Contents

USB driver reference

2/1/2021 • 7 minutes to read • Edit Online

Overview of the USB driver reference technology.

To develop USB driver reference, you need these headers:

- genericusbfnioctl.h
- usbuser.h
- winusb.h
- winusbio.h

For programming guidance for this technology, see:

- USB driver reference

IOCTLs

TITLE	DESCRIPTION
IOCTL_GENERICUSBFN_ACTIVATE_USB_BUS	This I/O control code (IOCTL) is sent by a user-mode service or application to notify GenericUSBFn.sys to activate the Universal Serial Bus (USB). Once activated, the bus is prepared to process bus events and handle traffic.
IOCTL_GENERICUSBFN_BUS_EVENT_NOTIFICATION	This I/O control code (IOCTL) is sent by a user-mode service or application to register for Universal Serial Bus (USB) event.
IOCTL_GENERICUSBFN_CONTROL_STATUS_HANDSHAKE_IN	This I/O control code (IOCTL) is sent by a user mode service or application to request a zero-length control status handshake on endpoint 0 in the IN direction.
IOCTL_GENERICUSBFN_CONTROL_STATUS_HANDSHAKE_OUT	This I/O control code (IOCTL) is sent by a user-mode service or application to complete a zero-length control status handshake on endpoint 0 in the OUT direction.
IOCTL_GENERICUSBFN_DEACTIVATE_USB_BUS	This IOCTL code is nevtot supported.
IOCTL_GENERICUSBFN_GET_CLASS_INFO	This I/O control code (IOCTL) is sent by the user-mode service or application to retrieve information about a device's available pipes as configured in the registry.
IOCTL_GENERICUSBFN_GET_CLASS_INFO_EX	This I/O control code (IOCTL) is sent by a user-mode service or application to retrieve information about a device's available pipes as configured in the registry.
IOCTL_GENERICUSBFN_GET_INTERFACE_DESCRIPTOR_SET	This I/O control code (IOCTL) is sent by a user-mode service or application to get the entire interface descriptor set for a function on the device.This IOCTL request does not retrieve the interface descriptor set for the entire device.Universal Serial Bus (USB) interface descriptor set for a function on the device.

TITLE	DESCRIPTION
IOCTL_GENERICUSBFN_GET_PIPE_STATE	This I/O control code (IOCTL) is sent by a user-mode service or application to get the state of the specified Universal Serial Bus (USB) pipe.
IOCTL_GENERICUSBFN_REGISTER_USB_STRING	This I/O control code (IOCTL) is sent by a user-mode service or application to register a string descriptor.Universal Serial Bus (USB) string descriptor.
IOCTL_GENERICUSBFN_SET_PIPE_STATE	This I/O control code (IOCTL) is sent by a user-mode service or application to set the state of the specified Universal Serial Bus (USB) pipe.
IOCTL_GENERICUSBFN_TRANSFER_IN	This I/O control code (IOCTL) is sent by a user-mode service or application to issue an IN direction transfer on the endpoint that corresponds to the specified pipe ID in the input buffer.
IOCTL_GENERICUSBFN_TRANSFER_IN_APPEND_ZERO_PKT	This I/O control code (IOCTL) is sent by a user-mode service or application to issue an IN direction transfer on the endpoint that corresponds to the specified pipe ID in the input buffer.
IOCTL_GENERICUSBFN_TRANSFER_OUT	This I/O control code (IOCTL) is sent by a user-mode service or application to issue an OUT direction transfer on the endpoint that corresponds to the specified pipe ID in the input buffer.
IOCTL_GET_HCD_DRIVERKEY_NAME	The IOCTL_GET_HCD_DRIVERKEY_NAME I/O control request retrieves the driver key name in the registry for a USB host controller driver.
IOCTL_USB_DIAGNOSTIC_MODE_OFF	The IOCTL_USB_DIAGNOSTIC_MODE_OFF I/O control has been deprecated. Do not use.
IOCTL_USB_DIAGNOSTIC_MODE_ON	The IOCTL_USB_DIAGNOSTIC_MODE_ON I/O control has been deprecated. Do not use.
IOCTL_USB_GET_ROOT_HUB_NAME	The IOCTL_USB_GET_ROOT_HUB_NAME I/O control request is used with the USB_ROOT_HUB_NAME structure to retrieve the symbolic link name of the root hub.IOCTL_USB_GET_ROOT_HUB_NAME is a user-mode I/O control request.
IOCTL_USB_USER_REQUEST	The IOCTL_USB_USER_REQUEST I/O control request is available to both user-mode applications and kernel-mode drivers.

Enumerations

TITLE	DESCRIPTION
USB_USER_ERROR_CODE	The USB_USER_ERROR_CODE enumeration lists the error codes that a USB user-mode request reports when it fails.

TITLE	DESCRIPTION
WDMUSB_POWER_STATE	The WDMUSB_POWER_STATE enumeration indicates the power state of a host controller or root hub.

Functions

TITLE	DESCRIPTION
WinUsb_AbortPipe	The WinUsb_AbortPipe function aborts all of the pending transfers for a pipe. This is a synchronous operation.
WinUsb_ControlTransfer	The WinUsb_ControlTransfer function transmits control data over a default control endpoint.
WinUsb_FlushPipe	The WinUsb_FlushPipe function discards any data that is cached in a pipe. This is a synchronous operation.
WinUsb_Free	The WinUsb_Free function releases all of the resources that WinUsb_Initialize allocated. This is a synchronous operation.
WinUsb_GetAdjustedFrameNumber	The WinUsb_GetAdjustedFrameNumber function computes what the current USB frame number should be based on the frame number value and timestamp.
WinUsb_GetAssociatedInterface	The WinUsb_GetAssociatedInterface function retrieves a handle for an associated interface. This is a synchronous operation.
WinUsb_GetCurrentAlternateSetting	The WinUsb_GetCurrentAlternateSetting function gets the current alternate interface setting for an interface. This is a synchronous operation.
WinUsb_GetCurrentFrameNumber	The WinUsb_GetCurrentFrameNumber function gets the current frame number for the bus.
WinUsb_GetCurrentFrameNumberAndQpc	The WinUsb_GetCurrentFrameNumberAndQpc function retrieves the system query performance counter (QPC) value synchronized with the frame and microframe.
WinUsb_GetDescriptor	The WinUsb_GetDescriptor function returns the requested descriptor. This is a synchronous operation.
WinUsb_GetOverlappedResult	The WinUsb_GetOverlappedResult function retrieves the results of an overlapped operation on the specified file.
WinUsb_GetPipePolicy	The WinUsb_GetPipePolicy function retrieves the policy for a specific pipe associated with an endpoint on the device. This is a synchronous operation.
WinUsb_GetPowerPolicy	The WinUsb_GetPowerPolicy function retrieves the power policy for a device. This is a synchronous operation.
WinUsb_Initialize	The WinUsb_Initialize function creates a WinUSB handle for the device specified by a file handle.

TITLE	DESCRIPTION
WinUsb_QueryDeviceInformation	The WinUsb_QueryDeviceInformation function gets information about the physical device that is associated with a WinUSB interface handle.
WinUsb_QueryInterfaceSettings	The WinUsb_QueryInterfaceSettings function retrieves the interface descriptor for the specified alternate interface settings for a particular interface handle.
WinUsb_QueryPipe	The WinUsb_QueryPipe function retrieves information about the specified endpoint and the associated pipe for an interface.
WinUsb_QueryPipeEx	The WinUsb_QueryPipeEx function retrieves extended information about the specified endpoint and the associated pipe for an interface.
WinUsb_ReadIsochPipe	The WinUsb_ReadIsochPipe function reads data from an isochronous OUT endpoint.
WinUsb_ReadIsochPipeAsap	The WinUsb_ReadIsochPipeAsap function submits a request that reads data from an isochronous OUT endpoint.
WinUsb_ReadPipe	The WinUsb_ReadPipe function reads data from the specified pipe.
WinUsb_RegisterIsochBuffer	The WinUsb_RegisterIsochBuffer function registers a buffer to be used for isochronous transfers.
WinUsb_ResetPipe	The WinUsb_ResetPipe function resets the data toggle and clears the stall condition on a pipe.
WinUsb_SetCurrentAlternateSetting	The WinUsb_SetCurrentAlternateSetting function sets the alternate setting of an interface.
WinUsb_SetPipePolicy	The WinUsb_SetPipePolicy function sets the policy for a specific pipe associated with an endpoint on the device. This is a synchronous operation.
WinUsb_SetPowerPolicy	The WinUsb_SetPowerPolicy function sets the power policy for a device.
WinUsb_StartTrackingForTimeSync	The WinUsb_StartTrackingForTimeSync function starts the time synchronization feature in the USB driver stack that gets the associated system QPC time for USB bus frames and microframes.
WinUsb_StopTrackingForTimeSync	The WinUsb_StopTrackingForTimeSync function tops the time synchronization feature in the USB driver stack that gets the associated system QPC time for USB bus frames and microframes.
WinUsb_UnregisterIsochBuffer	The WinUsb_UnregisterIsochBuffer function releases all of the resources that WinUsb_RegisterIsochBuffer allocated for isochronous transfers. This is a synchronous operation.

