof(LASTINPUTINFO)`.

`dwTime`

Type: **DWORD**

The tick count when the last input event was received.

Remarks

This function is useful for input idle detection. For more information on tick counts, see GetTickCount.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetLastInputInfo

GetTickCount

Keyboard Input

Reference

LoadKeyboardLayoutA function (winuser.h)

7/6/2022 • 4 minutes to read • Edit Online

Loads a new input locale identifier (formerly called the keyboard layout) into the system.

Prior to Windows 8: Several input locale identifiers can be loaded at a time, but only one per process is active at a time. Loading multiple input locale identifiers makes it possible to rapidly switch between them.

Beginning in Windows 8: The input locale identifier is loaded for the entire system. This function has no effect if the current process does not own the window with keyboard focus.

Syntax

```
HKL LoadKeyboardLayoutA(
  [in] LPCSTR pwszKLID,
  [in] UINT   Flags
);
```

Parameters

`[in] pwszKLID`

Type: **LPCTSTR**

The name of the input locale identifier to load. This name is a string composed of the hexadecimal value of the Language Identifier (low word) and a device identifier (high word). For example, U.S. English has a language identifier of 0x0409, so the primary U.S. English layout is named "00000409". Variants of U.S. English layout (such as the Dvorak layout) are named "00010409", "00020409", and so on.

`[in] Flags`

Type: **UINT**

Specifies how the input locale identifier is to be loaded. This parameter can be one or more of the following values.

VALUE	MEANING
KLF_ACTIVATE 0x00000001	**Prior to Windows 8:** If the specified input locale identifier is not already loaded, the function loads and activates the input locale identifier for the current thread. **Beginning in Windows 8:** If the specified input locale identifier is not already loaded, the function loads and activates the input locale identifier for the system.

KLF_NOTELLSHELL 0x00000080	**Prior to Windows 8:** Prevents a ShellProc hook procedure from receiving an **HSHELL_LANGUAGE** hook code when the new input locale identifier is loaded. This value is typically used when an application loads multiple input locale identifiers one after another. Applying this value to all but the last input locale identifier delays the shell's processing until all input locale identifiers have been added. **Beginning in Windows 8:** In this scenario, the last input locale identifier is set for the entire system.
KLF_REORDER 0x00000008	**Prior to Windows 8:** Moves the specified input locale identifier to the head of the input locale identifier list, making that locale identifier the active locale identifier for the current thread. This value reorders the input locale identifier list even if **KLF_ACTIVATE** is not provided. **Beginning in Windows 8:** Moves the specified input locale identifier to the head of the input locale identifier list, making that locale identifier the active locale identifier for the system. This value reorders the input locale identifier list even if **KLF_ACTIVATE** is not provided.
KLF_REPLACELANG 0x00000010	If the new input locale identifier has the same language identifier as a current input locale identifier, the new input locale identifier replaces the current one as the input locale identifier for that language. If this value is not provided and the input locale identifiers have the same language identifiers, the current input locale identifier is not replaced and the function returns **NULL**.
KLF_SUBSTITUTE_OK 0x00000002	Substitutes the specified input locale identifier with another locale preferred by the user. The system starts with this flag set, and it is recommended that your application always use this flag. The substitution occurs only if the registry key **HKEY_CURRENT_USER\Keyboard\Layout\Substitutes** explicitly defines a substitution locale. For example, if the key includes the value name "00000409" with value "00010409", loading the U.S. English layout ("00000409") causes the Dvorak U.S. English layout ("00010409") to be loaded instead. The system uses **KLF_SUBSTITUTE_OK** when booting, and it is recommended that all applications use this value when loading input locale identifiers to ensure that the user's preference is selected.
KLF_SETFORPROCESS 0x00000100	**Prior to Windows 8:** This flag is valid only with **KLF_ACTIVATE**. Activates the specified input locale identifier for the entire process and sends the WM_INPUTLANGCHANGE message to the current thread's Focus or Active window. Typically, **LoadKeyboardLayout** activates an input locale identifier only for the current thread. **Beginning in Windows 8:** This flag is not used. **LoadKeyboardLayout** always activates an input locale identifier for the entire system if the current process owns the window with keyboard focus.
KLF_UNLOADPREVIOUS	This flag is unsupported. Use the UnloadKeyboardLayout function instead.

Return value

Type: **HKL**

If the function succeeds, the return value is the input locale identifier corresponding to the name specified in *pwszKLID*. If no matching locale is available, the return value is the default language of the system.

If the function fails, the return value is NULL. This can occur if the layout library is loaded from the application directory.

To get extended error information, call GetLastError.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

An application can and will typically load the default input locale identifier or IME for a language and can do so by specifying only a string version of the language identifier. If an application wants to load a specific locale or IME, it should read the registry to determine the specific input locale identifier to pass to **LoadKeyboardLayout**. In this case, a request to activate the default input locale identifier for a locale will activate the first matching one. A specific IME should be activated using an explicit input locale identifier returned from GetKeyboardLayout or **LoadKeyboardLayout**.

Prior to Windows 8: This function only affects the layout for the current process or thread.

Beginning in Windows 8: This function affects the layout for the entire system.

NOTE

The winuser.h header defines LoadKeyboardLayout as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

ActivateKeyboardLayout

Conceptual

GetKeyboardLayoutName

Keyboard Input

MAKELANGID

Other Resources

Reference

UnloadKeyboardLayout

LoadKeyboardLayoutW function (winuser.h)

7/6/2022 • 4 minutes to read • Edit Online

Loads a new input locale identifier (formerly called the keyboard layout) into the system.

Prior to Windows 8: Several input locale identifiers can be loaded at a time, but only one per process is active at a time. Loading multiple input locale identifiers makes it possible to rapidly switch between them.

Beginning in Windows 8: The input locale identifier is loaded for the entire system. This function has no effect if the current process does not own the window with keyboard focus.

Syntax

```
HKL LoadKeyboardLayoutW(
  [in] LPCWSTR pwszKLID,
  [in] UINT    Flags
);
```

Parameters

`[in] pwszKLID`

Type: **LPCTSTR**

The name of the input locale identifier to load. This name is a string composed of the hexadecimal value of the Language Identifier (low word) and a device identifier (high word). For example, U.S. English has a language identifier of 0x0409, so the primary U.S. English layout is named "00000409". Variants of U.S. English layout (such as the Dvorak layout) are named "00010409", "00020409", and so on.

`[in] Flags`

Type: **UINT**

Specifies how the input locale identifier is to be loaded. This parameter can be one of the following values.

VALUE	MEANING
KLF_ACTIVATE 0x00000001	**Prior to Windows 8:** If the specified input locale identifier is not already loaded, the function loads and activates the input locale identifier for the current thread. **Beginning in Windows 8:** If the specified input locale identifier is not already loaded, the function loads and activates the input locale identifier for the system.

KLF_NOTELLSHELL 0x00000080	**Prior to Windows 8:** Prevents a ShellProc hook procedure from receiving an **HSHELL_LANGUAGE** hook code when the new input locale identifier is loaded. This value is typically used when an application loads multiple input locale identifiers one after another. Applying this value to all but the last input locale identifier delays the shell's processing until all input locale identifiers have been added. **Beginning in Windows 8:** In this scenario, the last input locale identifier is set for the entire system.
KLF_REORDER 0x00000008	**Prior to Windows 8:** Moves the specified input locale identifier to the head of the input locale identifier list, making that locale identifier the active locale identifier for the current thread. This value reorders the input locale identifier list even if **KLF_ACTIVATE** is not provided. **Beginning in Windows 8:** Moves the specified input locale identifier to the head of the input locale identifier list, making that locale identifier the active locale identifier for the system. This value reorders the input locale identifier list even if **KLF_ACTIVATE** is not provided.
KLF_REPLACELANG 0x00000010	If the new input locale identifier has the same language identifier as a current input locale identifier, the new input locale identifier replaces the current one as the input locale identifier for that language. If this value is not provided and the input locale identifiers have the same language identifiers, the current input locale identifier is not replaced and the function returns **NULL**.
KLF_SUBSTITUTE_OK 0x00000002	Substitutes the specified input locale identifier with another locale preferred by the user. The system starts with this flag set, and it is recommended that your application always use this flag. The substitution occurs only if the registry key **HKEY_CURRENT_USER\Keyboard\Layout\Substitutes** explicitly defines a substitution locale. For example, if the key includes the value name "00000409" with value "00010409", loading the U.S. English layout ("00000409") causes the Dvorak U.S. English layout ("00010409") to be loaded instead. The system uses **KLF_SUBSTITUTE_OK** when booting, and it is recommended that all applications use this value when loading input locale identifiers to ensure that the user's preference is selected.
KLF_SETFORPROCESS 0x00000100	**Prior to Windows 8:** This flag is valid only with **KLF_ACTIVATE**. Activates the specified input locale identifier for the entire process and sends the WM_INPUTLANGCHANGE message to the current thread's Focus or Active window. Typically, **LoadKeyboardLayout** activates an input locale identifier only for the current thread. **Beginning in Windows 8:** This flag is not used. **LoadKeyboardLayout** always activates an input locale identifier for the entire system if the current process owns the window with keyboard focus.
KLF_UNLOADPREVIOUS	This flag is unsupported. Use the UnloadKeyboardLayout function instead.

Return value

Type: **HKL**

If the function succeeds, the return value is the input locale identifier corresponding to the name specified in *pwszKLID*. If no matching locale is available, the return value is the default language of the system.

If the function fails, the return value is NULL. This can occur if the layout library is loaded from the application directory.

To get extended error information, call GetLastError.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

An application can and will typically load the default input locale identifier or IME for a language and can do so by specifying only a string version of the language identifier. If an application wants to load a specific locale or IME, it should read the registry to determine the specific input locale identifier to pass to **LoadKeyboardLayout**. In this case, a request to activate the default input locale identifier for a locale will activate the first matching one. A specific IME should be activated using an explicit input locale identifier returned from GetKeyboardLayout or **LoadKeyboardLayout**.

Prior to Windows 8: This function only affects the layout for the current process or thread.

Beginning in Windows 8: This function affects the layout for the entire system.

NOTE

The winuser.h header defines LoadKeyboardLayout as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

ActivateKeyboardLayout

Conceptual

GetKeyboardLayoutName

Keyboard Input

MAKELANGID

Other Resources

Reference

UnloadKeyboardLayout

MapVirtualKeyA function (winuser.h)

7/6/2022 • 3 minutes to read • Edit Online

Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code.

To specify a handle to the keyboard layout to use for translating the specified code, use the MapVirtualKeyEx function.

Syntax

```
UINT MapVirtualKeyA(
  [in] UINT uCode,
  [in] UINT uMapType
);
```

Parameters

`[in] uCode`

Type: **UINT**

The virtual key code or scan code for a key. How this value is interpreted depends on the value of the *uMapType* parameter.

Starting with Windows Vista, the high byte of the *uCode* value can contain either 0xe0 or 0xe1 to specify the extended scan code.

`[in] uMapType`

Type: **UINT**

The translation to be performed. The value of this parameter depends on the value of the *uCode* parameter.

VALUE	MEANING
MAPVK_VK_TO_VSC 0	The *uCode* parameter is a virtual-key code and is translated into a scan code. If it is a virtual-key code that does not distinguish between left- and right-hand keys, the left-hand scan code is returned. If there is no translation, the function returns 0.
MAPVK_VSC_TO_VK 1	The *uCode* parameter is a scan code and is translated into a virtual-key code that does not distinguish between left- and right-hand keys. If there is no translation, the function returns 0.
MAPVK_VK_TO_CHAR 2	The *uCode* parameter is a virtual-key code and is translated into an unshifted character value in the low order word of the return value. Dead keys (diacritics) are indicated by setting the top bit of the return value. If there is no translation, the function returns 0.

VALUE	MEANING
MAPVK_VSC_TO_VK_EX 3	The *uCode* parameter is a scan code and is translated into a virtual-key code that distinguishes between left- and right-hand keys. If there is no translation, the function returns 0.
MAPVK_VK_TO_VSC_EX 4	**Windows Vista and later:** The *uCode* parameter is a virtual-key code and is translated into a scan code. If it is a virtual-key code that does not distinguish between left- and right-hand keys, the left-hand scan code is returned. If the scan code is an extended scan code, the high byte of the *uCode* value can contain either 0xe0 or 0xe1 to specify the extended scan code. If there is no translation, the function returns 0.

Return value

Type: **UINT**

The return value is either a scan code, a virtual-key code, or a character value, depending on the value of *uCode* and *uMapType*. If there is no translation, the return value is zero.

Remarks

An application can use **MapVirtualKey** to translate scan codes to the virtual-key code constants **VK_SHIFT**, **VK_CONTROL**, and **VK_MENU**, and vice versa. These translations do not distinguish between the left and right instances of the SHIFT, CTRL, or ALT keys.

An application can get the scan code corresponding to the left or right instance of one of these keys by calling **MapVirtualKey** with *uCode* set to one of the following virtual-key code constants:

- **VK_LSHIFT**
- **VK_RSHIFT**
- **VK_LCONTROL**
- **VK_RCONTROL**
- **VK_LMENU**
- **VK_RMENU**

These left- and right-distinguishing constants are available to an application only through the GetKeyboardState, SetKeyboardState, GetAsyncKeyState, GetKeyState, MapVirtualKey, and **MapVirtualKeyEx** functions. For list complete table of virtual key codes, see Virtual Key Codes.

NOTE

The winuser.h header defines MapVirtualKey as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

GetKeyState

GetKeyboardState

Keyboard Input

MapVirtualKeyEx

Reference

SetKeyboardState

MapVirtualKeyExA function (winuser.h)

7/6/2022 • 3 minutes to read • Edit Online

Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code. The function translates the codes using the input language and an input locale identifier.

Syntax

```
UINT MapVirtualKeyExA(
  [in]                UINT uCode,
  [in]                UINT uMapType,
  [in, out, optional] HKL  dwhkl
);
```

Parameters

`[in] uCode`

Type: **UINT**

The virtual key code or scan code for a key. How this value is interpreted depends on the value of the *uMapType* parameter.