TITLE	DESCRIPTION
WinUsb_WriteIsochPipe	The WinUsb_WriteIsochPipe function writes the contents of a caller-supplied buffer to an isochronous OUT endpoint, starting on a specified frame number.
WinUsb_WriteIsochPipeAsap	The WinUsb_WriteIsochPipeAsap submits a request for writing the contents of a buffer to an isochronous OUT endpoint.
WinUsb_WritePipe	The WinUsb_WritePipe function writes data to a pipe.

Structures

TITLE	DESCRIPTION
USB_BANDWIDTH_INFO	The USB_BANDWIDTH_INFO structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve information about the allocated bandwidth.
USB_BUS_STATISTICS_0	The USB_BUS_STATISTICS_0 structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve bus statistics.
USB_CLOSE_RAW_DEVICE_PARAMETERS	This structure is not supported. The USB_CLOSE_RAW_DEVICE_PARAMETERS structure is used with the IOCTL_USB_USER_REQUEST I/O control request to close raw access to devices on the bus.
USB_CONTROLLER_INFO_0	The USB_CONTROLLER_INFO_0 structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve information about the USB host controller.
USB_DRIVER_VERSION_PARAMETERS	The USB_DRIVER_VERSION_PARAMETERS structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve version information.
USB_PASS_THRU_PARAMETERS	The USB_PASS_THRU_PARAMETERS structure is used with the IOCTL_USB_USER_REQUEST I/O control request to pass a vendor-specific command to the host controller miniport driver.
USB_POWER_INFO	The USB_POWER_INFO structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve device power state that the host controller power policy specifies for the indicated system power state.
USB_UNICODE_NAME	The USB_UNICODE_NAME structure contains a Unicode string that specifies a symbolic link name.
USBUSER_BANDWIDTH_INFO_REQUEST	The USBUSER_BANDWIDTH_INFO_REQUEST structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve information about the allocated bandwidth.

TITLE	DESCRIPTION
USBUSER_BUS_STATISTICS_0_REQUEST	The USBUSER_BUS_STATISTICS_0_REQUEST structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve bus statistics.
USBUSER_CONTROLLER_UNICODE_NAME	The USBUSER_CONTROLLER_UNICODE_NAME structure is used in conjunction with the IOCTL_USB_USER_REQUEST I/O control request to retrieve the USB host controller driverkey name.
USBUSER_GET_DRIVER_VERSION	The USBUSER_GET_DRIVER_VERSION structure is used with the IOCTL_USB_USER_REQUEST I/O control request to read driver and interface version information.
USBUSER_PASS_THRU_REQUEST	The USBUSER_PASS_THRU_REQUEST structure is used in conjunction with the IOCTL_USB_USER_REQUEST I/O control request to send a vendor-specific command to the host controller miniport driver.
USBUSER_POWER_INFO_REQUEST	The USBUSER_POWER_INFO_REQUEST structure is used in conjunction with the IOCTL_USB_USER_REQUEST I/O control request to retrieve power policy information concerning the relationship of a specific system state to the power state of the host controller and the root hub.
USBUSER_REQUEST_HEADER	The USBUSER_REQUEST_HEADER structure is used with the IOCTL_USB_USER_REQUEST I/O control request to send a user-mode request to the USB host controller driver.
WINUSB_PIPE_INFORMATION	The WINUSB_PIPE_INFORMATION structure contains pipe information that the WinUsb_QueryPipe routine retrieves.
WINUSB_PIPE_INFORMATION_EX	The WINUSB_PIPE_INFORMATION_EX structure contains pipe information that the WinUsb_QueryPipeEx routine retrieves.
WINUSB_SETUP_PACKET	The WINUSB_SETUP_PACKET structure describes a USB setup packet.

genericusbfnioctl.h header

2/1/2021 • 2 minutes to read • Edit Online

This header is used by USB driver reference. For more information, see:

- USB driver reference genericusbfnioctl.h contains the following programming interfaces:

IOCTLs

TITLE	DESCRIPTION
IOCTL_GENERICUSBFN_ACTIVATE_USB_BUS	This I/O control code (IOCTL) is sent by a user-mode service or application to notify GenericUSBFn.sys to activate the Universal Serial Bus (USB). Once activated, the bus is prepared to process bus events and handle traffic.
IOCTL_GENERICUSBFN_BUS_EVENT_NOTIFICATION	This I/O control code (IOCTL) is sent by a user-mode service or application to register for Universal Serial Bus (USB) event.
IOCTL_GENERICUSBFN_CONTROL_STATUS_HANDSHAKE_IN	This I/O control code (IOCTL) is sent by a user mode service or application to request a zero-length control status handshake on endpoint 0 in the IN direction.
IOCTL_GENERICUSBFN_CONTROL_STATUS_HANDSHAKE_OUT	This I/O control code (IOCTL) is sent by a user-mode service or application to complete a zero-length control status handshake on endpoint 0 in the OUT direction.
IOCTL_GENERICUSBFN_DEACTIVATE_USB_BUS	This IOCTL code is nevtot supported.
IOCTL_GENERICUSBFN_GET_CLASS_INFO	This I/O control code (IOCTL) is sent by the user-mode service or application to retrieve information about a device's available pipes as configured in the registry.
IOCTL_GENERICUSBFN_GET_CLASS_INFO_EX	This I/O control code (IOCTL) is sent by a user-mode service or application to retrieve information about a device's available pipes as configured in the registry.
IOCTL_GENERICUSBFN_GET_INTERFACE_DESCRIPTOR_SET	This I/O control code (IOCTL) is sent by a user-mode service or application to get the entire interface descriptor set for a function on the device.This IOCTL request does not retrieve the interface descriptor set for the entire device.Universal Serial Bus (USB) interface descriptor set for a function on the device.
IOCTL_GENERICUSBFN_GET_PIPE_STATE	This I/O control code (IOCTL) is sent by a user-mode service or application to get the state of the specified Universal Serial Bus (USB) pipe.
IOCTL_GENERICUSBFN_REGISTER_USB_STRING	This I/O control code (IOCTL) is sent by a user-mode service or application to register a string descriptor.Universal Serial Bus (USB) string descriptor.

TITLE	DESCRIPTION
IOCTL_GENERICUSBFN_SET_PIPE_STATE	This I/O control code (IOCTL) is sent by a user-mode service or application to set the state of the specified Universal Serial Bus (USB) pipe.
IOCTL_GENERICUSBFN_TRANSFER_IN	This I/O control code (IOCTL) is sent by a user-mode service or application to issue an IN direction transfer on the endpoint that corresponds to the specified pipe ID in the input buffer.
IOCTL_GENERICUSBFN_TRANSFER_IN_APPEND_ZERO_PKT	This I/O control code (IOCTL) is sent by a user-mode service or application to issue an IN direction transfer on the endpoint that corresponds to the specified pipe ID in the input buffer.
IOCTL_GENERICUSBFN_TRANSFER_OUT	This I/O control code (IOCTL) is sent by a user-mode service or application to issue an OUT direction transfer on the endpoint that corresponds to the specified pipe ID in the input buffer.

IOCTL_GENERICUSBFN_ACTIVATE_USB_BUS IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by a user-mode service or application to notify GenericUSBFn.sys to activate the Universal Serial Bus (USB). Once activated, the bus is prepared to process bus events and handle traffic.

Input buffer

NULL.

Input buffer length

None.

Output buffer

NULL.

Output buffer length

None.

Input / Output buffer

Input / Output buffer length

Remarks

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_BUS_EVENT_NOTIFICATION IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by a user-mode service or application to register for Universal Serial Bus (USB) event. After this request completes, notifications about events such as a change in the port type or the receipt of a non-standard setup packet can be received. The **USBFN_NOTIFICATION** structure contained in the output buffer specifies which event has occurred and any associated data.

Input buffer

NULL.

Input buffer length

None.

Output buffer

NULL.

Output buffer length

None.

Input / Output buffer

Input / Output buffer length

Remarks

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_CONTROL_STATUS_HANDSHAKE_IN IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by a user mode service or application to request a zero-length control status handshake on endpoint 0 in the IN direction.

Input buffer

A **USBFNPIPEID** that specifies the pipe ID of endpoint 0.

Input buffer length

The size of a **USBFNPIPEID**.

Output buffer

NULL.

Output buffer length

None.

Input / Output buffer

Input / Output buffer length

Remarks

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_CONTROL_STATUS_HANDSHAKE_OUT

IOCTL_GENERICUSBFN_CONTROL_STATUS_HANDSHAKE_OU IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by a user-mode service or application to complete a zero-length control status handshake on endpoint 0 in the OUT direction.

Input buffer

A **USBFNPIPEID** that specifies the pipe ID of endpoint 0.

Input buffer length

The size of a **USBFNPIPEID**.

Output buffer

NULL.

Output buffer length

None.

Input / Output buffer

Input / Output buffer length

Remarks

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_CONTROL_STATUS_HANDSHAKE_IN

IOCTL_GENERICUSBFN_DEACTIVATE_USB_BUS IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This IOCTL code is nevtot supported.

This I/O control code (IOCTL) is sent by the user-mode service or application this request to notify GenericUSBFn.sys to deactivate the Universal Serial Bus (USB). When deactivated, the bus can no longer process bus events and handle traffic.

Input buffer

NULL.

Input buffer length

None.

Output buffer

NULL.

Output buffer length

None.

Input / Output buffer

Input / Output buffer length

Remarks

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_ACTIVATE_USB_BUS

IOCTL_GENERICUSBFN_GET_CLASS_INFO IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by the user-mode service or application to retrieve information about a device's available pipes as configured in the registry.

Input buffer

NULL.

Input buffer length

None.

Output buffer

A USBFN_CLASS_INFORMATION_PACKET that provides information about the available pipes for a device.

Output buffer length

The size of a USBFN_CLASS_INFORMATION_PACKET structure.

Input / Output buffer

Input / Output buffer length

Remarks

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_GET_CLASS_INFO_EX

IOCTL_GENERICUSBFN_GET_CLASS_INFO_EX IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by a user-mode service or application to retrieve information about a device's available pipes as configured in the registry.

Input buffer

NULL.

Input buffer length

None.

Output buffer

A USBFN_CLASS_INFORMATION_PACKET_EX that provides information about the available pipes for a device.

Output buffer length

The size of a USBFN_CLASS_INFORMATION_PACKET_EX structure.

Input / Output buffer

Input / Output buffer length

Remarks

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_GET_CLASS_INFO

IOCTL_GENERICUSBFN_GET_INTERFACE_DESCRIPTOR_SET IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by a user-mode service or application to get the entire Universal Serial Bus (USB) interface descriptor set for a function on the device.

> **Note** This IOCTL request does not retrieve the interface descriptor set for the entire device.

Input buffer

A pointer to a USBFN_INTERFACE_INFO structure.

Input buffer length

The size of a USBFN_INTERFACE_INFO structure.

Output buffer

A pointer to a buffer that contains a USBFN_INTERFACE_INFO structure. The USB function class extension (UFX) populates the structure with the entire interface descriptor set including its endpoint descriptors.

Output buffer length

The size of a USBFN_INTERFACE_INFO.

Input / Output buffer

Input / Output buffer length

Remarks

This request must be sent after sending the IOCTL_GENERICUSBFN_ACTIVATE_USB_BUS request.

The length of the entire interface descriptor is variable. The class driver might need to send this IOCTL request twice to get the entire descriptor set.

If the length of the entire descriptor set is greater than the specified output buffer length, UFX sets the **Size** member of USBFN_INTERFACE_INFO to the actual buffer length and fails the request with STATUS_BUFFER_TOO_SMALL. The driver must then allocated an output buffer of length specified by **Size** and resend the request.

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_GET_PIPE_STATE IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by a user-mode service or application to get the state of the specified Universal Serial Bus (USB) pipe.

Input buffer

A **USBFNPIPEID** that specifies the ID of the pipe to retrieve state information for.

Input buffer length

The size of a **USBFNPIPEID**.

Output buffer

Contains a BOOLEAN value that specifies whether the specified pipe is stalled. A value of TRUE if the specified pipe is stalled; FALSE if otherwise.

Output buffer length

The size of the output buffer in bytes.

Input / Output buffer

Input / Output buffer length

Remarks

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_SET_PIPE_STATE

IOCTL_GENERICUSBFN_REGISTER_USB_STRING IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by a user-mode service or application to register a Universal Serial Bus (USB) string descriptor.

Input buffer

A pointer to a buffer that contains a USBFN_USB_STRING structure with the USB string descriptor.

Input buffer length

The size of a USBFN_USB_STRING structure.

Output buffer

NULL.

Output buffer length

None.

Input / Output buffer

Input / Output buffer length

Remarks

This request must be sent after sending the IOCTL_GENERICUSBFN_ACTIVATE_USB_BUS request.

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_SET_PIPE_STATE IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by a user-mode service or application to set the state of the specified Universal Serial Bus (USB) pipe.

Input buffer

A **USBFNPIPEID** that specifies the ID of the pipe to configure.

Input buffer length

The size of a **USBFNPIPEID**.

Output buffer

Contains a boolean value that specifies whether the specified pipe is stalled. A value of TRUE if the specified pipe is stalled; FALSE if otherwise.

Output buffer length

The size of the output buffer in bytes.

Input / Output buffer

Input / Output buffer length

Remarks

The pipe will send STALL transaction packets to the host when stalled. For more information, see the USB specification.

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_GET_PIPE_STATE

IOCTL_GENERICUSBFN_TRANSFER_IN IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by a user-mode service or application to issue an IN direction transfer on the endpoint that corresponds to the specified pipe ID in the input buffer.

Input buffer

A **USBFNPIPEID** that specifies the ID of the pipe to conduct the transfer on.

Input buffer length

The size of a **USBFNPIPEID**.

Output buffer

The data to send to the host. From the host perspective the data is sent from the IN direction, representing an outbound transfer from the device to the host.

Output buffer length

The size of the output buffer in bytes.

Input / Output buffer

Input / Output buffer length

Remarks

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_TRANSFER_IN_APPEND_ZERO_PKT

IOCTL_GENERICUSBFN_TRANSFER_OUT

IOCTL_GENERICUSBFN_TRANSFER_IN_APPEND_ZERO_PKT IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by a user-mode service or application to issue an IN direction transfer on the endpoint that corresponds to the specified pipe ID in the input buffer. A zero-length packet is automatically appended after the data in the output buffer is successfully sent and the transfer payload size is a multiple of the endpoint's maximum packet size. This should be used on the last I/O request that corresponds to a Universal Serial Bus (USB) transfer.

Input buffer

A **USBFNPIPEID** that specifies the ID of the pipe to conduct the transfer on.

Input buffer length

The size of a **USBFNPIPEID**.

Output buffer

The data to send to the host.

Output buffer length

The size of the output buffer in bytes.

Input / Output buffer

Input / Output buffer length

Remarks

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_TRANSFER_IN

IOCTL_GENERICUSBFN_TRANSFER_OUT

IOCTL_GENERICUSBFN_TRANSFER_OUT IOCTL (genericusbfnioctl.h)

2/1/2021 • 2 minutes to read • Edit Online

This I/O control code (IOCTL) is sent by a user-mode service or application to issue an OUT direction transfer on the endpoint that corresponds to the specified pipe ID in the input buffer.

Input buffer

A **USBFNPIPEID** that specifies the ID of the pipe to conduct the transfer on.

Input buffer length

The size of a **USBFNPIPEID**.

Output buffer

The data received from the host.

Output buffer length

The size of the output buffer in bytes.

Input / Output buffer

Input / Output buffer length

Remarks

If this I/O control code (IOCTL) is being called synchronously, set the *lpOverlapped* parameter to NULL. If this IOCTL is called asynchronously, assign the *lpOverlapped* parameter to a pointer to an OVERLAPPED structure that contains a handle to an event object. The event objects signal when the operation is completed.

The return value is a BOOL value that indicates success or failure of the operation. TRUE indicates success, FALSE otherwise.

Requirements

Header	genericusbfnioctl.h (include GenericUsbFnIoctl.h)

See also

DeviceIoControl

IOCTL_GENERICUSBFN_TRANSFER_IN

IOCTL_GENERICUSBFN_TRANSFER_IN_APPEND_ZERO_PKT

usbuser.h header

2/1/2021 • 2 minutes to read • Edit Online

This header is used by USB driver reference. For more information, see:

- USB driver reference usbuser.h contains the following programming interfaces:

IOCTLs

TITLE	DESCRIPTION
IOCTL_GET_HCD_DRIVERKEY_NAME	The IOCTL_GET_HCD_DRIVERKEY_NAME I/O control request retrieves the driver key name in the registry for a USB host controller driver.
IOCTL_USB_DIAGNOSTIC_MODE_OFF	The IOCTL_USB_DIAGNOSTIC_MODE_OFF I/O control has been deprecated. Do not use.
IOCTL_USB_DIAGNOSTIC_MODE_ON	The IOCTL_USB_DIAGNOSTIC_MODE_ON I/O control has been deprecated. Do not use.
IOCTL_USB_GET_ROOT_HUB_NAME	The IOCTL_USB_GET_ROOT_HUB_NAME I/O control request is used with the USB_ROOT_HUB_NAME structure to retrieve the symbolic link name of the root hub.IOCTL_USB_GET_ROOT_HUB_NAME is a user-mode I/O control request.
IOCTL_USB_USER_REQUEST	The IOCTL_USB_USER_REQUEST I/O control request is available to both user-mode applications and kernel-mode drivers.

Structures

TITLE	DESCRIPTION
USB_BANDWIDTH_INFO	The USB_BANDWIDTH_INFO structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve information about the allocated bandwidth.
USB_BUS_STATISTICS_0	The USB_BUS_STATISTICS_0 structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve bus statistics.
USB_CLOSE_RAW_DEVICE_PARAMETERS	This structure is not supported. The USB_CLOSE_RAW_DEVICE_PARAMETERS structure is used with the IOCTL_USB_USER_REQUEST I/O control request to close raw access to devices on the bus.
USB_CONTROLLER_INFO_0	The USB_CONTROLLER_INFO_0 structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve information about the USB host controller.