Starting with Windows Vista, the high byte of the *uCode* value can contain either 0xe0 or 0xe1 to specify the extended scan code.

`[in] uMapType`

Type: **UINT**

The translation to perform. The value of this parameter depends on the value of the *uCode* parameter.

VALUE	MEANING
MAPVK_VK_TO_VSC 0	The *uCode* parameter is a virtual-key code and is translated into a scan code. If it is a virtual-key code that does not distinguish between left- and right-hand keys, the left-hand scan code is returned. If there is no translation, the function returns 0.
MAPVK_VSC_TO_VK 1	The *uCode* parameter is a scan code and is translated into a virtual-key code that does not distinguish between left- and right-hand keys. If there is no translation, the function returns 0.
MAPVK_VK_TO_CHAR 2	The *uCode* parameter is a virtual-key code and is translated into an unshifted character value in the low order word of the return value. Dead keys (diacritics) are indicated by setting the top bit of the return value. If there is no translation, the function returns 0.

VALUE	MEANING
MAPVK_VSC_TO_VK_EX 3	The *uCode* parameter is a scan code and is translated into a virtual-key code that distinguishes between left- and right-hand keys. If there is no translation, the function returns 0.
MAPVK_VK_TO_VSC_EX 4	**Windows Vista and later:** The *uCode* parameter is a virtual-key code and is translated into a scan code. If it is a virtual-key code that does not distinguish between left- and right-hand keys, the left-hand scan code is returned. If the scan code is an extended scan code, the high byte of the *uCode* value can contain either 0xe0 or 0xe1 to specify the extended scan code. If there is no translation, the function returns 0.

`[in, out, optional] dwhkl`

Type: **HKL**

Input locale identifier to use for translating the specified code. This parameter can be any input locale identifier previously returned by the LoadKeyboardLayout function.

Return value

Type: **UINT**

The return value is either a scan code, a virtual-key code, or a character value, depending on the value of *uCode* and *uMapType*. If there is no translation, the return value is zero.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

An application can use **MapVirtualKeyEx** to translate scan codes to the virtual-key code constants **VK_SHIFT**, **VK_CONTROL**, and **VK_MENU**, and vice versa. These translations do not distinguish between the left and right instances of the SHIFT, CTRL, or ALT keys.

An application can get the scan code corresponding to the left or right instance of one of these keys by calling **MapVirtualKeyEx** with *uCode* set to one of the following virtual-key code constants:

- VK_LSHIFT
- VK_RSHIFT
- VK_LCONTROL
- VK_RCONTROL
- VK_LMENU
- VK_RMENU

These left- and right-distinguishing constants are available to an application only through the GetKeyboardState, SetKeyboardState, GetAsyncKeyState, GetKeyState, MapVirtualKey, and **MapVirtualKeyEx** functions. For list complete table of virtual key codes, see Virtual Key Codes.

NOTE

The winuser.h header defines MapVirtualKeyEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

GetKeyState

GetKeyboardState

Keyboard Input

LoadKeyboardLayout

Reference

SetKeyboardState

MapVirtualKeyExW function (winuser.h)

7/6/2022 • 3 minutes to read • Edit Online

Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code. The function translates the codes using the input language and an input locale identifier.

Syntax

```
UINT MapVirtualKeyExW(
  [in]                UINT uCode,
  [in]                UINT uMapType,
  [in, out, optional] HKL  dwhkl
);
```

Parameters

`[in] uCode`

Type: **UINT**

The virtual key code or scan code for a key. How this value is interpreted depends on the value of the *uMapType* parameter.

Starting with Windows Vista, the high byte of the *uCode* value can contain either 0xe0 or 0xe1 to specify the extended scan code.

`[in] uMapType`

Type: **UINT**

The translation to perform. The value of this parameter depends on the value of the *uCode* parameter.

VALUE	MEANING
MAPVK_VK_TO_VSC 0	The *uCode* parameter is a virtual-key code and is translated into a scan code. If it is a virtual-key code that does not distinguish between left- and right-hand keys, the left-hand scan code is returned. If there is no translation, the function returns 0.
MAPVK_VSC_TO_VK 1	The *uCode* parameter is a scan code and is translated into a virtual-key code that does not distinguish between left- and right-hand keys. If there is no translation, the function returns 0.
MAPVK_VK_TO_CHAR 2	The *uCode* parameter is a virtual-key code and is translated into an unshifted character value in the low order word of the return value. Dead keys (diacritics) are indicated by setting the top bit of the return value. If there is no translation, the function returns 0.

VALUE	MEANING
MAPVK_VSC_TO_VK_EX 3	The *uCode* parameter is a scan code and is translated into a virtual-key code that distinguishes between left- and right-hand keys. If there is no translation, the function returns 0.
MAPVK_VK_TO_VSC_EX 4	**Windows Vista and later:** The *uCode* parameter is a virtual-key code and is translated into a scan code. If it is a virtual-key code that does not distinguish between left- and right-hand keys, the left-hand scan code is returned. If the scan code is an extended scan code, the high byte of the *uCode* value can contain either 0xe0 or 0xe1 to specify the extended scan code. If there is no translation, the function returns 0.

`[in, out, optional] dwhkl`

Type: **HKL**

Input locale identifier to use for translating the specified code. This parameter can be any input locale identifier previously returned by the LoadKeyboardLayout function.

Return value

Type: **UINT**

The return value is either a scan code, a virtual-key code, or a character value, depending on the value of *uCode* and *uMapType*. If there is no translation, the return value is zero.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

An application can use **MapVirtualKeyEx** to translate scan codes to the virtual-key code constants **VK_SHIFT**, **VK_CONTROL**, and **VK_MENU**, and vice versa. These translations do not distinguish between the left and right instances of the SHIFT, CTRL, or ALT keys.

An application can get the scan code corresponding to the left or right instance of one of these keys by calling **MapVirtualKeyEx** with *uCode* set to one of the following virtual-key code constants:

- VK_LSHIFT
- VK_RSHIFT
- VK_LCONTROL
- VK_RCONTROL
- VK_LMENU
- VK_RMENU

These left- and right-distinguishing constants are available to an application only through the GetKeyboardState, SetKeyboardState, GetAsyncKeyState, GetKeyState, MapVirtualKey, and **MapVirtualKeyEx** functions. For list complete table of virtual key codes, see Virtual Key Codes.

NOTE

The winuser.h header defines MapVirtualKeyEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

GetKeyState

GetKeyboardState

Keyboard Input

LoadKeyboardLayout

Reference

SetKeyboardState

MapVirtualKeyW function (winuser.h)

7/6/2022 • 3 minutes to read • Edit Online

Translates (maps) a virtual-key code into a scan code or character value, or translates a scan code into a virtual-key code.

To specify a handle to the keyboard layout to use for translating the specified code, use the MapVirtualKeyEx function.

Syntax

```
UINT MapVirtualKeyW(
  [in] UINT uCode,
  [in] UINT uMapType
);
```

Parameters

`[in] uCode`

Type: **UINT**

The virtual key code or scan code for a key. How this value is interpreted depends on the value of the *uMapType* parameter.

Starting with Windows Vista, the high byte of the *uCode* value can contain either 0xe0 or 0xe1 to specify the extended scan code.

`[in] uMapType`

Type: **UINT**

The translation to be performed. The value of this parameter depends on the value of the *uCode* parameter.

VALUE	MEANING
MAPVK_VK_TO_VSC 0	The *uCode* parameter is a virtual-key code and is translated into a scan code. If it is a virtual-key code that does not distinguish between left- and right-hand keys, the left-hand scan code is returned. If there is no translation, the function returns 0.
MAPVK_VSC_TO_VK 1	The *uCode* parameter is a scan code and is translated into a virtual-key code that does not distinguish between left- and right-hand keys. If there is no translation, the function returns 0.
MAPVK_VK_TO_CHAR 2	The *uCode* parameter is a virtual-key code and is translated into an unshifted character value in the low order word of the return value. Dead keys (diacritics) are indicated by setting the top bit of the return value. If there is no translation, the function returns 0.

VALUE	MEANING
MAPVK_VSC_TO_VK_EX 3	The *uCode* parameter is a scan code and is translated into a virtual-key code that distinguishes between left- and right-hand keys. If there is no translation, the function returns 0.
MAPVK_VK_TO_VSC_EX 4	**Windows Vista and later:** The *uCode* parameter is a virtual-key code and is translated into a scan code. If it is a virtual-key code that does not distinguish between left- and right-hand keys, the left-hand scan code is returned. If the scan code is an extended scan code, the high byte of the *uCode* value can contain either 0xe0 or 0xe1 to specify the extended scan code. If there is no translation, the function returns 0.

Return value

Type: **UINT**

The return value is either a scan code, a virtual-key code, or a character value, depending on the value of *uCode* and *uMapType*. If there is no translation, the return value is zero.

Remarks

An application can use **MapVirtualKey** to translate scan codes to the virtual-key code constants **VK_SHIFT**, **VK_CONTROL**, and **VK_MENU**, and vice versa. These translations do not distinguish between the left and right instances of the SHIFT, CTRL, or ALT keys.

An application can get the scan code corresponding to the left or right instance of one of these keys by calling **MapVirtualKey** with *uCode* set to one of the following virtual-key code constants:

- **VK_LSHIFT**
- **VK_RSHIFT**
- **VK_LCONTROL**
- **VK_RCONTROL**
- **VK_LMENU**
- **VK_RMENU**

These left- and right-distinguishing constants are available to an application only through the GetKeyboardState, SetKeyboardState, GetAsyncKeyState, GetKeyState, MapVirtualKey, and **MapVirtualKeyEx** functions. For list complete table of virtual key codes, see Virtual Key Codes.

NOTE

The winuser.h header defines MapVirtualKey as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

GetKeyState

GetKeyboardState

Keyboard Input

MapVirtualKeyEx

Reference

SetKeyboardState

mouse_event function (winuser.h)

7/6/2022 • 5 minutes to read • Edit Online

The **mouse_event** function synthesizes mouse motion and button clicks.

Note This function has been superseded. Use SendInput instead.

Syntax

```
void mouse_event(
  [in] DWORD     dwFlags,
  [in] DWORD     dx,
  [in] DWORD     dy,
  [in] DWORD     dwData,
  [in] ULONG_PTR dwExtraInfo
);
```

Parameters

`[in] dwFlags`

Type: **DWORD**

Controls various aspects of mouse motion and button clicking. This parameter can be certain combinations of the following values.

VALUE	MEANING
MOUSEEVENTF_ABSOLUTE 0x8000	The *dx* and *dy* parameters contain normalized absolute coordinates. If not set, those parameters contain relative data: the change in position since the last reported position. This flag can be set, or not set, regardless of what kind of mouse or mouse-like device, if any, is connected to the system. For further information about relative mouse motion, see the following Remarks section.
MOUSEEVENTF_LEFTDOWN 0x0002	The left button is down.
MOUSEEVENTF_LEFTUP 0x0004	The left button is up.
MOUSEEVENTF_MIDDLEDOWN 0x0020	The middle button is down.

MOUSEEVENTF_MIDDLEUP 0x0040	The middle button is up.
MOUSEEVENTF_MOVE 0x0001	Movement occurred.
MOUSEEVENTF_RIGHTDOWN 0x0008	The right button is down.
MOUSEEVENTF_RIGHTUP 0x0010	The right button is up.
MOUSEEVENTF_WHEEL 0x0800	The wheel has been moved, if the mouse has a wheel. The amount of movement is specified in *dwData*
MOUSEEVENTF_XDOWN 0x0080	An X button was pressed.
MOUSEEVENTF_XUP 0x0100	An X button was released.
MOUSEEVENTF_WHEEL 0x0800	The wheel button is rotated.
MOUSEEVENTF_HWHEEL 0x01000	The wheel button is tilted.

The values that specify mouse button status are set to indicate changes in status, not ongoing conditions. For example, if the left mouse button is pressed and held down, **MOUSEEVENTF_LEFTDOWN** is set when the left button is first pressed, but not for subsequent motions. Similarly, **MOUSEEVENTF_LEFTUP** is set only when the button is first released.

You cannot specify both **MOUSEEVENTF_WHEEL** and either **MOUSEEVENTF_XDOWN** or **MOUSEEVENTF_XUP** simultaneously in the *dwFlags* parameter, because they both require use of the *dwData* field.

`[in] dx`

Type: **DWORD**

The mouse's absolute position along the x-axis or its amount of motion since the last mouse event was generated, depending on the setting of **MOUSEEVENTF_ABSOLUTE**. Absolute data is specified as the mouse's actual x-coordinate; relative data is specified as the number of mickeys moved. A *mickey* is the amount that a mouse has to move for it to report that it has moved.

`[in] dy`

Type: **DWORD**

The mouse's absolute position along the y-axis or its amount of motion since the last mouse event was generated, depending on the setting of **MOUSEEVENTF_ABSOLUTE**. Absolute data is specified as the mouse's actual y-coordinate; relative data is specified as the number of mickeys moved.

`[in] dwData`

Type: **DWORD**

If *dwFlags* contains **MOUSEEVENTF_WHEEL**, then *dwData* specifies the amount of wheel movement. A positive value indicates that the wheel was rotated forward, away from the user; a negative value indicates that the wheel was rotated backward, toward the user. One wheel click is defined as **WHEEL_DELTA**, which is 120.

If *dwFlags* contains **MOUSEEVENTF_HWHEEL**, then *dwData* specifies the amount of wheel movement. A positive value indicates that the wheel was tilted to the right; a negative value indicates that the wheel was tilted to the left.

If *dwFlags* contains **MOUSEEVENTF_XDOWN** or **MOUSEEVENTF_XUP**, then *dwData* specifies which X buttons were pressed or released. This value may be any combination of the following flags.

If *dwFlags* is not **MOUSEEVENTF_WHEEL**, **MOUSEEVENTF_XDOWN**, or **MOUSEEVENTF_XUP**, then *dwData* should be zero.