TITLE	DESCRIPTION
USB_DRIVER_VERSION_PARAMETERS	The USB_DRIVER_VERSION_PARAMETERS structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve version information.
USB_PASS_THRU_PARAMETERS	The USB_PASS_THRU_PARAMETERS structure is used with the IOCTL_USB_USER_REQUEST I/O control request to pass a vendor-specific command to the host controller miniport driver.
USB_POWER_INFO	The USB_POWER_INFO structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve device power state that the host controller power policy specifies for the indicated system power state.
USB_UNICODE_NAME	The USB_UNICODE_NAME structure contains a Unicode string that specifies a symbolic link name.
USBUSER_BANDWIDTH_INFO_REQUEST	The USBUSER_BANDWIDTH_INFO_REQUEST structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve information about the allocated bandwidth.
USBUSER_BUS_STATISTICS_0_REQUEST	The USBUSER_BUS_STATISTICS_0_REQUEST structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve bus statistics.
USBUSER_CONTROLLER_UNICODE_NAME	The USBUSER_CONTROLLER_UNICODE_NAME structure is used in conjunction with the IOCTL_USB_USER_REQUEST I/O control request to retrieve the USB host controller driverkey name.
USBUSER_GET_DRIVER_VERSION	The USBUSER_GET_DRIVER_VERSION structure is used with the IOCTL_USB_USER_REQUEST I/O control request to read driver and interface version information.
USBUSER_PASS_THRU_REQUEST	The USBUSER_PASS_THRU_REQUEST structure is used in conjunction with the IOCTL_USB_USER_REQUEST I/O control request to send a vendor-specific command to the host controller miniport driver.
USBUSER_POWER_INFO_REQUEST	The USBUSER_POWER_INFO_REQUEST structure is used in conjunction with the IOCTL_USB_USER_REQUEST I/O control request to retrieve power policy information concerning the relationship of a specific system state to the power state of the host controller and the root hub.
USBUSER_REQUEST_HEADER	The USBUSER_REQUEST_HEADER structure is used with the IOCTL_USB_USER_REQUEST I/O control request to send a user-mode request to the USB host controller driver.

Enumerations

TITLE	DESCRIPTION

TITLE	DESCRIPTION
USB_USER_ERROR_CODE	The USB_USER_ERROR_CODE enumeration lists the error codes that a USB user-mode request reports when it fails.
WDMUSB_POWER_STATE	The WDMUSB_POWER_STATE enumeration indicates the power state of a host controller or root hub.

IOCTL_GET_HCD_DRIVERKEY_NAME IOCTL (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **IOCTL_GET_HCD_DRIVERKEY_NAME** I/O control request retrieves the driver key name in the registry for a USB host controller driver.

IOCTL_GET_HCD_DRIVERKEY_NAME is a user-mode I/O control request. This request targets the USB host controller (GUID_DEVINTERFACE_USB_HOST_CONTROLLER).

Input buffer

None.

Input buffer length

None.

Output buffer

The **AssociatedIrp.SystemBuffer** member specifies the address of a caller-allocated buffer that contains a USB_HCD_DRIVERKEY_NAME structure. On output, this structure holds the driver key name. For more information, see Remarks.

Output buffer length

The size of this buffer is specified in the **Parameters.DeviceIoControl.OutputBufferLength** member.

Input / Output buffer

Input / Output buffer length

Remarks

To get the driver key name in the registry, you must perform the following tasks:

1. Declare a variable of the type USB_HCD_DRIVERKEY_NAME.
2. Send an **IOCTL_GET_HCD_DRIVERKEY_NAME** request by specifying the address and size of the variable in the output parameters. On return, the **ActualLength** member of USB_HCD_DRIVERKEY_NAME contains the length required to allocate a buffer to hold a **USB_HCD_DRIVERKEY_NAME** that is populated with the driver key name.
3. Allocate memory for a buffer to hold a USB_HCD_DRIVERKEY_NAME structure. The size of the buffer must be the received **ActualLength** value.
4. Send an **IOCTL_GET_HCD_DRIVERKEY_NAME** request by passing a pointer to the allocated buffer and its size in the output parameters. On return, the **DriverKeyName** member of USB_HCD_DRIVERKEY_NAME is a null-terminated Unicode string that contains the name of the driver key associated with the host controller driver.

The following example code shows how to send the **IOCTL_GET_HCD_DRIVERKEY_NAME** I/O control request.

```
/*++

Routine Description:

This routine prints the name of the driver key associated with
the specified host controller driver
```

```
the specified host controller driver.

Arguments:

HCD - Handle for host controller driver.

Return Value: Boolean that indicates success or failure.

--*/

BOOL GetHCDDriverKeyName (HANDLE  HCD)
{
    BOOL                    success;
    ULONG                   nBytes;
    USB_HCD_DRIVERKEY_NAME  driverKeyName;
    PUSB_HCD_DRIVERKEY_NAME driverKeyNameW;

    driverKeyNameW = NULL;

    // 1. Get the length of the name of the driver key.
    success = DeviceIoControl(HCD,
        IOCTL_GET_HCD_DRIVERKEY_NAME,
        NULL,
        0,
        &driverKeyName,
        sizeof(driverKeyName),
        &nBytes,
        NULL);

    if (!success)
    {
        printf("First IOCTL_GET_HCD_DRIVERKEY_NAME request failed\n");
        goto GetHCDDriverKeyNameDone;
    }

    //2. Get the length of the driver key name.
    nBytes = driverKeyName.ActualLength;

    if (nBytes <= sizeof(driverKeyName))
    {
        printf("Incorrect length received by IOCTL_GET_HCD_DRIVERKEY_NAME.\n");
        goto GetHCDDriverKeyNameDone;
    }

    // 3. Allocate memory for a USB_HCD_DRIVERKEY_NAME
    //    to hold the driver key name.
    driverKeyNameW = (PUSB_HCD_DRIVERKEY_NAME) malloc(nBytes);

    if (driverKeyNameW == NULL)
    {
        printf("Failed to allocate memory.\n");
        goto GetHCDDriverKeyNameDone;
    }

    // Get the name of the driver key of the device attached to
    // the specified port.
    success = DeviceIoControl(HCD,
        IOCTL_GET_HCD_DRIVERKEY_NAME,
        NULL,
        0,
        driverKeyNameW,
        nBytes,
        &nBytes,
        NULL);

    if (!success)
    {
        printf("Second IOCTL_GET_HCD_DRIVERKEY_NAME request failed.\n");
        goto GetHCDDriverKeyNameDone;
```

```
    }

    // print the driver key name.
    printf("Driver Key Name: %s.\n", driverKeyNameW->DriverKeyName);


GetHCDDriverKeyNameDone:

    // Cleanup.
    // Free the allocated memory for USB_HCD_DRIVERKEY_NAME.

    if (driverKeyNameW != NULL)
    {
        free(driverKeyNameW);
        driverKeyNameW = NULL;
    }

    return success;
}
```

Requirements

Header	usbuser.h (include Usbioctl.h)

See also

USB_HCD_DRIVERKEY_NAME

IOCTL_USB_DIAGNOSTIC_MODE_OFF IOCTL (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **IOCTL_USB_DIAGNOSTIC_MODE_OFF** I/O control has been deprecated. Do not use.

Input buffer

Input buffer length

Output buffer

Output buffer length

Input / Output buffer

Input / Output buffer length

Requirements

Header	usbuser.h (include Usbioctl.h)

IOCTL_USB_DIAGNOSTIC_MODE_ON IOCTL (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **IOCTL_USB_DIAGNOSTIC_MODE_ON** I/O control has been deprecated. Do not use.

Input buffer

Input buffer length

Output buffer

Output buffer length

Input / Output buffer

Input / Output buffer length

Requirements

Header	usbuser.h (include Usbioctl.h)

IOCTL_USB_GET_ROOT_HUB_NAME IOCTL (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **IOCTL_USB_GET_ROOT_HUB_NAME** I/O control request is used with the USB_ROOT_HUB_NAME structure to retrieve the symbolic link name of the root hub.

IOCTL_USB_GET_ROOT_HUB_NAME is a user-mode I/O control request. This request targets the USB host controller (GUID_DEVINTERFACE_USB_HOST_CONTROLLER).

Input buffer

None.

Input buffer length

None.

Output buffer

On output, the **AssociatedIrp.SystemBuffer** member points to a USB_ROOT_HUB_NAME structure that contains the symbolic link name of the root hub. The leading "\xxx\ " text is not included in the retrieved string.

Output buffer length

The size of a USB_ROOT_HUB_NAME structure.

Input / Output buffer

Input / Output buffer length

Requirements

Header	usbuser.h (include Usbioctl.h)

See also

USB_ROOT_HUB_NAME

IOCTL_USB_USER_REQUEST IOCTL (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **IOCTL_USB_USER_REQUEST** I/O control request is available to both user-mode applications and kernel-mode drivers.

IOCTL_USB_USER_REQUEST is a user-mode I/O control request. This request targets the USB host controller (GUID_DEVINTERFACE_USB_HOST_CONTROLLER).

Callers can specify any of the following request codes:

USBUSER_CLEAR_ROOTPORT_FEATURE
Do not use this request.
USBUSER_GET_CONTROLLER_DRIVER_KEY
Reports the host controller driver key in a USB_UNICODE_NAME-typed Unicode string. This request is always enabled.
USBUSER_GET_CONTROLLER_INFO_0
Retrieves a USB_CONTROLLER_INFO_0 structure that describes the host controller. This request is always enabled.
USBUSER_GET_ROOTPORT_STATUS
Do not use this request.
USBUSER_GET_ROOTHUB_SYMBOLIC_NAME
Reports the root hub symbolic name in a USB_UNICODE_NAME-typed Unicode string. This request is always enabled.
USBUSER_INVALID_REQUEST
Do not use this request.
USBUSER_OP_CLOSE_RAW_DEVICE
Do not use this request.
USBUSER_OP_OPEN_RAW_DEVICE
Do not use this request.
USBUSER_OP_MASK_DEVONLY_API
Do not use this request.
USBUSER_OP_MASK_HCTEST_API
Do not use this request.
USBUSER_OP_RAW_RESET_PORT
Do not use this request.
USBUSER_OP_SEND_ONE_PACKET
Do not use this request.
USBUSER_OP_SEND_RAW_COMMAND
Do not use this request.
USBUSER_SET_ROOTPORT_FEATURE
Do not use this request.
USBUSER_PASS_THRU
Sends a vendor specific command that is defined by the USB_PASS_THRU_PARAMETERS structure to the host controller miniport driver. This request is always enabled.
USBUSER_GET_BANDWIDTH_INFORMATION
Retrieves a USB_BANDWIDTH_INFO structure that contains information about the allocated bandwidth. This request is always enabled.
USBUSER_GET_POWER_STATE_MAP
Retrieves a USB_POWER_INFO structure that contains information about the power state of the host controller and root hubs. This request is always enabled.