VALUE	MEANING
XBUTTON1 0x0001	Set if the first X button was pressed or released.
XBUTTON2 0x0002	Set if the second X button was pressed or released.

`[in] dwExtraInfo`

Type: **ULONG_PTR**

An additional value associated with the mouse event. An application calls GetMessageExtraInfo to obtain this extra information.

Return value

None

Remarks

If the mouse has moved, indicated by **MOUSEEVENTF_MOVE** being set, *dx* and *dy* hold information about that motion. The information is specified as absolute or relative integer values.

If **MOUSEEVENTF_ABSOLUTE** value is specified, *dx* and *dy* contain normalized absolute coordinates between 0 and 65,535. The event procedure maps these coordinates onto the display surface. Coordinate (0,0) maps onto the upper-left corner of the display surface, (65535,65535) maps onto the lower-right corner.

If the **MOUSEEVENTF_ABSOLUTE** value is not specified, *dx* and *dy* specify relative motions from when the last mouse event was generated (the last reported position). Positive values mean the mouse moved right (or down);

negative values mean the mouse moved left (or up).

Relative mouse motion is subject to the settings for mouse speed and acceleration level. An end user sets these values using the Mouse application in Control Panel. An application obtains and sets these values with the SystemParametersInfo function.

The system applies two tests to the specified relative mouse motion when applying acceleration. If the specified distance along either the x or y axis is greater than the first mouse threshold value, and the mouse acceleration level is not zero, the operating system doubles the distance. If the specified distance along either the x- or y-axis is greater than the second mouse threshold value, and the mouse acceleration level is equal to two, the operating system doubles the distance that resulted from applying the first threshold test. It is thus possible for the operating system to multiply relatively-specified mouse motion along the x- or y-axis by up to four times.

Once acceleration has been applied, the system scales the resultant value by the desired mouse speed. Mouse speed can range from 1 (slowest) to 20 (fastest) and represents how much the pointer moves based on the distance the mouse moves. The default value is 10, which results in no additional modification to the mouse motion.

The **mouse_event** function is used to synthesize mouse events by applications that need to do so. It is also used by applications that need to obtain more information from the mouse than its position and button state. For example, if a tablet manufacturer wants to pass pen-based information to its own applications, it can write a DLL that communicates directly to the tablet hardware, obtains the extra information, and saves it in a queue. The DLL then calls **mouse_event** with the standard button and x/y position data, along with, in the *dwExtraInfo* parameter, some pointer or index to the queued extra information. When the application needs the extra information, it calls the DLL with the pointer or index stored in *dwExtraInfo*, and the DLL returns the extra information.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetMessageExtraInfo

Mouse Input

Other Resources

Reference

SystemParametersInfo

MOUSEINPUT structure (winuser.h)

7/6/2022 • 4 minutes to read • Edit Online

Contains information about a simulated mouse event.

Syntax

```
typedef struct tagMOUSEINPUT {
  LONG      dx;
  LONG      dy;
  DWORD     mouseData;
  DWORD     dwFlags;
  DWORD     time;
  ULONG_PTR dwExtraInfo;
} MOUSEINPUT, *PMOUSEINPUT, *LPMOUSEINPUT;
```

Members

`dx`

Type: **LONG**

The absolute position of the mouse, or the amount of motion since the last mouse event was generated, depending on the value of the **dwFlags** member. Absolute data is specified as the x coordinate of the mouse; relative data is specified as the number of pixels moved.

`dy`

Type: **LONG**

The absolute position of the mouse, or the amount of motion since the last mouse event was generated, depending on the value of the **dwFlags** member. Absolute data is specified as the y coordinate of the mouse; relative data is specified as the number of pixels moved.

`mouseData`

Type: **DWORD**

If **dwFlags** contains **MOUSEEVENTF_WHEEL**, then **mouseData** specifies the amount of wheel movement. A positive value indicates that the wheel was rotated forward, away from the user; a negative value indicates that the wheel was rotated backward, toward the user. One wheel click is defined as **WHEEL_DELTA**, which is 120.

Windows Vista: If *dwFlags* contains **MOUSEEVENTF_HWHEEL**, then *dwData* specifies the amount of wheel movement. A positive value indicates that the wheel was rotated to the right; a negative value indicates that the wheel was rotated to the left. One wheel click is defined as **WHEEL_DELTA**, which is 120.

If **dwFlags** does not contain **MOUSEEVENTF_WHEEL**, **MOUSEEVENTF_XDOWN**, or **MOUSEEVENTF_XUP**, then **mouseData** should be zero.

If **dwFlags** contains **MOUSEEVENTF_XDOWN** or **MOUSEEVENTF_XUP**, then **mouseData** specifies which X buttons were pressed or released. This value may be any combination of the following flags.

VALUE	MEANING
XBUTTON1 0x0001	Set if the first X button is pressed or released.
XBUTTON2 0x0002	Set if the second X button is pressed or released.

`dwFlags`

Type: **DWORD**

A set of bit flags that specify various aspects of mouse motion and button clicks. The bits in this member can be any reasonable combination of the following values.

The bit flags that specify mouse button status are set to indicate changes in status, not ongoing conditions. For example, if the left mouse button is pressed and held down, **MOUSEEVENTF_LEFTDOWN** is set when the left button is first pressed, but not for subsequent motions. Similarly **MOUSEEVENTF_LEFTUP** is set only when the button is first released.

You cannot specify both the **MOUSEEVENTF_WHEEL** flag and either **MOUSEEVENTF_XDOWN** or **MOUSEEVENTF_XUP** flags simultaneously in the **dwFlags** parameter, because they both require use of the **mouseData** field.

VALUE	MEANING
MOUSEEVENTF_MOVE 0x0001	Movement occurred.
MOUSEEVENTF_LEFTDOWN 0x0002	The left button was pressed.
MOUSEEVENTF_LEFTUP 0x0004	The left button was released.
MOUSEEVENTF_RIGHTDOWN 0x0008	The right button was pressed.
MOUSEEVENTF_RIGHTUP 0x0010	The right button was released.
MOUSEEVENTF_MIDDLEDOWN 0x0020	The middle button was pressed.
MOUSEEVENTF_MIDDLEUP 0x0040	The middle button was released.
MOUSEEVENTF_XDOWN 0x0080	An X button was pressed.
MOUSEEVENTF_XUP 0x0100	An X button was released.
MOUSEEVENTF_WHEEL 0x0800	The wheel was moved, if the mouse has a wheel. The amount of movement is specified in **mouseData**.

VALUE	MEANING
MOUSEEVENTF_HWHEEL 0x1000	The wheel was moved horizontally, if the mouse has a wheel. The amount of movement is specified in **mouseData**. **Windows XP/2000**: This value is not supported.
MOUSEEVENTF_MOVE_NOCOALESCE 0x2000	The WM_MOUSEMOVE messages will not be coalesced. The default behavior is to coalesce **WM_MOUSEMOVE** messages. **Windows XP/2000**: This value is not supported.
MOUSEEVENTF_VIRTUALDESK 0x4000	Maps coordinates to the entire desktop. Must be used with **MOUSEEVENTF_ABSOLUTE**.
MOUSEEVENTF_ABSOLUTE 0x8000	The **dx** and **dy** members contain normalized absolute coordinates. If the flag is not set, **dx**and **dy** contain relative data (the change in position since the last reported position). This flag can be set, or not set, regardless of what kind of mouse or other pointing device, if any, is connected to the system. For further information about relative mouse motion, see the following Remarks section.

`time`

Type: **DWORD**

The time stamp for the event, in milliseconds. If this parameter is 0, the system will provide its own time stamp.

`dwExtraInfo`

Type: **ULONG_PTR**

An additional value associated with the mouse event. An application calls GetMessageExtraInfo to obtain this extra information.

Remarks

If the mouse has moved, indicated by **MOUSEEVENTF_MOVE**, **dx** and **dy** specify information about that movement. The information is specified as absolute or relative integer values.

If **MOUSEEVENTF_ABSOLUTE** value is specified, **dx** and **dy** contain normalized absolute coordinates between 0 and 65,535. The event procedure maps these coordinates onto the display surface. Coordinate (0,0) maps onto the upper-left corner of the display surface; coordinate (65535,65535) maps onto the lower-right corner. In a multimonitor system, the coordinates map to the primary monitor.

If **MOUSEEVENTF_VIRTUALDESK** is specified, the coordinates map to the entire virtual desktop.

If the **MOUSEEVENTF_ABSOLUTE** value is not specified, **dx**and **dy** specify movement relative to the previous mouse event (the last reported position). Positive values mean the mouse moved right (or down); negative values mean the mouse moved left (or up).

Relative mouse motion is subject to the effects of the mouse speed and the two-mouse threshold values. A user sets these three values with the **Pointer Speed** slider of the Control Panel's **Mouse Properties** sheet. You can obtain and set these values using the SystemParametersInfo function.

The system applies two tests to the specified relative mouse movement. If the specified distance along either the x or y axis is greater than the first mouse threshold value, and the mouse speed is not zero, the system doubles the distance. If the specified distance along either the x or y axis is greater than the second mouse threshold value, and the mouse speed is equal to two, the system doubles the distance that resulted from applying the first threshold test. It is thus possible for the system to multiply specified relative mouse movement along the x or y axis by up to four times.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetMessageExtraInfo

INPUT

Keyboard Input

Reference

SendInput

MOUSEMOVEPOINT structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Contains information about the mouse's location in screen coordinates.

Syntax

```
typedef struct tagMOUSEMOVEPOINT {
  int       x;
  int       y;
  DWORD     time;
  ULONG_PTR dwExtraInfo;
} MOUSEMOVEPOINT, *PMOUSEMOVEPOINT, *LPMOUSEMOVEPOINT;
```

Members

`x`

Type: **int**

The x-coordinate of the mouse.

`y`

Type: **int**

The y-coordinate of the mouse.

`time`

Type: **DWORD**

The time stamp of the mouse coordinate.

`dwExtraInfo`

Type: **ULONG_PTR**

Additional information associated with this coordinate.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetMouseMovePointsEx

Mouse Input

Reference

NEXTRAWINPUTBLOCK macro (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Retrieves the location of the next structure in an array of RAWINPUT structures.

Syntax

```
void NEXTRAWINPUTBLOCK(
   ptr
);
```

Parameters

`ptr`

A pointer to a structure in an array of RAWINPUT structures.

Return value

None

Remarks

This macro is called repeatedly to traverse an array of RAWINPUT structures.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

See also

Conceptual

RAWINPUT

Raw Input

Reference

OemKeyScan function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Maps OEMASCII codes 0 through 0x0FF into the OEM scan codes and shift states. The function provides information that allows a program to send OEM text to another program by simulating keyboard input.

Syntax

```
DWORD OemKeyScan(
  [in] WORD wOemChar
);
```

Parameters

`[in] wOemChar`

Type: **WORD**

The ASCII value of the OEM character.

Return value

Type: **DWORD**

The low-order word of the return value contains the scan code of the OEM character, and the high-order word contains the shift state, which can be a combination of the following bits.

BIT	DESCRIPTION
1	Either SHIFT key is pressed.
2	Either CTRL key is pressed.
4	Either ALT key is pressed.
8	The Hankaku key is pressed.
16	Reserved (defined by the keyboard layout driver).
32	Reserved (defined by the keyboard layout driver).

If the character cannot be produced by a single keystroke using the current keyboard layout, the return value is

–1.

Remarks

This function does not provide translations for characters that require CTRL+ALT or dead keys. Characters not translated by this function must be copied by simulating input using the ALT+ keypad mechanism. The NUMLOCK key must be off.

This function does not provide translations for characters that cannot be typed with one keystroke using the current keyboard layout, such as characters with diacritics requiring dead keys. Characters not translated by this function may be simulated using the ALT+ keypad mechanism. The NUMLOCK key must be on.

This function is implemented using the VkKeyScan function.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Keyboard Input

Reference

VkKeyScan

RAWHID structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Describes the format of the raw input from a Human Interface Device (HID).

Syntax

```
typedef struct tagRAWHID {
  DWORD dwSizeHid;
  DWORD dwCount;
  BYTE  bRawData[1];
} RAWHID, *PRAWHID, *LPRAWHID;
```

Members

`dwSizeHid`

Type: **DWORD**

The size, in bytes, of each HID input in **bRawData**.

`dwCount`

Type: **DWORD**

The number of HID inputs in **bRawData**.

`bRawData`

Type: **BYTE[1]**

The raw input data, as an array of bytes.

Remarks

Each WM_INPUT can indicate several inputs, but all of the inputs come from the same HID. The size of the **bRawData** array is **dwSizeHid** * **dwCount**.

For more information, see Interpreting HID Reports.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

RAWINPUT

Raw Input

Introduction to Human Interface Devices (HID)

Reference

WM_INPUT

Interpreting HID Reports

RAWINPUT structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Contains the raw input from a device.

Syntax

```
typedef struct tagRAWINPUT {
  RAWINPUTHEADER header;
  union {
    RAWMOUSE    mouse;
    RAWKEYBOARD keyboard;
    RAWHID      hid;
  } data;
} RAWINPUT, *PRAWINPUT, *LPRAWINPUT;
```

Members

`header`

Type: **RAWINPUTHEADER**

The raw input data.

`data`

`data.mouse`

Type: RAWMOUSE If the data comes from a mouse, this is the raw input data.

`data.keyboard`

Type: RAWKEYBOARD If the data comes from a keyboard, this is the raw input data.

`data.hid`

Type: RAWHID If the data comes from an HID, this is the raw input data.

Remarks

The handle to this structure is passed in the *lParam* parameter of WM_INPUT.

To get detailed information -- such as the header and the content of the raw input -- call GetRawInputData.

To read the **RAWINPUT** in the message loop as a buffered read, call GetRawInputBuffer.

To get device specific information, call GetRawInputDeviceInfo with the *hDevice* from RAWINPUTHEADER.