USBUSER_GET_BUS_STATISTICS_0
Retrieves a USB_BUS_STATISTICS_0 structure that contains bus statistics. This request is always enabled.
USBUSER_GET_BUS_STATISTICS_0_AND_RESET
Do not use this request.
USBUSER_GET_USB_DRIVER_INFORMATION
Retrieves a USB_DRIVER_VERSION_PARAMETERS structure that indicates the version of the driver, USB stack, and associated interfaces. This request is always enabled.
USBUSER_GET_USB2_HW_VERSION
Do not use this request.

Input buffer

The buffer at **Irp->AssociatedIrp.SystemBuffer** contains a user request header structure (USBUSER_REQUEST_HEADER) that defines the request. Following the header structure is a structure that holds the parameters of the request. For more information about the parameter structures that correspond to each request, see the description of each request.

Input buffer length

The size of a USBUSER_REQUEST_HEADER structure.

Output buffer

A parameter structure immediately follows the USBUSER_REQUEST_HEADER structure at **Irp->AssociatedIrp.SystemBuffer**. For some user requests, the parameter structure will contain output data when the request completes.

Output buffer length

The length of the parameter structure.

Input / Output buffer

Input / Output buffer length

Requirements

Header	usbuser.h (include Usbuser.h)

See also

USBUSER_REQUEST_HEADER

USB_BANDWIDTH_INFO

USB_CONTROLLER_INFO_0

USB_DRIVER_VERSION_PARAMETERS

USB_PASS_THRU_PARAMETERS

USB_POWER_INFO

USB_UNICODE_NAME

USB_BANDWIDTH_INFO structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USB_BANDWIDTH_INFO** structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve information about the allocated bandwidth.

Syntax

```
typedef struct _USB_BANDWIDTH_INFO {
  ULONG DeviceCount;
  ULONG TotalBusBandwidth;
  ULONG Total32secBandwidth;
  ULONG AllocedBulkAndControl;
  ULONG AllocedIso;
  ULONG AllocedInterrupt_1ms;
  ULONG AllocedInterrupt_2ms;
  ULONG AllocedInterrupt_4ms;
  ULONG AllocedInterrupt_8ms;
  ULONG AllocedInterrupt_16ms;
  ULONG AllocedInterrupt_32ms;
} USB_BANDWIDTH_INFO, *PUSB_BANDWIDTH_INFO;
```

Members

`DeviceCount`

The number of devices on the bus.

`TotalBusBandwidth`

The amount of allocated bandwidth, in bits per millisecond.

`Total32secBandwidth`

The amount of allocated bandwidth bits in each 32-millisecond time slice.

`AllocedBulkAndControl`

The amount of bandwidth, in bits per 32-millisecond, that is allocated for bulk and control transfers.

`AllocedIso`

The amount of bandwidth, in bits per 32-millisecond, that is allocated for isochronous transfers.

`AllocedInterrupt_1ms`

The amount of bandwidth, in bits per 32-millisecond, that is allocated for interrupt transactions when the period is 1 millisecond.

`AllocedInterrupt_2ms`

The amount of bandwidth, in bits per 32-millisecond, that is allocated for interrupt transactions when the period is 2 milliseconds.

`AllocedInterrupt_4ms`

The amount of bandwidth, in bits per 32-millisecond, that is allocated for interrupt transactions when the period

is 4 milliseconds.

`AllocedInterrupt_8ms`

The amount of bandwidth, in bits per 32-millisecond, that is allocated for interrupt transactions when the period is 8 milliseconds.

`AllocedInterrupt_16ms`

The amount of bandwidth, in bits per 32-millisecond, that is allocated for interrupt transactions when the period is 16 milliseconds.

`AllocedInterrupt_32ms`

The amount of bandwidth, in bits per 32-millisecond, that is allocated for interrupt transactions when the period is 32 milliseconds.

Remarks

The **USB_BANDWIDTH_INFO** structure is used with the USBUSER_GET_BANDWIDTH_INFORMATION user-mode request. For a description of this request, see IOCTL_USB_USER_REQUEST.

In Windows 8, this request completes successfully. However, the values retrieved from the underlying USB 3.0 driver stack do not reflect actual information about the allocated bandwidth. That is because the bandwidth information is not exposed by xHCI controllers.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Structures

USB_BUS_STATISTICS_0 structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USB_BUS_STATISTICS_0** structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve bus statistics.

Syntax

```
typedef struct _USB_BUS_STATISTICS_0 {
  ULONG         DeviceCount;
  LARGE_INTEGER CurrentSystemTime;
  ULONG         CurrentUsbFrame;
  ULONG         BulkBytes;
  ULONG         IsoBytes;
  ULONG         InterruptBytes;
  ULONG         ControlDataBytes;
  ULONG         PciInterruptCount;
  ULONG         HardResetCount;
  ULONG         WorkerSignalCount;
  ULONG         CommonBufferBytes;
  ULONG         WorkerIdleTimeMs;
  BOOLEAN       RootHubEnabled;
  UCHAR         RootHubDevicePowerState;
  UCHAR         Unused;
  UCHAR         NameIndex;
} USB_BUS_STATISTICS_0, *PUSB_BUS_STATISTICS_0;
```

Members

`DeviceCount`

The number of devices on the bus.

`CurrentSystemTime`

The current system time.

`CurrentUsbFrame`

The number of the current USB frame.

`BulkBytes`

The amount, in bytes, of bulk transfer data.

`IsoBytes`

The amount, in bytes, of isochronous data.

`InterruptBytes`

The amount, in bytes, of interrupt data.

`ControlDataBytes`

The amount, in bytes, of control data.

`PciInterruptCount`

The amount, in bytes, of interrupt data.

`HardResetCount`

The number of hard bus resets that have occurred.

`WorkerSignalCount`

The number of times that a worker thread has signaled completion of a task.

`CommonBufferBytes`

The number of bytes that are transferred by common buffer.

`WorkerIdleTimeMs`

The amount of time, in milliseconds, that worker threads have been idle.

`RootHubEnabled`

A Boolean value that indicates whether the root hub is enabled. If **TRUE**, he root hub is enabled. If **FALSE**, the root hub is disabled.

`RootHubDevicePowerState`

The power state of the root hub devices. This member can have any of the following values:

VALUE	MEANING
0	D0 power state
1	D1 power state
2	D2 power state
3	D3 power state

`Unused`

If this member is 1, the bus is active. If 0, the bus is inactive.

`NameIndex`

The index that is used to generate a symbolic link name for the hub PDO. This format of the symbolic link is USBPDO-*n*, where *n* is the value in **NameIndex**.

Remarks

The **USB_BUS_STATISTICS_0** structure is used with the USBUSER_BUS_STATISTICS_0 user-mode request. For a description of this request, see IOCTL_USB_USER_REQUEST.

In Windows 8, this request completes successfully. However, the values retrieved from the underlying USB 3.0 driver stack do not reflect actual bus statistics.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Structures

USB_CLOSE_RAW_DEVICE_PARAMETERS structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

This structure is not supported.

The **USB_CLOSE_RAW_DEVICE_PARAMETERS** structure is used with the IOCTL_USB_USER_REQUEST I/O control request to close raw access to devices on the bus.

Syntax

```
typedef struct _USB_CLOSE_RAW_DEVICE_PARAMETERS {
  ULONG xxx;
} USB_CLOSE_RAW_DEVICE_PARAMETERS, *PUSB_CLOSE_RAW_DEVICE_PARAMETERS;
```

Members

`xxx`

Reserved.

Remarks

The USB_CLOSE_RAW_DEVICE_PARAMETERS structure is used with the USBUSER_OP_CLOSE_RAW_DEVICE user-mode request. For a description of this request, see IOCTL_USB_USER_REQUEST.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB_CONTROLLER_INFO_0 structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USB_CONTROLLER_INFO_0** structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve information about the USB host controller.

Syntax

```
typedef struct _USB_CONTROLLER_INFO_0 {
  ULONG                  PciVendorId;
  ULONG                  PciDeviceId;
  ULONG                  PciRevision;
  ULONG                  NumberOfRootPorts;
  USB_CONTROLLER_FLAVOR ControllerFlavor;
  ULONG                  HcFeatureFlags;
} USB_CONTROLLER_INFO_0, *PUSB_CONTROLLER_INFO_0;
```

Members

`PciVendorId`

The vendor identifier that is associated with the host controller device.

`PciDeviceId`

The device identifier that is associated with the host controller.

`PciRevision`

The revision number of the host controller device.

`NumberOfRootPorts`

The number of root hub ports that the host controller has.