Raw input is available only when the application calls RegisterRawInputDevices with valid device specifications.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetRawInputBuffer

GetRawInputData

RAWHID

RAWINPUTHEADER

RAWKEYBOARD

RAWMOUSE

Raw Input

Reference

RAWINPUTDEVICE structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Defines information for the raw input devices.

Syntax

```
typedef struct tagRAWINPUTDEVICE {
  USHORT usUsagePage;
  USHORT usUsage;
  DWORD  dwFlags;
  HWND   hwndTarget;
} RAWINPUTDEVICE, *PRAWINPUTDEVICE, *LPRAWINPUTDEVICE;
```

Members

`usUsagePage`

Type: **USHORT**

Top level collection Usage page for the raw input device. See HID Clients Supported in Windows for details on possible values.

`usUsage`

Type: **USHORT**

Top level collection Usage ID for the raw input device. See HID Clients Supported in Windows for details on possible values.

`dwFlags`

Type: **DWORD**

Mode flag that specifies how to interpret the information provided by **usUsagePage** and **usUsage**. It can be zero (the default) or one of the following values. By default, the operating system sends raw input from devices with the specified top level collection (TLC) to the registered application as long as it has the window focus.

VALUE	MEANING
RIDEV_REMOVE 0x00000001	If set, this removes the top level collection from the inclusion list. This tells the operating system to stop reading from a device which matches the top level collection.
RIDEV_EXCLUDE 0x00000010	If set, this specifies the top level collections to exclude when reading a complete usage page. This flag only affects a TLC whose usage page is already specified with **RIDEV_PAGEONLY**.
RIDEV_PAGEONLY 0x00000020	If set, this specifies all devices whose top level collection is from the specified **usUsagePage**. Note that **usUsage** must be zero. To exclude a particular top level collection, use **RIDEV_EXCLUDE**.

RIDEV_NOLEGACY 0x00000030	If set, this prevents any devices specified by **usUsagePage** or **usUsage** from generating legacy messages. This is only for the mouse and keyboard. See Remarks.
RIDEV_INPUTSINK 0x00000100	If set, this enables the caller to receive the input even when the caller is not in the foreground. Note that **hwndTarget** must be specified.
RIDEV_CAPTUREMOUSE 0x00000200	If set, the mouse button click does not activate the other window. **RIDEV_CAPTUREMOUSE** can be specified only if **RIDEV_NOLEGACY** is specified for a mouse device.
RIDEV_NOHOTKEYS 0x00000200	If set, the application-defined keyboard device hotkeys are not handled. However, the system hotkeys; for example, ALT+TAB and CTRL+ALT+DEL, are still handled. By default, all keyboard hotkeys are handled. **RIDEV_NOHOTKEYS** can be specified even if **RIDEV_NOLEGACY** is not specified and **hwndTarget** is **NULL**.
RIDEV_APPKEYS 0x00000400	If set, the application command keys are handled. **RIDEV_APPKEYS** can be specified only if **RIDEV_NOLEGACY** is specified for a keyboard device.
RIDEV_EXINPUTSINK 0x00001000	If set, this enables the caller to receive input in the background only if the foreground application does not process it. In other words, if the foreground application is not registered for raw input, then the background application that is registered will receive the input. **Windows XP:** This flag is not supported until Windows Vista
RIDEV_DEVNOTIFY 0x00002000	If set, this enables the caller to receive WM_INPUT_DEVICE_CHANGE notifications for device arrival and device removal. **Windows XP:** This flag is not supported until Windows Vista

`hwndTarget`

Type: **HWND**

A handle to the target window. If **NULL** it follows the keyboard focus.

Remarks

If **RIDEV_NOLEGACY** is set for a mouse or a keyboard, the system does not generate any legacy message for that device for the application. For example, if the mouse TLC is set with **RIDEV_NOLEGACY**, **WM_LBUTTONDOWN** and related legacy mouse messages are not generated. Likewise, if the keyboard TLC is set with **RIDEV_NOLEGACY**, **WM_KEYDOWN** and related legacy keyboard messages are not generated.

If **RIDEV_REMOVE** is set and the **hwndTarget** member is not set to **NULL**, then RegisterRawInputDevices function will fail.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetRegisteredRawInputDevices

Raw Input

Introduction to Human Interface Devices (HID)

HID Clients Supported in Windows

HID USB homepage

Reference

RegisterRawInputDevices

RAWINPUTDEVICELIST structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Contains information about a raw input device.

Syntax

```
typedef struct tagRAWINPUTDEVICELIST {
  HANDLE hDevice;
  DWORD  dwType;
} RAWINPUTDEVICELIST, *PRAWINPUTDEVICELIST;
```

Members

`hDevice`

Type: **HANDLE**

A handle to the raw input device.

`dwType`

Type: **DWORD**

The type of device. This can be one of the following values.

VALUE	MEANING
RIM_TYPEHID 2	The device is an HID that is not a keyboard and not a mouse.
RIM_TYPEKEYBOARD 1	The device is a keyboard.
RIM_TYPEMOUSE 0	The device is a mouse.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetRawInputDeviceList

Raw Input

Reference

RAWINPUTHEADER structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Contains the header information that is part of the raw input data.

Syntax

```
typedef struct tagRAWINPUTHEADER {
  DWORD  dwType;
  DWORD  dwSize;
  HANDLE hDevice;
  WPARAM wParam;
} RAWINPUTHEADER, *PRAWINPUTHEADER, *LPRAWINPUTHEADER;
```

Members

`dwType`

Type: **DWORD**

The type of raw input. It can be one of the following values:

VALUE	MEANING
RIM_TYPEMOUSE 0	Raw input comes from the mouse.
RIM_TYPEKEYBOARD 1	Raw input comes from the keyboard.
RIM_TYPEHID 2	Raw input comes from some device that is not a keyboard or a mouse.

`dwSize`

Type: **DWORD**

The size, in bytes, of the entire input packet of data. This includes RAWINPUT plus possible extra input reports in the RAWHID variable length array.

`hDevice`

Type: **HANDLE**

A handle to the device generating the raw input data.

`wParam`

Type: **WPARAM**

The value passed in the *wParam* parameter of the WM_INPUT message.

Remarks

To get more information on the device, use **hDevice** in a call to GetRawInputDeviceInfo. **hDevice** can be zero if an input is received from a precision touchpad.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetRawInputDeviceInfo

RAWINPUT structure

RAWKEYBOARD structure

RAWMOUSE structure

RAWHID structure

Raw Input

Reference

WM_INPUT

RAWKEYBOARD structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Contains information about the state of the keyboard.

Syntax

```
typedef struct tagRAWKEYBOARD {
  USHORT MakeCode;
  USHORT Flags;
  USHORT Reserved;
  USHORT VKey;
  UINT   Message;
  ULONG  ExtraInformation;
} RAWKEYBOARD, *PRAWKEYBOARD, *LPRAWKEYBOARD;
```

Members

`MakeCode`

Type: **USHORT**

Specifies the scan code (from Scan Code Set 1) associated with a key press. See Remarks.

`Flags`

Type: **USHORT**

Flags for scan code information. It can be one or more of the following:

VALUE	MEANING
RI_KEY_MAKE 0	The key is down.
RI_KEY_BREAK 1	The key is up.
RI_KEY_E0 2	The scan code has the E0 prefix.
RI_KEY_E1 4	The scan code has the E1 prefix.

`Reserved`

Type: **USHORT**

Reserved; must be zero.

`VKey`

Type: **USHORT**

The corresponding legacy virtual-key code.

`Message`

Type: **UINT**

The corresponding legacy keyboard window message, for example WM_KEYDOWN, WM_SYSKEYDOWN, and so forth.

`ExtraInformation`

Type: **ULONG**

The device-specific additional information for the event.

Remarks

For a **MakeCode** value HID client mapper driver converts HID usages into scan codes according to USB HID to PS/2 Scan Code Translation Table (see **PS/2 Set 1 Make** column).

Older PS/2 keyboards actually transmit Scan Code Set 2 values down the wire from the keyboard to the keyboard port. These values are translated to Scan Code Set 1 by the i8042 port chip. Possible values are listed in Keyboard Scan Code Specification (see **Scan Code Table**).

KEYBOARD_OVERRUN_MAKE_CODE is a special **MakeCode** value sent when an invalid or unrecognizable combination of keys is pressed or the number of keys pressed exceeds the limit for this keyboard.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetRawInputDeviceInfo

RAWINPUT

Raw Input

Keyboard and mouse HID client drivers

Reference

USB HID to PS/2 Scan Code Translation Table

PS/2 Keyboard Scan Code Specification

KEYBOARD_INPUT_DATA structure

RAWMOUSE structure (winuser.h)

7/6/2022 • 5 minutes to read • Edit Online

Contains information about the state of the mouse.

Syntax

```
typedef struct tagRAWMOUSE {
  USHORT usFlags;
  union {
    ULONG ulButtons;
    struct {
      USHORT usButtonFlags;
      USHORT usButtonData;
    } DUMMYSTRUCTNAME;
  } DUMMYUNIONNAME;
  ULONG  ulRawButtons;
  LONG   lLastX;
  LONG   lLastY;
  ULONG  ulExtraInformation;
} RAWMOUSE, *PRAWMOUSE, *LPRAWMOUSE;
```

Members

`usFlags`

Type: **USHORT**

The mouse state. This member can be any reasonable combination of the following.

VALUE	MEANING
MOUSE_MOVE_RELATIVE 0x00	Mouse movement data is relative to the last mouse position. For further information about mouse motion, see the following Remarks section.
MOUSE_MOVE_ABSOLUTE 0x01	Mouse movement data is based on absolute position. For further information about mouse motion, see the following Remarks section.
MOUSE_VIRTUAL_DESKTOP 0x02	Mouse coordinates are mapped to the virtual desktop (for a multiple monitor system). For further information about mouse motion, see the following Remarks section.
MOUSE_ATTRIBUTES_CHANGED 0x04	Mouse attributes changed; application needs to query the mouse attributes.
MOUSE_MOVE_NOCOALESCE 0x08	This mouse movement event was not coalesced. Mouse movement events can be coalesced by default. Windows XP/2000: This value is not supported.

`DUMMYUNIONNAME`

`DUMMYUNIONNAME.ulButtons`

Type: **ULONG**

Reserved.

`DUMMYUNIONNAME.DUMMYSTRUCTNAME`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.usButtonFlags`

Type: **USHORT**

The transition state of the mouse buttons. This member can be one or more of the following values.

VALUE	MEANING
RI_MOUSE_BUTTON_1_DOWN **RI_MOUSE_LEFT_BUTTON_DOWN** 0x0001	Left button changed to down.
RI_MOUSE_BUTTON_1_UP **RI_MOUSE_LEFT_BUTTON_UP** 0x0002	Left button changed to up.
RI_MOUSE_BUTTON_2_DOWN **RI_MOUSE_RIGHT_BUTTON_DOWN** 0x0004	Right button changed to down.
RI_MOUSE_BUTTON_2_UP **RI_MOUSE_RIGHT_BUTTON_UP** 0x0008	Right button changed to up.
RI_MOUSE_BUTTON_3_DOWN **RI_MOUSE_MIDDLE_BUTTON_DOWN** 0x0010	Middle button changed to down.
RI_MOUSE_BUTTON_3_UP **RI_MOUSE_MIDDLE_BUTTON_UP** 0x0020	Middle button changed to up.
RI_MOUSE_BUTTON_4_DOWN 0x0040	XBUTTON1 changed to down.
RI_MOUSE_BUTTON_4_UP 0x0080	XBUTTON1 changed to up.
RI_MOUSE_BUTTON_5_DOWN 0x0100	XBUTTON2 changed to down.
RI_MOUSE_BUTTON_5_UP 0x0200	XBUTTON2 changed to up.
RI_MOUSE_WHEEL 0x0400	Raw input comes from a mouse wheel. The wheel delta is stored in **usButtonData**. A positive value indicates that the wheel was rotated forward, away from the user; a negative value indicates that the wheel was rotated backward, toward the user. For further information see the following Remarks section.

VALUE	MEANING
RI_MOUSE_HWHEEL 0x0800	Raw input comes from a horizontal mouse wheel. The wheel delta is stored in **usButtonData**. A positive value indicates that the wheel was rotated to the right; a negative value indicates that the wheel was rotated to the left. For further information see the following Remarks section. Windows XP/2000: This value is not supported.

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.usButtonData`

Type: **USHORT**

If **usButtonFlags** has **RI_MOUSE_WHEEL** or **RI_MOUSE_HWHEEL**, this member specifies the distance the wheel is rotated. For further information see the following Remarks section.

`ulRawButtons`

Type: **ULONG**

The raw state of the mouse buttons. The Win32 subsystem does not use this member.

`lLastX`

Type: **LONG**

The motion in the X direction. This is signed relative motion or absolute motion, depending on the value of **usFlags**.

`lLastY`

Type: **LONG**

The motion in the Y direction. This is signed relative motion or absolute motion, depending on the value of **usFlags**.

`ulExtraInformation`

Type: **ULONG**

The device-specific additional information for the event.

Remarks

If the mouse has moved, indicated by **MOUSE_MOVE_RELATIVE** or **MOUSE_MOVE_ABSOLUTE**, **lLastX** and **lLastY** specify information about that movement. The information is specified as relative or absolute integer values.

If **MOUSE_MOVE_RELATIVE** value is specified, **lLastX** and **lLastY** specify movement relative to the previous mouse event (the last reported position). Positive values mean the mouse moved right (or down); negative values mean the mouse moved left (or up).

If **MOUSE_MOVE_ABSOLUTE** value is specified, **lLastX** and **lLastY** contain normalized absolute coordinates between 0 and 65,535. Coordinate (0,0) maps onto the upper-left corner of the display surface; coordinate (65535,65535) maps onto the lower-right corner. In a multimonitor system, the coordinates map to the primary monitor.