> **Note** In Windows 8, the USB 3.0 driver stack does not include the number of SuperSpeed hubs in the reported **NumberOfRootPorts** value.

`ControllerFlavor`

A USB_CONTROLLER_FLAVOR-typed enumerator that specifies the type of controller.

`HcFeatureFlags`

A bitwise OR of some combination of the following host controller feature flags.

HOST CONTROLLER FEATURE	MEANING
USB_HC_FEATURE_FLAG_PORT_POWER_SWITCHING	Power switching is enabled on the host controller. This flag allows powering of hot-plug devices.
USB_HC_FEATURE_FLAG_SEL_SUSPEND	Selective suspend is enabled on the host controller.

USB_HC_FEATURE_LEGACY_BIOS	The host controller has a legacy BIOS.

> **Note** In Windows 8, the underlying USB 3.0 driver stack does not set any host controller feature flags in **HcFeatureFlags**.

Remarks

The **USB_CONTROLLER_INFO_0** structure is used with the USBUSER_GET_CONTROLLER_INFO_0 user-mode request. For a description of this request, see IOCTL_USB_USER_REQUEST.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB_CONTROLLER_FLAVOR

USB_DRIVER_VERSION_PARAMETERS structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USB_DRIVER_VERSION_PARAMETERS** structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve version information.

Syntax

```
typedef struct _USB_DRIVER_VERSION_PARAMETERS {
  ULONG   DriverTrackingCode;
  ULONG   USBDI_Version;
  ULONG   USBUSER_Version;
  BOOLEAN CheckedPortDriver;
  BOOLEAN CheckedMiniportDriver;
  USHORT  USB_Version;
} USB_DRIVER_VERSION_PARAMETERS, *PUSB_DRIVER_VERSION_PARAMETERS;
```

Members

`DriverTrackingCode`

A tracking code that identifies the revision of the USB stack.

`USBDI_Version`

The version of the USB driver interface that the USB stack supports.

`USBUSER_Version`

The version of the USB user interface that the USB stack supports.

`CheckedPortDriver`

A Boolean value that indicates whether the checked version of the host controller driver is loaded. If **TRUE**, the checked version of the host controller driver is loaded. If **FALSE**, the checked version is not loaded.

`CheckedMiniportDriver`

A Boolean value that indicates whether the checked version of the host controller miniport driver is loaded. If **TRUE**, the checked version of the host controller miniport driver is loaded. If **FALSE**, the checked version is not loaded.

`USB_Version`

The USB version that the USB stack supports. A value of 0x0110 indicates that the USB stack supports version 1.1. A value of 0x0200 indicates the USB stack supports version 2.0.

Remarks

The **USB_DRIVER_VERSION_PARAMETERS** structure is used with the USBUSER_GET_USB_DRIVER_VERSION user-mode request. For a description of this request, see IOCTL_USB_USER_REQUEST.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Structures

USB_PASS_THRU_PARAMETERS structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USB_PASS_THRU_PARAMETERS** structure is used with the IOCTL_USB_USER_REQUEST I/O control request to pass a vendor-specific command to the host controller miniport driver.

Syntax

```
typedef struct _USB_PASS_THRU_PARAMETERS {
  GUID  FunctionGUID;
  ULONG ParameterLength;
  UCHAR Parameters[4];
} USB_PASS_THRU_PARAMETERS, *PUSB_PASS_THRU_PARAMETERS;
```

Members

`FunctionGUID`

A GUID that identifies the operation for the host controller miniport driver.

`ParameterLength`

The size, in bytes, of the USB_PASS_THRU_PARAMETERS structure.

`Parameters`

A variable length array with the parameter data for the command.

Remarks

The **USB_PASS_THRU_PARAMETERS** structure is used with the USBUSER_PASS_THRU user-mode request. For more information about this request, see IOCTL_USB_USER_REQUEST.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Structures

USB_POWER_INFO structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USB_POWER_INFO** structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve device power state that the host controller power policy specifies for the indicated system power state.

Syntax

```
typedef struct _USB_POWER_INFO {
  WDMUSB_POWER_STATE SystemState;
  WDMUSB_POWER_STATE HcDevicePowerState;
  WDMUSB_POWER_STATE HcDeviceWake;
  WDMUSB_POWER_STATE HcSystemWake;
  WDMUSB_POWER_STATE RhDevicePowerState;
  WDMUSB_POWER_STATE RhDeviceWake;
  WDMUSB_POWER_STATE RhSystemWake;
  WDMUSB_POWER_STATE LastSystemSleepState;
  BOOLEAN            CanWakeup;
  BOOLEAN            IsPowered;
} USB_POWER_INFO, *PUSB_POWER_INFO;
```

Members

`SystemState`

On input, a WDMUSB_POWER_STATE-type enumerator value that specifies the system power state.

`HcDevicePowerState`

On output, an WDMUSB_POWER_STATE-type enumerator value that specifies the device power state of the host controller.

`HcDeviceWake`

On output, a WDMUSB_POWER_STATE-type enumerator value that specifies whether the host controller is in a wake state.

`HcSystemWake`

On output, a WDMUSB_POWER_STATE-type enumerator value that specifies whether the host controller can wake the system.

`RhDevicePowerState`

On output, a WDMUSB_POWER_STATE-type enumerator value that specifies the device power state of the root hub.

`RhDeviceWake`

On output, a WDMUSB_POWER_STATE-type enumerator value that specifies whether the root hub is in a wake state.

`RhSystemWake`

On output, a WDMUSB_POWER_STATE-type enumerator value that specifies whether the root hub can wake the system.

`LastSystemSleepState`

On output, a WDMUSB_POWER_STATE-type enumerator value that specifies the last system sleep state.

`CanWakeup`

A Boolean value that indicates whether the host controller device can wake up the system from the specified system power state. If **TRUE**, the host controller device can wake up the system. If **FALSE**, the host controller cannot wake up the system.

`IsPowered`

A Boolean value that indicates whether the host controller is powered when in the specified system power state. If **TRUE**, the host controller is powered. If **FALSE**, the host controller is not powered.

Remarks

The **USB_POWER_INFO** structure is used with the USBUSER_GET_POWER_STATE_MAP user-mode request. For more information about this request, see IOCTL_USB_USER_REQUEST.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Structures

USB_UNICODE_NAME structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USB_UNICODE_NAME** structure contains a Unicode string that specifies a symbolic link name.

Syntax

```
typedef struct _USB_UNICODE_NAME {
  ULONG Length;
  WCHAR String[1];
} USB_UNICODE_NAME, *PUSB_UNICODE_NAME;
```

Members

`Length`

The length, in bytes, of the string.

`String`

A pointer to the string.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

USB Structures

USB_USER_ERROR_CODE enumeration (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USB_USER_ERROR_CODE** enumeration lists the error codes that a USB user-mode request reports when it fails.

Syntax

```
typedef enum _USB_USER_ERROR_CODE {
  UsbUserSuccess,
  UsbUserNotSupported,
  UsbUserInvalidRequestCode,
  UsbUserFeatureDisabled,
  UsbUserInvalidHeaderParameter,
  UsbUserInvalidParameter,
  UsbUserMiniportError,
  UsbUserBufferTooSmall,
  UsbUserErrorNotMapped,
  UsbUserDeviceNotStarted,
  UsbUserNoDeviceConnected
} USB_USER_ERROR_CODE;
```

Constants

NAME	DESCRIPTION
UsbUserSuccess	The user request succeeded.
UsbUserNotSupported	The user request was not supported.
UsbUserInvalidRequestCode	The user request code was invalid.
UsbUserFeatureDisabled	The feature that was specified by user request is disabled.
UsbUserInvalidHeaderParameter	The user request contains an invalid header parameter.
UsbUserInvalidParameter	The user request contains an invalid parameter.
UsbUserMiniportError	The user request failed because of a miniport driver error.
UsbUserBufferTooSmall	The user request failed because the data buffer was too small.
UsbUserErrorNotMapped	The USB stack could not map the error to one of the errors that are listed in this enumeration.
UsbUserDeviceNotStarted	The device was not started.
UsbUserNoDeviceConnected	The device was not connected.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

USB Constants and Enumerations

USBUSER_BANDWIDTH_INFO_REQUEST structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USBUSER_BANDWIDTH_INFO_REQUEST** structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve information about the allocated bandwidth.

Syntax

```
typedef struct _USBUSER_BANDWIDTH_INFO_REQUEST {
  USBUSER_REQUEST_HEADER Header;
  USB_BANDWIDTH_INFO     BandwidthInformation;
} USBUSER_BANDWIDTH_INFO_REQUEST, *PUSBUSER_BANDWIDTH_INFO_REQUEST;
```

Members

`Header`

A USBUSER_REQUEST_HEADER structure that specifies the user-mode request on input to IOCTL_USB_USER_REQUEST and provides buffer and status information on output.

`BandwidthInformation`

A USB_BANDWIDTH_INFO structure that reports bandwidth allocation information.

Remarks

The **USBUSER_BANDWIDTH_INFO_REQUEST** structure is used with the USBUSER_GET_BANDWIDTH_INFORMATION user-mode request. For more information about this request, see IOCTL_USB_USER_REQUEST.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Structures

USBUSER_REQUEST_HEADER

USB_BANDWIDTH_INFO

USBUSER_BUS_STATISTICS_0_REQUEST structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USBUSER_BUS_STATISTICS_0_REQUEST** structure is used with the IOCTL_USB_USER_REQUEST I/O control request to retrieve bus statistics.

Syntax

```
typedef struct _USBUSER_BUS_STATISTICS_0_REQUEST {
  USBUSER_REQUEST_HEADER Header;
  USB_BUS_STATISTICS_0   BusStatistics0;
} USBUSER_BUS_STATISTICS_0_REQUEST, *PUSBUSER_BUS_STATISTICS_0_REQUEST;
```

Members

`Header`

A USBUSER_REQUEST_HEADER structure that specifies the user-mode request on input to IOCTL_USB_USER_REQUEST and provides buffer and status information on output.

`BusStatistics0`

A USB_BUS_STATISTICS_0 structure that reports bus statistics.

Remarks

The **USBUSER_BUS_STATISTICS_0_REQUEST** structure is used with the USBUSER_GET_BUS_STATISTICS_0 user-mode request. For more information about this request, see IOCTL_USB_USER_REQUEST.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Structures

USBUSER_REQUEST_HEADER

USB_BUS_STATISTICS_0

USBUSER_CONTROLLER_UNICODE_NAME structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USBUSER_CONTROLLER_UNICODE_NAME** structure is used in conjunction with the IOCTL_USB_USER_REQUEST I/O control request to retrieve the USB host controller driverkey name.

Syntax

```
typedef struct _USBUSER_CONTROLLER_UNICODE_NAME {
  USBUSER_REQUEST_HEADER Header;
  USB_UNICODE_NAME       UnicodeName;
} USBUSER_CONTROLLER_UNICODE_NAME, *PUSBUSER_CONTROLLER_UNICODE_NAME;
```

Members

`Header`

Contains a structure of type USBUSER_REQUEST_HEADER that specifies the user-mode request on input to IOCTL_USB_USER_REQUEST, and provides buffer and status information on output.

`UnicodeName`

Contains a Unicode string of type USB_UNICODE_NAME that reports the host controller's driverkey name.

Remarks

The **USBUSER_CONTROLLER_UNICODE_NAME** structure is used in conjunction with the USBUSER_GET_CONTROLLER_DRIVER_KEY user-mode request. For a description of this request, see IOCTL_USB_USER_REQUEST.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Structures

USBUSER_REQUEST_HEADER

USB_UNICODE_NAME

USBUSER_GET_DRIVER_VERSION structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USBUSER_GET_DRIVER_VERSION** structure is used with the IOCTL_USB_USER_REQUEST I/O control request to read driver and interface version information.

Syntax

```
typedef struct _USBUSER_GET_DRIVER_VERSION {
  USBUSER_REQUEST_HEADER        Header;
  USB_DRIVER_VERSION_PARAMETERS Parameters;
} USBUSER_GET_DRIVER_VERSION, *PUSBUSER_GET_DRIVER_VERSION;
```

Members

`Header`

A USBUSER_REQUEST_HEADER structure that specifies the user-mode request on input to IOCTL_USB_USER_REQUEST and provides buffer and status information on output.

`Parameters`

A USB_DRIVER_VERSION_PARAMETERS structure that specifies the parameters that are associated with this request.

Remarks

The **USBUSER_GET_DRIVER_VERSION** structure is used with the USBUSER_GET_USB_DRIVER_VERSION user-mode request. For more information about this request, see IOCTL_USB_USER_REQUEST.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Structures

USBUSER_REQUEST_HEADER

USB_DRIVER_VERSION_PARAMETERS

USBUSER_PASS_THRU_REQUEST structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USBUSER_PASS_THRU_REQUEST** structure is used in conjunction with the IOCTL_USB_USER_REQUEST I/O control request to send a vendor-specific command to the host controller miniport driver.

Syntax

```
typedef struct _USBUSER_PASS_THRU_REQUEST {
  USBUSER_REQUEST_HEADER   Header;
  USB_PASS_THRU_PARAMETERS PassThru;
} USBUSER_PASS_THRU_REQUEST, *PUSBUSER_PASS_THRU_REQUEST;
```

Members

`Header`

Contains a structure of type USBUSER_REQUEST_HEADER that specifies the user-mode request on input to IOCTL_USB_USER_REQUEST, and provides buffer and status information on output.

`PassThru`

Contains a structure of type USB_PASS_THRU_PARAMETERS that specifies the parameters associated with this request.

Remarks

The **USBUSER_PASS_THRU_REQUEST** structure is used in conjunction with the USBUSER_PASS_THRU user-mode request. For a description of this request, see IOCTL_USB_USER_REQUEST.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Structures

USBUSER_POWER_INFO_REQUEST structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USBUSER_POWER_INFO_REQUEST** structure is used in conjunction with the IOCTL_USB_USER_REQUEST I/O control request to retrieve power policy information concerning the relationship of a specific system state to the power state of the host controller and the root hub.

Syntax

```
typedef struct _USBUSER_POWER_INFO_REQUEST {
  USBUSER_REQUEST_HEADER Header;
  USB_POWER_INFO         PowerInformation;
} USBUSER_POWER_INFO_REQUEST, *PUSBUSER_POWER_INFO_REQUEST;
```

Members

`Header`

Contains a structure of type USBUSER_REQUEST_HEADER that specifies the user-mode request on input to IOCTL_USB_USER_REQUEST, and provides buffer and status information on output.

`PowerInformation`

Contains a structure of type USB_POWER_INFO that specifies the parameters associated with this request.

Remarks

The **USBUSER_POWER_INFO_REQUEST** structure is used in conjunction with the USBUSER_GET_POWER_STATE_MAP user-mode request. For a description of this request, see IOCTL_USB_USER_REQUEST.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Structures

USBUSER_REQUEST_HEADER structure (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **USBUSER_REQUEST_HEADER** structure is used with the IOCTL_USB_USER_REQUEST I/O control request to send a user-mode request to the USB host controller driver.

Syntax

```
typedef struct _USBUSER_REQUEST_HEADER {
  ULONG               UsbUserRequest;
  USB_USER_ERROR_CODE UsbUserStatusCode;
  ULONG               RequestBufferLength;
  ULONG               ActualBufferLength;
} USBUSER_REQUEST_HEADER, *PUSBUSER_REQUEST_HEADER;
```

Members

`UsbUserRequest`

The user-mode request. For a list and description of possible values for this member, see IOCTL_USB_USER_REQUEST.

`UsbUserStatusCode`

The status code that is returned by port driver.

`RequestBufferLength`

The size, in bytes, of the data buffer. The same buffer is used for both input and output.

`ActualBufferLength`

The size, in bytes, of the data that is retrieved by the request.

Remarks

The **USBUSER_REQUEST_HEADER** structure is used with the IOCTL_USB_USER_REQUEST I/O control request to send a user-mode request to the USB port driver.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Structures

WDMUSB_POWER_STATE enumeration (usbuser.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WDMUSB_POWER_STATE** enumeration indicates the power state of a host controller or root hub.

Syntax

```
typedef enum _WDMUSB_POWER_STATE {
  WdmUsbPowerNotMapped,
  WdmUsbPowerSystemUnspecified,
  WdmUsbPowerSystemWorking,
  WdmUsbPowerSystemSleeping1,
  WdmUsbPowerSystemSleeping2,
  WdmUsbPowerSystemSleeping3,
  WdmUsbPowerSystemHibernate,
  WdmUsbPowerSystemShutdown,
  WdmUsbPowerDeviceUnspecified,
  WdmUsbPowerDeviceD0,
  WdmUsbPowerDeviceD1,
  WdmUsbPowerDeviceD2,
  WdmUsbPowerDeviceD3
} WDMUSB_POWER_STATE;
```

Constants

NAME	DESCRIPTION
WdmUsbPowerNotMapped	Power state information is not mapped.
WdmUsbPowerSystemUnspecified	Power state information is not available.
WdmUsbPowerSystemWorking	The system is in the working state.
WdmUsbPowerSystemSleeping1	The system is in the S1 power state.
WdmUsbPowerSystemSleeping2	The system is in the S2 power state.
WdmUsbPowerSystemSleeping3	The system is in the S3 power state.
WdmUsbPowerSystemHibernate	The system is hibernating.
WdmUsbPowerSystemShutdown	The system is shutdown.
WdmUsbPowerDeviceUnspecified	A device is not specified.
WdmUsbPowerDeviceD0	The host controller is in the D0 power state.
WdmUsbPowerDeviceD1	The host controller is in the D1 power state.
WdmUsbPowerDeviceD2	The host controller is in the D2 power state.

NAME	DESCRIPTION
WdmUsbPowerDeviceD3	The host controller is in the D3 power state.

Remarks

The USB stack uses the **WDMUSB_POWER_STATE** enumeration to report the power state of the host controller after receiving a USBUSER_GET_POWER_STATE_MAP request. For more information about this request, see IOCTL_USB_USER_REQUEST.