If **MOUSE_VIRTUAL_DESKTOP** is specified in addition to **MOUSE_MOVE_ABSOLUTE**, the coordinates map to the entire virtual desktop.

```
if ((rawMouse.usFlags & MOUSE_MOVE_ABSOLUTE) == MOUSE_MOVE_ABSOLUTE)
{
    bool isVirtualDesktop = (rawMouse.usFlags & MOUSE_VIRTUAL_DESKTOP) == MOUSE_VIRTUAL_DESKTOP;

    int width = GetSystemMetrics(isVirtualDesktop ? SM_CXVIRTUALSCREEN : SM_CXSCREEN);
    int height = GetSystemMetrics(isVirtualDesktop ? SM_CYVIRTUALSCREEN : SM_CYSCREEN);

    int absoluteX = int((rawMouse.lLastX / 65535.0f) * width);
    int absoluteY = int((rawMouse.lLastY / 65535.0f) * height);
}
else if (rawMouse.lLastX != 0 || rawMouse.lLastY != 0)
{
    int relativeX = rawMouse.lLastX;
    int relativeY = rawMouse.lLastY;
}
```

In contrast to legacy WM_MOUSEMOVE window messages Raw Input mouse events is not subject to the effects of the mouse speed set in the Control Panel's **Mouse Properties** sheet. See About Mouse Input for details.

If mouse wheel is moved, indicated by **RI_MOUSE_WHEEL** or **RI_MOUSE_HWHEEL** in **usButtonFlags**, then **usButtonData** contains a signed **short** value that specifies the distance the wheel is rotated.

The wheel rotation will be a multiple of **WHEEL_DELTA**, which is set at 120. This is the threshold for action to be taken, and one such action (for example, scrolling one increment) should occur for each delta.

The delta was set to 120 to allow Microsoft or other vendors to build finer-resolution wheels (a freely-rotating wheel with no notches) to send more messages per rotation, but with a smaller value in each message. To use this feature, you can either add the incoming delta values until **WHEEL_DELTA** is reached (so for a delta-rotation you get the same response), or scroll partial lines in response to the more frequent messages. You can also choose your scroll granularity and accumulate deltas until it is reached.

The application could also retrieve the current lines-to-scroll and characters-to-scroll user setting by using the SystemParametersInfo API with **SPI_GETWHEELSCROLLLINES** or **SPI_GETWHEELSCROLLCHARS** parameter.

Here is example of such wheel handling code:

```
if ((rawMouse.usButtonFlags & RI_MOUSE_WHEEL) == RI_MOUSE_WHEEL ||
    (rawMouse.usButtonFlags & RI_MOUSE_HWHEEL) == RI_MOUSE_HWHEEL)
{
    static const unsigned long defaultScrollLinesPerWheelDelta = 3;
    static const unsigned long defaultScrollCharsPerWheelDelta = 1;

    float wheelDelta = (float)(short)rawMouse.usButtonData;
    float numTicks = wheelDelta / WHEEL_DELTA;

    bool isHorizontalScroll = (rawMouse.usButtonFlags & RI_MOUSE_HWHEEL) == RI_MOUSE_HWHEEL;
    bool isScrollByPage = false;
    float scrollDelta = numTicks;

    if (isHorizontalScroll)
    {
        unsigned long scrollChars = defaultScrollCharsPerWheelDelta;
        SystemParametersInfo(SPI_GETWHEELSCROLLCHARS, 0, &scrollChars, 0);
        scrollDelta *= scrollChars;
    }
    else
    {
        unsigned long scrollLines = defaultScrollLinesPerWheelDelta;
        SystemParametersInfo(SPI_GETWHEELSCROLLLINES, 0, &scrollLines, 0);
        if (scrollLines == WHEEL_PAGESCROLL)
            isScrollByPage = true;
        else
            scrollDelta *= scrollLines;
    }
}
```

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetRawInputDeviceInfo

RAWINPUT

Raw Input

Reference

MOUSEINPUT structure

SendInput function

MOUSE_INPUT_DATA structure

About Mouse Input (legacy)

Mouse Input Notifications (legacy)

RegisterForTooltipDismissNotification function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Registers or unregisters windows to receive notification to dismiss their tooltip windows.

Syntax

```
BOOL RegisterForTooltipDismissNotification(
  HWND                  hWnd,
  TOOLTIP_DISMISS_FLAGS tdFlags
);
```

Parameters

`hWnd`

Type: **HWND**

The handle of the window to receive the **WM_TOOLTIPDISMISS** message.

`tdFlags`

Type: TOOLTIP_DISMISS_FLAGS

A value of the enumeration that specifies whether the function registers or unregisters the window. **TDF_REGISTER** to register; **TDF_UNREGISTER** to unregister.

Return value

TRUE if the window was successfully registered or unregistered; otherwise, **FALSE**. (See Remarks.)

Remarks

This function makes tooltips more accessible by letting apps and frameworks that support tooltips register and unregister to be notified by a **WM_TOOLTIPDISMISS** message when the system requires all showing tooltips to be dismissed.

Apps should register for this notification each time they show a tooltip and hide their tooltips in response to a **WM_TOOLTIPDISMISS** message. When a tooltip is hidden for some other reason, like a mouse action, the app should unregister.

System-defined triggers for tooltip dismissal include a solitary Ctrl key up or Ctrl+Shift+F10. (The set of triggers may change over time.)

The function takes either the **HWND** of a tooltip window or the **HWND** of an app window that has child tooltips.

- If a tooltip **HWND** itself is registered, the tooltip window is expected to register upon showing and to dismiss upon receiving a **WM_TOOLTIPDISMISS** message.
- If an app **HWND** registers on behalf of its tooltips, the app's window is expected to register upon showing tooltips and dismiss all of its tooltips upon receiving a **WM_TOOLTIPDISMISS** message.

Tooltip or app windows are expected to call the function to register each time tooltips are shown. Registered windows are automatically unregistered upon posting **WM_TOOLTIPDISMISS**.

The **TDF_UNREGISTER** flag is used to explicitly unregister a window when a tooltip window is dismissed by application or framework prerogative (such as moving the cursor out of the "safe zone"). If an app or framework calls `RegisterForTooltipDismissNotification` with **TDF_UNREGISTER** after the window has been automatically unregistered, the function returns **FALSE**. There is no impact on future registrations.

Return values

The HWND passed into the function must be owned by the calling process; otherwise, the function returns **FALSE**.

When called with **TDF_REGISTER** and a window belonging to the calling process, the function returns **TRUE** if the window was successfully registered or **FALSE** if the window was already registered. The window is treated as registered either way.

When called with **TDF_UNREGISTER** and a windows belonging to the calling process, the function returns **TRUE** if the window is successfully unregistered, or **FALSE** if the windows was not currently registered. The window is treated as unregistered either way.

Requirements

Minimum supported client	Windows 11 Build 22621
Target Platform	Windows
Header	winuser.h
Library	user32.lib

See also

Tooltip

RegisterHotKey function (winuser.h)

7/6/2022 • 3 minutes to read • Edit Online

Defines a system-wide hot key.

Syntax

```
BOOL RegisterHotKey(
  [in, optional] HWND hWnd,
  [in]           int  id,
  [in]           UINT fsModifiers,
  [in]           UINT vk
);
```

Parameters

`[in, optional] hWnd`

Type: **HWND**

A handle to the window that will receive WM_HOTKEY messages generated by the hot key. If this parameter is **NULL**, **WM_HOTKEY** messages are posted to the message queue of the calling thread and must be processed in the message loop.

`[in] id`

Type: **int**

The identifier of the hot key. If the *hWnd* parameter is NULL, then the hot key is associated with the current thread rather than with a particular window. If a hot key already exists with the same *hWnd* and *id* parameters, see Remarks for the action taken.

`[in] fsModifiers`

Type: **UINT**

The keys that must be pressed in combination with the key specified by the *uVirtKey* parameter in order to generate the WM_HOTKEY message. The *fsModifiers* parameter can be a combination of the following values.

VALUE	MEANING
MOD_ALT 0x0001	Either ALT key must be held down.
MOD_CONTROL 0x0002	Either CTRL key must be held down.
MOD_NOREPEAT 0x4000	Changes the hotkey behavior so that the keyboard auto-repeat does not yield multiple hotkey notifications. **Windows Vista:** This flag is not supported.

MOD_SHIFT 0x0004	Either SHIFT key must be held down.
MOD_WIN 0x0008	Either WINDOWS key was held down. These keys are labeled with the Windows logo. Keyboard shortcuts that involve the WINDOWS key are reserved for use by the operating system.

`[in] vk`

Type: **UINT**

The virtual-key code of the hot key. See Virtual Key Codes.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

When a key is pressed, the system looks for a match against all hot keys. Upon finding a match, the system posts the WM_HOTKEY message to the message queue of the window with which the hot key is associated. If the hot key is not associated with a window, then the **WM_HOTKEY** message is posted to the thread associated with the hot key.

This function cannot associate a hot key with a window created by another thread.

RegisterHotKey fails if the keystrokes specified for the hot key have already been registered by another hot key.

If a hot key already exists with the same *hWnd* and *id* parameters, it is maintained along with the new hot key. The application must explicitly call UnregisterHotKey to unregister the old hot key.

Windows Server 2003: If a hot key already exists with the same *hWnd* and *id* parameters, it is replaced by the new hot key.

The F12 key is reserved for use by the debugger at all times, so it should not be registered as a hot key. Even when you are not debugging an application, F12 is reserved in case a kernel-mode debugger or a just-in-time debugger is resident.

An application must specify an id value in the range 0x0000 through 0xBFFF. A shared DLL must specify a value in the range 0xC000 through 0xFFFF (the range returned by the GlobalAddAtom function). To avoid conflicts with hot-key identifiers defined by other shared DLLs, a DLL should use the **GlobalAddAtom** function to obtain the hot-key identifier.

Examples

The following example shows how to use the **RegisterHotKey** function with the **MOD_NOREPEAT** flag. In this example, the hotkey 'ALT+b' is registered for the main thread. When the hotkey is pressed, the thread will receive a WM_HOTKEY message, which will get picked up in the GetMessage call. Because this example uses **MOD_ALT** with the **MOD_NOREPEAT** value for *fsModifiers*, the thread will only receive another **WM_HOTKEY** message when the 'b' key is released and then pressed again while the 'ALT' key is being pressed down.

```
#include "stdafx.h"

int _cdecl _tmain (
    int argc,
    TCHAR *argv[])
{
    if (RegisterHotKey(
        NULL,
        1,
        MOD_ALT | MOD_NOREPEAT,
        0x42))  //0x42 is 'b'
    {
        _tprintf(_T("Hotkey 'ALT+b' registered, using MOD_NOREPEAT flag\n"));
    }

    MSG msg = {0};
    while (GetMessage(&msg, NULL, 0, 0) != 0)
    {
        if (msg.message == WM_HOTKEY)
        {
            _tprintf(_T("WM_HOTKEY received\n"));
        }
    }

    return 0;
}
```

Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GlobalAddAtom

Keyboard Input

Reference

Register hotkey for the current app (CSRegisterHotkey)

Register hotkey for the current app (CppRegisterHotkey)

Register hotkey for the current app (VBRegisterHotkey)

Samples

UnregisterHotKey

WM_HOTKEY

RegisterRawInputDevices function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Registers the devices that supply the raw input data.

Syntax

```
BOOL RegisterRawInputDevices(
  [in] PCRAWINPUTDEVICE pRawInputDevices,
  [in] UINT             uiNumDevices,
  [in] UINT             cbSize
);
```

Parameters

`[in] pRawInputDevices`

Type: **PCRAWINPUTDEVICE**

An array of RAWINPUTDEVICE structures that represent the devices that supply the raw input.

`[in] uiNumDevices`

Type: **UINT**

The number of RAWINPUTDEVICE structures pointed to by *pRawInputDevices*.

`[in] cbSize`

Type: **UINT**

The size, in bytes, of a RAWINPUTDEVICE structure.

Return value

Type: **BOOL**

TRUE if the function succeeds; otherwise, **FALSE**. If the function fails, call GetLastError for more information.

Remarks

To receive WM_INPUT messages, an application must first register the raw input devices using **RegisterRawInputDevices**. By default, an application does not receive raw input.

To receive WM_INPUT_DEVICE_CHANGE messages, an application must specify the RIDEV_DEVNOTIFY flag for each device class that is specified by the usUsagePage and usUsage fields of the RAWINPUTDEVICE structure . By default, an application does not receive **WM_INPUT_DEVICE_CHANGE** notifications for raw input device arrival and removal.

If a RAWINPUTDEVICE structure has the RIDEV_REMOVE flag set and the hwndTarget parameter is not set to NULL, then parameter validation will fail.

Only one window per raw input device class may be registered to receive raw input within a process (the window passed in the last call to RegisterRawInputDevices). Because of this, RegisterRawInputDevices should not be used from a library, as it may interfere with any raw input processing logic already present in applications that load it.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-rawinput-l1-1-0 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

RAWINPUTDEVICE

Raw Input

Reference

WM_INPUT

ReleaseCapture function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Releases the mouse capture from a window in the current thread and restores normal mouse input processing. A window that has captured the mouse receives all mouse input, regardless of the position of the cursor, except when a mouse button is clicked while the cursor hot spot is in the window of another thread.

Syntax

```
BOOL ReleaseCapture();
```

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

An application calls this function after calling the SetCapture function.

Examples

For an example, see Drawing Lines with the Mouse.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-mouse-l1-1-0 (introduced in Windows 8)

See also

Conceptual

GetCapture

Mouse Input

Reference

SetCapture

WM_CAPTURECHANGED

RID_DEVICE_INFO structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Defines the raw input data coming from any device.

Syntax

```
typedef struct tagRID_DEVICE_INFO {
  DWORD cbSize;
  DWORD dwType;
  union {
    RID_DEVICE_INFO_MOUSE    mouse;
    RID_DEVICE_INFO_KEYBOARD keyboard;
    RID_DEVICE_INFO_HID      hid;
  } DUMMYUNIONNAME;
} RID_DEVICE_INFO, *PRID_DEVICE_INFO, *LPRID_DEVICE_INFO;
```

Members

`cbSize`

Type: **DWORD**

The size, in bytes, of the **RID_DEVICE_INFO** structure.

`dwType`

Type: **DWORD**

The type of raw input data. This member can be one of the following values.

VALUE	MEANING
RIM_TYPEMOUSE 0	Data comes from a mouse.
RIM_TYPEKEYBOARD 1	Data comes from a keyboard.
RIM_TYPEHID 2	Data comes from an HID that is not a keyboard or a mouse.

`DUMMYUNIONNAME`

`DUMMYUNIONNAME.mouse`

Type: **RID_DEVICE_INFO_MOUSE**

If **dwType** is **RIM_TYPEMOUSE**, this is the RID_DEVICE_INFO_MOUSE structure that defines the mouse.

`DUMMYUNIONNAME.keyboard`

Type: **RID_DEVICE_INFO_KEYBOARD**

If **dwType** is **RIM_TYPEKEYBOARD**, this is the RID_DEVICE_INFO_KEYBOARD structure that defines the keyboard.

`DUMMYUNIONNAME.hid`

Type: **RID_DEVICE_INFO_HID**

If **dwType** is **RIM_TYPEHID**, this is the RID_DEVICE_INFO_HID structure that defines the HID device.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

GetRawInputDeviceInfo

RID_DEVICE_INFO_HID

RID_DEVICE_INFO_KEYBOARD

RID_DEVICE_INFO_MOUSE

Raw Input

Reference

RID_DEVICE_INFO_HID structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Defines the raw input data coming from the specified Human Interface Device (HID).

Syntax

```
typedef struct tagRID_DEVICE_INFO_HID {
  DWORD  dwVendorId;
  DWORD  dwProductId;
  DWORD  dwVersionNumber;
  USHORT usUsagePage;
  USHORT usUsage;
} RID_DEVICE_INFO_HID, *PRID_DEVICE_INFO_HID;
```

Members

`dwVendorId`

Type: **DWORD**

The vendor identifier for the HID.

`dwProductId`

Type: **DWORD**

The product identifier for the HID.

`dwVersionNumber`

Type: **DWORD**

The version number for the HID.

`usUsagePage`

Type: **USHORT**

The top-level collection Usage Page for the device.

`usUsage`

Type: **USHORT**

The top-level collection Usage for the device.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]

Header	winuser.h (include Windows.h)

See also

Conceptual

RID_DEVICE_INFO

Raw Input

Reference

RID_DEVICE_INFO_KEYBOARD structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Defines the raw input data coming from the specified keyboard.

Syntax

```
typedef struct tagRID_DEVICE_INFO_KEYBOARD {
  DWORD dwType;
  DWORD dwSubType;
  DWORD dwKeyboardMode;
  DWORD dwNumberOfFunctionKeys;
  DWORD dwNumberOfIndicators;
  DWORD dwNumberOfKeysTotal;
} RID_DEVICE_INFO_KEYBOARD, *PRID_DEVICE_INFO_KEYBOARD;
```

Members

`dwType`

Type: **DWORD**

The type of keyboard. See Remarks.

VALUE	DESCRIPTION
0x4	Enhanced 101- or 102-key keyboards (and compatibles)
0x7	Japanese Keyboard
0x8	Korean Keyboard
0x51	Unknown type or HID keyboard

`dwSubType`

Type: **DWORD**

The vendor-specific subtype of the keyboard. See Remarks.

`dwKeyboardMode`

Type: **DWORD**

The scan code mode. Usually 1, which means that *Scan Code Set 1* is used. See Keyboard Scan Code Specification.

`dwNumberOfFunctionKeys`

Type: **DWORD**

The number of function keys on the keyboard.

`dwNumberOfIndicators`

Type: **DWORD**

The number of LED indicators on the keyboard.

`dwNumberOfKeysTotal`

Type: **DWORD**

The total number of keys on the keyboard.

Remarks

For information about keyboard types, subtypes, scan code modes, and related keyboard layouts, see the documentation in *kbd.h*, *ntdd8042.h* and *ntddkbd.h* headers in Windows SDK, and the Keyboard Layout Samples.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

RID_DEVICE_INFO

Raw Input

Reference

KEYBOARD_ATTRIBUTES structure

RID_DEVICE_INFO_MOUSE structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Defines the raw input data coming from the specified mouse.

Syntax

```
typedef struct tagRID_DEVICE_INFO_MOUSE {
  DWORD dwId;
  DWORD dwNumberOfButtons;
  DWORD dwSampleRate;
  BOOL  fHasHorizontalWheel;
} RID_DEVICE_INFO_MOUSE, *PRID_DEVICE_INFO_MOUSE;
```

Members

`dwId`

Type: **DWORD**

The bitfield of the mouse device identification properties:

VALUE	NTDDMOU.H CONSTANT	DESCRIPTION
0x0080	MOUSE_HID_HARDWARE	HID mouse
0x0100	WHEELMOUSE_HID_HARDWARE	HID wheel mouse
0x8000	HORIZONTAL_WHEEL_PRESENT	Mouse with horizontal wheel

`dwNumberOfButtons`

Type: **DWORD**

The number of buttons for the mouse.

`dwSampleRate`

Type: **DWORD**

The number of data points per second. This information may not be applicable for every mouse device.

`fHasHorizontalWheel`

Type: **BOOL**

TRUE if the mouse has a wheel for horizontal scrolling; otherwise, **FALSE**.

Windows XP: This member is only supported starting with Windows Vista.

Remarks

For the mouse, the Usage Page is 1 and the Usage is 2.

Requirements

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Conceptual

RID_DEVICE_INFO

Raw Input

Reference

SendInput function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Synthesizes keystrokes, mouse motions, and button clicks.

Syntax

```
UINT SendInput(
  [in] UINT    cInputs,
  [in] LPINPUT pInputs,
  [in] int     cbSize
);
```

Parameters

`[in] cInputs`

Type: **UINT**

The number of structures in the *pInputs* array.

`[in] pInputs`

Type: **LPINPUT**

An array of INPUT structures. Each structure represents an event to be inserted into the keyboard or mouse input stream.

`[in] cbSize`

Type: **int**

The size, in bytes, of an INPUT structure. If *cbSize* is not the size of an **INPUT** structure, the function fails.

Return value

Type: **UINT**

The function returns the number of events that it successfully inserted into the keyboard or mouse input stream. If the function returns zero, the input was already blocked by another thread. To get extended error information, call GetLastError.

This function fails when it is blocked by UIPI. Note that neither GetLastError nor the return value will indicate the failure was caused by UIPI blocking.

Remarks

This function is subject to UIPI. Applications are permitted to inject input only into applications that are at an equal or lesser integrity level.

The **SendInput** function inserts the events in the INPUT structures serially into the keyboard or mouse input stream. These events are not interspersed with other keyboard or mouse input events inserted either by the user (with the keyboard or mouse) or by calls to keybd_event, mouse_event, or other calls to **SendInput**.

This function does not reset the keyboard's current state. Any keys that are already pressed when the function is called might interfere with the events that this function generates. To avoid this problem, check the keyboard's state with the GetAsyncKeyState function and correct as necessary.

Because the touch keyboard uses the surrogate macros defined in winnls.h to send input to the system, a listener on the keyboard event hook must decode input originating from the touch keyboard. For more information, see Surrogates and Supplementary Characters.

An accessibility application can use **SendInput** to inject keystrokes corresponding to application launch shortcut keys that are handled by the shell. This functionality is not guaranteed to work for other types of applications.

Example

```
//**********************************************************************
//
// Sends Win + D to toggle to the desktop
//
//**********************************************************************
void ShowDesktop()
{
    OutputString(L"Sending 'Win-D'\r\n");
    INPUT inputs[4] = {};
    ZeroMemory(inputs, sizeof(inputs));

    inputs[0].type = INPUT_KEYBOARD;
    inputs[0].ki.wVk = VK_LWIN;

    inputs[1].type = INPUT_KEYBOARD;
    inputs[1].ki.wVk = VK_D;

    inputs[2].type = INPUT_KEYBOARD;
    inputs[2].ki.wVk = VK_D;
    inputs[2].ki.dwFlags = KEYEVENTF_KEYUP;

    inputs[3].type = INPUT_KEYBOARD;
    inputs[3].ki.wVk = VK_LWIN;
    inputs[3].ki.dwFlags = KEYEVENTF_KEYUP;

    UINT uSent = SendInput(ARRAYSIZE(inputs), inputs, sizeof(INPUT));
    if (uSent != ARRAYSIZE(inputs))
    {
        OutputString(L"SendInput failed: 0x%x\n", HRESULT_FROM_WIN32(GetLastError()));
    }
}
```

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)

Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

INPUT

Keyboard Input

Reference

Surrogates and Supplementary Characters

keybd_event

mouse_event

SetActiveWindow function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Activates a window. The window must be attached to the calling thread's message queue.

Syntax

```
HWND SetActiveWindow(
  [in] HWND hWnd
);
```

Parameters

`[in] hWnd`

Type: **HWND**

A handle to the top-level window to be activated.

Return value

Type: **HWND**

If the function succeeds, the return value is the handle to the window that was previously active.

If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

Remarks

The **SetActiveWindow** function activates a window, but not if the application is in the background. The window will be brought into the foreground (top of Z-Order) if its application is in the foreground when the system activates the window.

If the window identified by the *hWnd* parameter was created by the calling thread, the active window status of the calling thread is set to *hWnd*. Otherwise, the active window status of the calling thread is set to **NULL**.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib

DLL	User32.dll
API set	ext-ms-win-ntuser-window-l1-1-4 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

GetActiveWindow

Keyboard Input

Reference

SetForegroundWindow

WM_ACTIVATE

SetCapture function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Sets the mouse capture to the specified window belonging to the current thread. **SetCapture** captures mouse input either when the mouse is over the capturing window, or when the mouse button was pressed while the mouse was over the capturing window and the button is still down. Only one window at a time can capture the mouse.

If the mouse cursor is over a window created by another thread, the system will direct mouse input to the specified window only if a mouse button is down.

Syntax

```
HWND SetCapture(
  [in] HWND hWnd
);
```

Parameters

`[in] hWnd`

Type: **HWND**

A handle to the window in the current thread that is to capture the mouse.

Return value

Type: **HWND**

The return value is a handle to the window that had previously captured the mouse. If there is no such window, the return value is **NULL**.

Remarks

Only the foreground window can capture the mouse. When a background window attempts to do so, the window receives messages only for mouse events that occur when the cursor hot spot is within the visible portion of the window. Also, even if the foreground window has captured the mouse, the user can still click another window, bringing it to the foreground.

When the window no longer requires all mouse input, the thread that created the window should call the ReleaseCapture function to release the mouse.

This function cannot be used to capture mouse input meant for another process.

When the mouse is captured, menu hotkeys and other keyboard accelerators do not work.

Examples

For an example, see Drawing Lines with the Mouse.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-mouse-l1-1-0 (introduced in Windows 8)

See also

Conceptual

GetCapture

Mouse Input

Reference

ReleaseCapture

WM_CAPTURECHANGED

SetDoubleClickTime function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Sets the double-click time for the mouse. A double-click is a series of two clicks of a mouse button, the second occurring within a specified time after the first. The double-click time is the maximum number of milliseconds that may occur between the first and second clicks of a double-click.

Syntax

```
BOOL SetDoubleClickTime(
  [in] UINT unnamedParam1
);
```

Parameters

`[in] unnamedParam1`

Type: **UINT**

The number of milliseconds that may occur between the first and second clicks of a double-click. If this parameter is set to 0, the system uses the default double-click time of 500 milliseconds. If this parameter value is greater than 5000 milliseconds, the system sets the value to 5000 milliseconds.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

The **SetDoubleClickTime** function alters the double-click time for all windows in the system.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

API set	ext-ms-win-ntuser-mouse-l1-1-1 (introduced in Windows 10, version 10.0.14393)

See also

Conceptual

GetDoubleClickTime

Mouse Input

Reference

SetFocus function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Sets the keyboard focus to the specified window. The window must be attached to the calling thread's message queue.

Syntax

```
HWND SetFocus(
  [in, optional] HWND hWnd
);
```

Parameters

`[in, optional] hWnd`

Type: **HWND**

A handle to the window that will receive the keyboard input. If this parameter is NULL, keystrokes are ignored.

Return value

Type: **HWND**

If the function succeeds, the return value is the handle to the window that previously had the keyboard focus. If the *hWnd* parameter is invalid or the window is not attached to the calling thread's message queue, the return value is NULL. To get extended error information, call GetLastError function.

Extended error ERROR_INVALID_PARAMETER (0x57) means that window is in disabled state.

Remarks

This function sends a WM_KILLFOCUS message to the window that loses the keyboard focus and a WM_SETFOCUS message to the window that receives the keyboard focus. It also activates either the window that receives the focus or the parent of the window that receives the focus.

If a window is active but does not have the focus, any key pressed produces the WM_SYSCHAR, WM_SYSKEYDOWN, or WM_SYSKEYUP message. If the VK_MENU key is also pressed, bit 30 of the *lParam* parameter of the message is set. Otherwise, the messages produced do not have this bit set.

By using the AttachThreadInput function, a thread can attach its input processing to another thread. This allows a thread to call SetFocus to set the keyboard focus to a window attached to another thread's message queue.

Examples

For an example, see Initializing a Dialog Box.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]

Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-window-l1-1-4 (introduced in Windows 10, version 10.0.14393)

See also

AttachThreadInput function, GetFocus function, WM_KILLFOCUS, WM_SETFOCUS, WM_SYSCHAR, WM_SYSKEYDOWN, WM_SYSKEYUP, Keyboard Input

SetKeyboardState function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Copies an array of keyboard key states into the calling thread's keyboard input-state table. This is the same table accessed by the GetKeyboardState and GetKeyState functions. Changes made to this table do not affect keyboard input to any other thread.

Syntax

```
BOOL SetKeyboardState(
  [in] LPBYTE lpKeyState
);
```

Parameters

`[in] lpKeyState`

Type: **LPBYTE**

A pointer to a 256-byte array that contains keyboard key states.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Remarks

Because the **SetKeyboardState** function alters the input state of the calling thread and not the global input state of the system, an application cannot use **SetKeyboardState** to set the NUM LOCK, CAPS LOCK, or SCROLL LOCK (or the Japanese KANA) indicator lights on the keyboard. These can be set or cleared using SendInput to simulate keystrokes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib

DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

GetKeyState

GetKeyboardState

Keyboard Input

MapVirtualKey

Reference

SendInput

keybd_event

SwapMouseButton function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Reverses or restores the meaning of the left and right mouse buttons.

Syntax

```
BOOL SwapMouseButton(
  [in] BOOL fSwap
);
```

Parameters

`[in] fSwap`

Type: **BOOL**

If this parameter is **TRUE**, the left button generates right-button messages and the right button generates left-button messages. If this parameter is **FALSE**, the buttons are restored to their original meanings.

Return value

Type: **BOOL**

If the meaning of the mouse buttons was reversed previously, before the function was called, the return value is nonzero.

If the meaning of the mouse buttons was not reversed, the return value is zero.

Remarks

Button swapping is provided as a convenience to people who use the mouse with their left hands. The **SwapMouseButton** function is usually called by Control Panel only. Although an application is free to call the function, the mouse is a shared resource and reversing the meaning of its buttons affects all applications.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Mouse Input

Reference

SetDoubleClickTime

ToAscii function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Translates the specified virtual-key code and keyboard state to the corresponding character or characters. The function translates the code using the input language and physical keyboard layout identified by the keyboard layout handle.

To specify a handle to the keyboard layout to use to translate the specified code, use the ToAsciiEx function.

Syntax

```
int ToAscii(
  [in]           UINT       uVirtKey,
  [in]           UINT       uScanCode,
  [in, optional] const BYTE *lpKeyState,
  [out]          LPWORD     lpChar,
  [in]           UINT       uFlags
);
```

Parameters

`[in] uVirtKey`

Type: **UINT**

The virtual-key code to be translated. See Virtual-Key Codes.

`[in] uScanCode`

Type: **UINT**

The hardware scan code of the key to be translated. The high-order bit of this value is set if the key is up (not pressed).

`[in, optional] lpKeyState`

Type: **const BYTE***

A pointer to a 256-byte array that contains the current keyboard state. Each element (byte) in the array contains the state of one key. If the high-order bit of a byte is set, the key is down (pressed).

The low bit, if set, indicates that the key is toggled on. In this function, only the toggle bit of the CAPS LOCK key is relevant. The toggle state of the NUM LOCK and SCROLL LOCK keys is ignored.

`[out] lpChar`

Type: **LPWORD**

The buffer that receives the translated character or characters.

`[in] uFlags`

Type: **UINT**

This parameter must be 1 if a menu is active, or 0 otherwise.

Return value

Type: **int**

If the specified key is a dead key, the return value is negative. Otherwise, it is one of the following values.

RETURN VALUE	DESCRIPTION
0	The specified virtual key has no translation for the current state of the keyboard.
1	One character was copied to the buffer.
2	Two characters were copied to the buffer. This usually happens when a dead-key character (accent or diacritic) stored in the keyboard layout cannot be composed with the specified virtual key to form a single character.

Remarks

The parameters supplied to the **ToAscii** function might not be sufficient to translate the virtual-key code, because a previous dead key is stored in the keyboard layout.

Typically, **ToAscii** performs the translation based on the virtual-key code. In some cases, however, bit 15 of the *uScanCode* parameter may be used to distinguish between a key press and a key release. The scan code is used for translating ALT+ *number key* combinations.

Although NUM LOCK is a toggle key that affects keyboard behavior, **ToAscii** ignores the toggle setting (the low bit) of *lpKeyState* (**VK_NUMLOCK**) because the *uVirtKey* parameter alone is sufficient to distinguish the cursor movement keys (**VK_HOME**, **VK_INSERT**, and so on) from the numeric keys (**VK_DECIMAL**, **VK_NUMPAD0** - **VK_NUMPAD9**).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Keyboard Input

OemKeyScan

Reference

ToAsciiEx

ToUnicode

VkKeyScan

ToAsciiEx function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Translates the specified virtual-key code and keyboard state to the corresponding character or characters. The function translates the code using the input language and physical keyboard layout identified by the input locale identifier.

Syntax

```
int ToAsciiEx(
  [in]           UINT       uVirtKey,
  [in]           UINT       uScanCode,
  [in, optional] const BYTE *lpKeyState,
  [out]          LPWORD     lpChar,
  [in]           UINT       uFlags,
  [in, optional] HKL        dwhkl
);
```

Parameters

`[in] uVirtKey`

Type: **UINT**

The virtual-key code to be translated. See Virtual-Key Codes.

`[in] uScanCode`

Type: **UINT**

The hardware scan code of the key to be translated. The high-order bit of this value is set if the key is up (not pressed).

`[in, optional] lpKeyState`

Type: **const BYTE***

A pointer to a 256-byte array that contains the current keyboard state. Each element (byte) in the array contains the state of one key. If the high-order bit of a byte is set, the key is down (pressed).

The low bit, if set, indicates that the key is toggled on. In this function, only the toggle bit of the CAPS LOCK key is relevant. The toggle state of the NUM LOCK and SCOLL LOCK keys is ignored.

`[out] lpChar`

Type: **LPWORD**

A pointer to the buffer that receives the translated character or characters.

`[in] uFlags`

Type: **UINT**

This parameter must be 1 if a menu is active, zero otherwise.

`[in, optional] dwhkl`

Type: **HKL**

Input locale identifier to use to translate the code. This parameter can be any input locale identifier previously returned by the LoadKeyboardLayout function.

Return value

Type: **int**

If the specified key is a dead key, the return value is negative. Otherwise, it is one of the following values.

RETURN VALUE	DESCRIPTION
0	The specified virtual key has no translation for the current state of the keyboard.
1	One character was copied to the buffer.
2	Two characters were copied to the buffer. This usually happens when a dead-key character (accent or diacritic) stored in the keyboard layout cannot be composed with the specified virtual key to form a single character.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

The parameters supplied to the **ToAsciiEx** function might not be sufficient to translate the virtual-key code, because a previous dead key is stored in the keyboard layout.

Typically, **ToAsciiEx** performs the translation based on the virtual-key code. In some cases, however, bit 15 of the *uScanCode* parameter may be used to distinguish between a key press and a key release. The scan code is used for translating ALT+number key combinations.

Although NUM LOCK is a toggle key that affects keyboard behavior, **ToAsciiEx** ignores the toggle setting (the low bit) of *lpKeyState* (**VK_NUMLOCK**) because the *uVirtKey* parameter alone is sufficient to distinguish the cursor movement keys (**VK_HOME**, **VK_INSERT**, and so on) from the numeric keys (**VK_DECIMAL**, **VK_NUMPAD0** - **VK_NUMPAD9**).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib

DLL	User32.dll

See also

Conceptual

Keyboard Input

LoadKeyboardLayout

MapVirtualKeyEx

Reference

ToUnicodeEx

VkKeyScan

TOOLTIP_DISMISS_FLAGS enumeration (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Defines constants that indicate whether a window is registered or unregistered to receive tooltip dismiss notifications.

Syntax

```
typedef enum  {
  TDF_REGISTER,
  TDF_UNREGISTER
} TOOLTIP_DISMISS_FLAGS;
```

Constants

`TDF_REGISTER` The window is registered to receive tooltip dismiss notifications.
`TDF_UNREGISTER` The window is unregistered from receiving tooltip dismiss notifications.

Remarks

This enumeration is used by the RegisterForTooltipDismissNotification function.

Requirements

Minimum supported client	Windows 11 Build 22621
Header	winuser.h

See also

RegisterForTooltipDismissNotification, Tooltip

ToUnicode function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Translates the specified virtual-key code and keyboard state to the corresponding Unicode character or characters.

To specify a handle to the keyboard layout to use to translate the specified code, use the ToUnicodeEx function.

Syntax

```
int ToUnicode(
  [in]           UINT       wVirtKey,
  [in]           UINT       wScanCode,
  [in, optional] const BYTE *lpKeyState,
  [out]          LPWSTR     pwszBuff,
  [in]           int        cchBuff,
  [in]           UINT       wFlags
);
```

Parameters

`[in] wVirtKey`

Type: **UINT**

The virtual-key code to be translated. See Virtual-Key Codes.

`[in] wScanCode`

Type: **UINT**

The hardware scan code of the key to be translated. The high-order bit of this value is set if the key is up.

`[in, optional] lpKeyState`

Type: **const BYTE***

A pointer to a 256-byte array that contains the current keyboard state. Each element (byte) in the array contains the state of one key. If the high-order bit of a byte is set, the key is down.

`[out] pwszBuff`

Type: **LPWSTR**

The buffer that receives the translated Unicode character or characters. However, this buffer may be returned without being null-terminated even though the variable name suggests that it is null-terminated.

`[in] cchBuff`

Type: **int**

The size, in characters, of the buffer pointed to by the *pwszBuff* parameter.

`[in] wFlags`

Type: **UINT**

The behavior of the function.

If bit 0 is set, a menu is active.

If bit 2 is set, keyboard state is not changed (Windows 10, version 1607 and newer)

All other bits (through 31) are reserved.

Return value

Type: **int**

The function returns one of the following values.

RETURN VALUE	DESCRIPTION
-1	The specified virtual key is a dead-key character (accent or diacritic). This value is returned regardless of the keyboard layout, even if several characters have been typed and are stored in the keyboard state. If possible, even with Unicode keyboard layouts, the function has written a spacing version of the dead-key character to the buffer specified by *pwszBuff* . For example, the function writes the character SPACING ACUTE (0x00B4), rather than the character NON_SPACING ACUTE (0x0301).
0	The specified virtual key has no translation for the current state of the keyboard. Nothing was written to the buffer specified by *pwszBuff*.
1	One character was written to the buffer specified by *pwszBuff*.
2 ≤ *value*	Two or more characters were written to the buffer specified by *pwszBuff*. The most common cause for this is that a dead-key character (accent or diacritic) stored in the keyboard layout could not be combined with the specified virtual key to form a single character. However, the buffer may contain more characters than the return value specifies. When this happens, any extra characters are invalid and should be ignored.

Remarks

The parameters supplied to the **ToUnicode** function might not be sufficient to translate the virtual-key code because a previous dead key is stored in the keyboard layout.

Typically, **ToUnicode** performs the translation based on the virtual-key code. In some cases, however, bit 15 of the *wScanCode* parameter can be used to distinguish between a key press and a key release.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]

Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Keyboard Input

Reference

ToAscii

ToUnicodeEx

VkKeyScan

ToUnicodeEx function (winuser.h)

7/6/2022 • 3 minutes to read • Edit Online

Translates the specified virtual-key code and keyboard state to the corresponding Unicode character or characters.

Syntax

```
int ToUnicodeEx(
  [in]           UINT       wVirtKey,
  [in]           UINT       wScanCode,
  [in]           const BYTE *lpKeyState,
  [out]          LPWSTR     pwszBuff,
  [in]           int        cchBuff,
  [in]           UINT       wFlags,
  [in, optional] HKL        dwhkl
);
```

Parameters

`[in] wVirtKey`

Type: **UINT**

The virtual-key code to be translated. See Virtual-Key Codes.

`[in] wScanCode`

Type: **UINT**

The hardware scan code of the key to be translated. The high-order bit of this value is set if the key is up.

`[in] lpKeyState`

Type: **const BYTE***

A pointer to a 256-byte array that contains the current keyboard state. Each element (byte) in the array contains the state of one key. If the high-order bit of a byte is set, the key is down.

`[out] pwszBuff`

Type: **LPWSTR**

The buffer that receives the translated Unicode character or characters. However, this buffer may be returned without being null-terminated even though the variable name suggests that it is null-terminated.

`[in] cchBuff`

Type: **int**

The size, in characters, of the buffer pointed to by the *pwszBuff* parameter.

`[in] wFlags`

Type: **UINT**

The behavior of the function.

If bit 0 is set, a menu is active.

If bit 2 is set, keyboard state is not changed (Windows 10, version 1607 and newer)

All other bits (through 31) are reserved.

`[in, optional] dwhkl`

Type: **HKL**

The input locale identifier used to translate the specified code. This parameter can be any input locale identifier previously returned by the LoadKeyboardLayout function.

Return value

Type: **int**

The function returns one of the following values.

RETURN VALUE	DESCRIPTION
-1	The specified virtual key is a dead-key character (accent or diacritic). This value is returned regardless of the keyboard layout, even if several characters have been typed and are stored in the keyboard state. If possible, even with Unicode keyboard layouts, the function has written a spacing version of the dead-key character to the buffer specified by *pwszBuff* . For example, the function writes the character SPACING ACUTE (0x00B4), rather than the character NON_SPACING ACUTE (0x0301).
0	The specified virtual key has no translation for the current state of the keyboard. Nothing was written to the buffer specified by *pwszBuff*.
1	One character was written to the buffer specified by *pwszBuff*.
2 ≤ *value*	Two or more characters were written to the buffer specified by *pwszBuff*. The most common cause for this is that a dead-key character (accent or diacritic) stored in the keyboard layout could not be combined with the specified virtual key to form a single character. However, the buffer may contain more characters than the return value specifies. When this happens, any extra characters are invalid and should be ignored.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

The parameters supplied to the **ToUnicodeEx** function might not be sufficient to translate the virtual-key code because a previous dead key is stored in the keyboard layout.

Typically, **ToUnicodeEx** performs the translation based on the virtual-key code. In some cases, however, bit 15 of the *wScanCode* parameter can be used to distinguish between a key press and a key release.

As **ToUnicodeEx** translates the virtual-key code, it also changes the state of the kernel-mode keyboard buffer. This state-change affects dead keys, ligatures, alt+numpad key entry, and so on. It might also cause undesired side-effects if used in conjunction with TranslateMessage (which also changes the state of the kernel-mode keyboard buffer).

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Keyboard Input

LoadKeyboardLayout

Reference

ToAsciiEx

VkKeyScan

TrackMouseEvent function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Posts messages when the mouse pointer leaves a window or hovers over a window for a specified amount of time.

> **Note** The _TrackMouseEvent function calls **TrackMouseEvent** if it exists, otherwise **_TrackMouseEvent** emulates **TrackMouseEvent**.

Syntax

```
BOOL TrackMouseEvent(
  [in, out] LPTRACKMOUSEEVENT lpEventTrack
);
```

Parameters

`[in, out] lpEventTrack`

Type: **LPTRACKMOUSEEVENT**

A pointer to a TRACKMOUSEEVENT structure that contains tracking information.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero .

If the function fails, return value is zero. To get extended error information, call GetLastError.

Remarks

The mouse pointer is considered to be hovering when it stays within a specified rectangle for a specified period of time. Call SystemParametersInfo. and use the values **SPI_GETMOUSEHOVERWIDTH**, **SPI_GETMOUSEHOVERHEIGHT**, and **SPI_GETMOUSEHOVERTIME** to retrieve the size of the rectangle and the time.

The function can post the following messages.

MESSAGE	DESCRIPTION
WM_NCMOUSEHOVER	The same meaning as WM_MOUSEHOVER except this is for the nonclient area of the window.
WM_NCMOUSELEAVE	The same meaning as WM_MOUSELEAVE except this is for the nonclient area of the window.

WM_MOUSEHOVER	The mouse hovered over the client area of the window for the period of time specified in a prior call to **TrackMouseEvent**. Hover tracking stops when this message is generated. The application must call **TrackMouseEvent** again if it requires further tracking of mouse hover behavior.
WM_MOUSELEAVE	The mouse left the client area of the window specified in a prior call to **TrackMouseEvent**. All tracking requested by **TrackMouseEvent** is canceled when this message is generated. The application must call **TrackMouseEvent** when the mouse reenters its window if it requires further tracking of mouse hover behavior.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-mouse-l1-1-0 (introduced in Windows 8)

See also

Conceptual

Mouse Input

Other Resources

Reference

SystemParametersInfo

TRACKMOUSEEVENT

_TrackMouseEvent

TRACKMOUSEEVENT structure (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Used by the TrackMouseEvent function to track when the mouse pointer leaves a window or hovers over a window for a specified amount of time.

Syntax

```
typedef struct tagTRACKMOUSEEVENT {
  DWORD cbSize;
  DWORD dwFlags;
  HWND  hwndTrack;
  DWORD dwHoverTime;
} TRACKMOUSEEVENT, *LPTRACKMOUSEEVENT;
```

Members

`cbSize`

Type: **DWORD**

The size of the **TRACKMOUSEEVENT** structure, in bytes.

`dwFlags`

Type: **DWORD**

The services requested. This member can be a combination of the following values.

VALUE	MEANING
TME_CANCEL 0x80000000	The caller wants to cancel a prior tracking request. The caller should also specify the type of tracking that it wants to cancel. For example, to cancel hover tracking, the caller must pass the **TME_CANCEL** and **TME_HOVER** flags.
TME_HOVER 0x00000001	The caller wants hover notification. Notification is delivered as a WM_MOUSEHOVER message. If the caller requests hover tracking while hover tracking is already active, the hover timer will be reset. This flag is ignored if the mouse pointer is not over the specified window or area.
TME_LEAVE 0x00000002	The caller wants leave notification. Notification is delivered as a WM_MOUSELEAVE message. If the mouse is not over the specified window or area, a leave notification is generated immediately and no further tracking is performed.
TME_NONCLIENT 0x00000010	The caller wants hover and leave notification for the nonclient areas. Notification is delivered as WM_NCMOUSEHOVER and WM_NCMOUSELEAVE messages.

TME_QUERY 0x40000000	The function fills in the structure instead of treating it as a tracking request. The structure is filled such that had that structure been passed to TrackMouseEvent, it would generate the current tracking. The only anomaly is that the hover time-out returned is always the actual time-out and not **HOVER_DEFAULT**, if **HOVER_DEFAULT** was specified during the original **TrackMouseEvent** request.

`hwndTrack`

Type: **HWND**

A handle to the window to track.

`dwHoverTime`

Type: **DWORD**

The hover time-out (if **TME_HOVER** was specified in **dwFlags**), in milliseconds. Can be **HOVER_DEFAULT**, which means to use the system default hover time-out.

Remarks

The system default hover time-out is initially the menu drop-down time, which is 400 milliseconds. You can call SystemParametersInfo and use **SPI_GETMOUSEHOVERTIME** to retrieve the default hover time-out.

The system default hover rectangle is the same as the double-click rectangle. You can call SystemParametersInfo and use **SPI_GETMOUSEHOVERWIDTH** and **SPI_GETMOUSEHOVERHEIGHT** to retrieve the size of the rectangle within which the mouse pointer has to stay for TrackMouseEvent to generate a WM_MOUSEHOVER message.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Header	winuser.h (include Windows.h)

See also

Mouse Input

UnloadKeyboardLayout function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Unloads an input locale identifier (formerly called a keyboard layout).

Syntax

```
BOOL UnloadKeyboardLayout(
  [in] HKL hkl
);
```

Parameters

`[in] hkl`

Type: **HKL**

The input locale identifier to be unloaded.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. The function can fail for the following reasons:

- An invalid input locale identifier was passed.
- The input locale identifier was preloaded.
- The input locale identifier is in use.

To get extended error information, call GetLastError.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

UnloadKeyboardLayout cannot unload the system default input locale identifier if it is the only keyboard layout loaded. You must first load another input locale identifier before unloading the default input locale identifier.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows

Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

ActivateKeyboardLayout

Conceptual

GetKeyboardLayoutName

Keyboard Input

LoadKeyboardLayout

Reference

UnregisterHotKey function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Frees a hot key previously registered by the calling thread.

Syntax

```
BOOL UnregisterHotKey(
  [in, optional] HWND hWnd,
  [in]           int  id
);
```

Parameters

`[in, optional] hWnd`

Type: **HWND**

A handle to the window associated with the hot key to be freed. This parameter should be **NULL** if the hot key is not associated with a window.

`[in] id`

Type: **int**

The identifier of the hot key to be freed.

Return value

Type: **BOOL**

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call GetLastError.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

Keyboard Input

Reference

RegisterHotKey

WM_HOTKEY

VkKeyScanA function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

[This function has been superseded by the VkKeyScanEx function. You can still use **VkKeyScan**, however, if you do not need to specify a keyboard layout.]

Translates a character to the corresponding virtual-key code and shift state for the current keyboard.

Syntax

```
SHORT VkKeyScanA(
  [in] CHAR ch
);
```

Parameters

`[in] ch`

Type: **TCHAR**

The character to be translated into a virtual-key code.

Return value

Type: **SHORT**

If the function succeeds, the low-order byte of the return value contains the virtual-key code and the high-order byte contains the shift state, which can be a combination of the following flag bits.

RETURN VALUE	DESCRIPTION
1	Either SHIFT key is pressed.
2	Either CTRL key is pressed.
4	Either ALT key is pressed.
8	The Hankaku key is pressed
16	Reserved (defined by the keyboard layout driver).
32	Reserved (defined by the keyboard layout driver).

If the function finds no key that translates to the passed character code, both the low-order and high-order bytes contain –1.

Remarks

For keyboard layouts that use the right-hand ALT key as a shift key (for example, the French keyboard layout), the shift state is represented by the value 6, because the right-hand ALT key is converted internally into CTRL+ALT.

Translations for the numeric keypad (**VK_NUMPAD0** through **VK_DIVIDE**) are ignored. This function is intended to translate characters into keystrokes from the main keyboard section only. For example, the character "7" is translated into VK_7, not VK_NUMPAD7.

VkKeyScan is used by applications that send characters by using the WM_KEYUP and WM_KEYDOWN messages.

NOTE

The winuser.h header defines VkKeyScan as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

GetKeyNameText

GetKeyState

GetKeyboardState

Keyboard Input

Reference

SetKeyboardState

VkKeyScanEx

WM_KEYDOWN

WM_KEYUP

VkKeyScanExA function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Translates a character to the corresponding virtual-key code and shift state. The function translates the character using the input language and physical keyboard layout identified by the input locale identifier.

Syntax

```
SHORT VkKeyScanExA(
  [in] CHAR ch,
  [in] HKL  dwhkl
);
```

Parameters

`[in] ch`

Type: **TCHAR**

The character to be translated into a virtual-key code.

`[in] dwhkl`

Type: **HKL**

Input locale identifier used to translate the character. This parameter can be any input locale identifier previously returned by the LoadKeyboardLayout function.

Return value

Type: **SHORT**

If the function succeeds, the low-order byte of the return value contains the virtual-key code and the high-order byte contains the shift state, which can be a combination of the following flag bits.

RETURN VALUE	DESCRIPTION
1	Either SHIFT key is pressed.
2	Either CTRL key is pressed.
4	Either ALT key is pressed.
8	The Hankaku key is pressed

16	Reserved (defined by the keyboard layout driver).
32	Reserved (defined by the keyboard layout driver).

If the function finds no key that translates to the passed character code, both the low-order and high-order bytes contain –1.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

For keyboard layouts that use the right-hand ALT key as a shift key (for example, the French keyboard layout), the shift state is represented by the value 6, because the right-hand ALT key is converted internally into CTRL+ALT.

Translations for the numeric keypad (VK_NUMPAD0 through VK_DIVIDE) are ignored. This function is intended to translate characters into keystrokes from the main keyboard section only. For example, the character "7" is translated into VK_7, not VK_NUMPAD7.

VkKeyScanEx is used by applications that send characters by using the WM_KEYUP and WM_KEYDOWN messages.

NOTE
The winuser.h header defines VkKeyScanEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

GetKeyNameText

GetKeyState

GetKeyboardState

Keyboard Input

LoadKeyboardLayout

Reference

SetKeyboardState

ToAsciiEx

VkKeyScanExW function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

Translates a character to the corresponding virtual-key code and shift state. The function translates the character using the input language and physical keyboard layout identified by the input locale identifier.

Syntax

```
SHORT VkKeyScanExW(
  [in] WCHAR ch,
  [in] HKL   dwhkl
);
```

Parameters

`[in] ch`

Type: **TCHAR**

The character to be translated into a virtual-key code.

`[in] dwhkl`

Type: **HKL**

Input locale identifier used to translate the character. This parameter can be any input locale identifier previously returned by the LoadKeyboardLayout function.

Return value

Type: **SHORT**

If the function succeeds, the low-order byte of the return value contains the virtual-key code and the high-order byte contains the shift state, which can be a combination of the following flag bits.

RETURN VALUE	DESCRIPTION
1	Either SHIFT key is pressed.
2	Either CTRL key is pressed.
4	Either ALT key is pressed.
8	The Hankaku key is pressed

16	Reserved (defined by the keyboard layout driver).
32	Reserved (defined by the keyboard layout driver).

If the function finds no key that translates to the passed character code, both the low-order and high-order bytes contain –1.

Remarks

The input locale identifier is a broader concept than a keyboard layout, since it can also encompass a speech-to-text converter, an Input Method Editor (IME), or any other form of input.

For keyboard layouts that use the right-hand ALT key as a shift key (for example, the French keyboard layout), the shift state is represented by the value 6, because the right-hand ALT key is converted internally into CTRL+ALT.

Translations for the numeric keypad (VK_NUMPAD0 through VK_DIVIDE) are ignored. This function is intended to translate characters into keystrokes from the main keyboard section only. For example, the character "7" is translated into VK_7, not VK_NUMPAD7.

VkKeyScanEx is used by applications that send characters by using the WM_KEYUP and WM_KEYDOWN messages.

> **NOTE**
> The winuser.h header defines VkKeyScanEx as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

GetKeyNameText

GetKeyState

GetKeyboardState

Keyboard Input

LoadKeyboardLayout

Reference

SetKeyboardState

ToAsciiEx

VkKeyScanW function (winuser.h)

7/6/2022 • 2 minutes to read • Edit Online

[This function has been superseded by the VkKeyScanEx function. You can still use **VkKeyScan**, however, if you do not need to specify a keyboard layout.]

Translates a character to the corresponding virtual-key code and shift state for the current keyboard.

Syntax

```
SHORT VkKeyScanW(
  [in] WCHAR ch
);
```

Parameters

`[in] ch`

Type: **TCHAR**

The character to be translated into a virtual-key code.

Return value

Type: **SHORT**

If the function succeeds, the low-order byte of the return value contains the virtual-key code and the high-order byte contains the shift state, which can be a combination of the following flag bits.

RETURN VALUE	DESCRIPTION
1	Either SHIFT key is pressed.
2	Either CTRL key is pressed.
4	Either ALT key is pressed.
8	The Hankaku key is pressed
16	Reserved (defined by the keyboard layout driver).
32	Reserved (defined by the keyboard layout driver).

If the function finds no key that translates to the passed character code, both the low-order and high-order bytes contain –1.

Remarks

For keyboard layouts that use the right-hand ALT key as a shift key (for example, the French keyboard layout), the shift state is represented by the value 6, because the right-hand ALT key is converted internally into CTRL+ALT.

Translations for the numeric keypad (**VK_NUMPAD0** through **VK_DIVIDE**) are ignored. This function is intended to translate characters into keystrokes from the main keyboard section only. For example, the character "7" is translated into VK_7, not VK_NUMPAD7.

VkKeyScan is used by applications that send characters by using the WM_KEYUP and WM_KEYDOWN messages.

NOTE

The winuser.h header defines VkKeyScan as an alias which automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Minimum supported client	Windows 2000 Professional [desktop apps only]
Minimum supported server	Windows 2000 Server [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Conceptual

GetAsyncKeyState

GetKeyNameText

GetKeyState

GetKeyboardState

Keyboard Input

Reference

SetKeyboardState

VkKeyScanEx

WM_KEYDOWN

WM_KEYUP