Requirements

Header	usbuser.h (include Usbuser.h)

See also

IOCTL_USB_USER_REQUEST

USB Constants and Enumerations

winusb.h header

2/1/2021 • 2 minutes to read • Edit Online

This header is used by USB driver reference. For more information, see:

- USB driver reference winusb.h contains the following programming interfaces:

Functions

TITLE	DESCRIPTION
WinUsb_AbortPipe	The WinUsb_AbortPipe function aborts all of the pending transfers for a pipe. This is a synchronous operation.
WinUsb_ControlTransfer	The WinUsb_ControlTransfer function transmits control data over a default control endpoint.
WinUsb_FlushPipe	The WinUsb_FlushPipe function discards any data that is cached in a pipe. This is a synchronous operation.
WinUsb_Free	The WinUsb_Free function releases all of the resources that WinUsb_Initialize allocated. This is a synchronous operation.
WinUsb_GetAdjustedFrameNumber	The WinUsb_GetAdjustedFrameNumber function computes what the current USB frame number should be based on the frame number value and timestamp.
WinUsb_GetAssociatedInterface	The WinUsb_GetAssociatedInterface function retrieves a handle for an associated interface. This is a synchronous operation.
WinUsb_GetCurrentAlternateSetting	The WinUsb_GetCurrentAlternateSetting function gets the current alternate interface setting for an interface. This is a synchronous operation.
WinUsb_GetCurrentFrameNumber	The WinUsb_GetCurrentFrameNumber function gets the current frame number for the bus.
WinUsb_GetCurrentFrameNumberAndQpc	The WinUsb_GetCurrentFrameNumberAndQpc function retrieves the system query performance counter (QPC) value synchronized with the frame and microframe.
WinUsb_GetDescriptor	The WinUsb_GetDescriptor function returns the requested descriptor. This is a synchronous operation.
WinUsb_GetOverlappedResult	The WinUsb_GetOverlappedResult function retrieves the results of an overlapped operation on the specified file.
WinUsb_GetPipePolicy	The WinUsb_GetPipePolicy function retrieves the policy for a specific pipe associated with an endpoint on the device. This is a synchronous operation.

TITLE	DESCRIPTION
WinUsb_GetPowerPolicy	The WinUsb_GetPowerPolicy function retrieves the power policy for a device. This is a synchronous operation.
WinUsb_Initialize	The WinUsb_Initialize function creates a WinUSB handle for the device specified by a file handle.
WinUsb_QueryDeviceInformation	The WinUsb_QueryDeviceInformation function gets information about the physical device that is associated with a WinUSB interface handle.
WinUsb_QueryInterfaceSettings	The WinUsb_QueryInterfaceSettings function retrieves the interface descriptor for the specified alternate interface settings for a particular interface handle.
WinUsb_QueryPipe	The WinUsb_QueryPipe function retrieves information about the specified endpoint and the associated pipe for an interface.
WinUsb_QueryPipeEx	The WinUsb_QueryPipeEx function retrieves extended information about the specified endpoint and the associated pipe for an interface.
WinUsb_ReadIsochPipe	The WinUsb_ReadIsochPipe function reads data from an isochronous OUT endpoint.
WinUsb_ReadIsochPipeAsap	The WinUsb_ReadIsochPipeAsap function submits a request that reads data from an isochronous OUT endpoint.
WinUsb_ReadPipe	The WinUsb_ReadPipe function reads data from the specified pipe.
WinUsb_RegisterIsochBuffer	The WinUsb_RegisterIsochBuffer function registers a buffer to be used for isochronous transfers.
WinUsb_ResetPipe	The WinUsb_ResetPipe function resets the data toggle and clears the stall condition on a pipe.
WinUsb_SetCurrentAlternateSetting	The WinUsb_SetCurrentAlternateSetting function sets the alternate setting of an interface.
WinUsb_SetPipePolicy	The WinUsb_SetPipePolicy function sets the policy for a specific pipe associated with an endpoint on the device. This is a synchronous operation.
WinUsb_SetPowerPolicy	The WinUsb_SetPowerPolicy function sets the power policy for a device.
WinUsb_StartTrackingForTimeSync	The WinUsb_StartTrackingForTimeSync function starts the time synchronization feature in the USB driver stack that gets the associated system QPC time for USB bus frames and microframes.

TITLE	DESCRIPTION
WinUsb_StopTrackingForTimeSync	The WinUsb_StopTrackingForTimeSync function tops the time synchronization feature in the USB driver stack that gets the associated system QPC time for USB bus frames and microframes.
WinUsb_UnregisterIsochBuffer	The WinUsb_UnregisterIsochBuffer function releases all of the resources that WinUsb_RegisterIsochBuffer allocated for isochronous transfers. This is a synchronous operation.
WinUsb_WriteIsochPipe	The WinUsb_WriteIsochPipe function writes the contents of a caller-supplied buffer to an isochronous OUT endpoint, starting on a specified frame number.
WinUsb_WriteIsochPipeAsap	The WinUsb_WriteIsochPipeAsap submits a request for writing the contents of a buffer to an isochronous OUT endpoint.
WinUsb_WritePipe	The WinUsb_WritePipe function writes data to a pipe.

Structures

TITLE	DESCRIPTION
WINUSB_SETUP_PACKET	The WINUSB_SETUP_PACKET structure describes a USB setup packet.

WinUsb_AbortPipe function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_AbortPipe** function aborts all of the pending transfers for a pipe. This is a synchronous operation.

Syntax

```
BOOL WinUsb_AbortPipe(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  UCHAR                   PipeID
);
```

Parameters

`InterfaceHandle`

An opaque handle to an interface that contains the endpoint with which the pipe is associated.

To abort transfers on the pipe associated with the endpoint in the first interface, use the handle returned by WinUsb_Initialize. For all other interfaces, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`PipeID`

The identifier (ID) of the control pipe. The *PipeID* parameter is an 8-bit value that consists of a 7-bit address and a direction bit. This parameter corresponds to the **bEndpointAddress** field in the endpoint descriptor.

Return value

WinUsb_AbortPipe returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib

DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Free

WinUsb_Initialize

WinUsb_ControlTransfer function (winusb.h)

2/1/2021 • 3 minutes to read • Edit Online

The **WinUsb_ControlTransfer** function transmits control data over a default control endpoint.

Syntax

```
BOOL WinUsb_ControlTransfer(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  WINUSB_SETUP_PACKET     SetupPacket,
  PUCHAR                  Buffer,
  ULONG                   BufferLength,
  PULONG                  LengthTransferred,
  LPOVERLAPPED            Overlapped
);
```

Parameters

`InterfaceHandle`

An opaque handle to an interface in the selected configuration.

To specify the recipient of a control request as the entire device or the first interface, use the handle returned by WinUsb_Initialize. For all other interfaces, obtain the handle to the target interface by calling WinUsb_GetAssociatedInterface, and then call **WinUsb_ControlTransfer** by specifying the obtained interface handle.

`SetupPacket`

The 8-byte setup packet of type WINUSB_SETUP_PACKET.

`Buffer`

A caller-allocated buffer that contains the data to transfer. The length of this buffer must not exceed 4KB.

`BufferLength`

The number of bytes to transfer, not including the setup packet. This number must be less than or equal to the size, in bytes, of *Buffer*.

`LengthTransferred`

A pointer to a ULONG variable that receives the actual number of transferred bytes. If the application does not expect any data to be transferred during the data phase (*BufferLength* is zero), *LengthTransferred* can be **NULL**.

`Overlapped`

An optional pointer to an OVERLAPPED structure, which is used for asynchronous operations. If this parameter is specified, **WinUsb_ControlTransfer** immediately returns, and the event is signaled when the operation is complete. If *Overlapped* is not supplied, the **WinUsb_ControlTransfer** function transfers data synchronously.

Return value

WinUsb_ControlTransfer returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and

the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return one of the following error codes.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.
ERROR_NOT_ENOUGH_MEMORY	Indicates that there is insufficient memory to perform the operation.

Remarks

A control request is always sent by the host to the default endpoint of a USB device but the recipient of the request can be the entire device, an interface, or an endpoint in the selected alternate setting. In the **WinUsb_ControlTransfer** call, the application must indicate the recipient through two parameters: *InterfaceHandle* and *SetupPacket*.

If the recipient of a control request is the entire device, the first interface, or any endpoint in that interface, the application must use the handle returned by WinUsb_Initialize. If the recipient is any other interface or its endpoint, then the application must obtain the WinUSB handle that is associated with the target interface by calling WinUsb_GetAssociatedInterface, and then call **WinUsb_ControlTransfer** by specifying the obtained interface handle.

As per Section 9.3 of the official USB specification, the setup token of a control transfer contains information about the request. For a WinUSB application, that setup token is described by using the WINUSB_SETUP_PACKET structure.

Within the setup token, **bmRequestType** and **wIndex** fields are used to indicate the recipient of the request. Those fields correspond to **RequestType** and **Index** members of WINUSB_SETUP_PACKET, respectively.

The lowest two bits of **RequestType** indicate the recipient of the request. The recipient can be the device, an interface, endpoint, or other (for vendor request). Depending on the recipient, the lower byte of **Index** indicates the device-defined index of the recipient. The value of **Index** depends on the type of request. For example, for standard control requests, the value is 0, or indicates the interface or endpoint number. For certain types of standard requests, such as a GET_DESCRIPTOR request to obtain a string descriptor, the **Index** value indicates the Language ID.

If the recipient is the device, the application must set **RequestType** and **Index** values. The lowest two bits of **RequestType** value must be 0. The lower byte of **Index** value depends on the type of request. The *InterfaceHandle* must be the WinUSB handle returned by WinUsb_Initialize.

If the recipient of the request is an interface, the application must set the lowest two bits of **RequestType** to 0x01. The application is not required to set the lower byte of **Index** for any type of request. For standard, class, and vendor requests, Winusb.sys sets the value to the interface number of the target interface. The *InterfaceHandle* must be associated with the target interface. The application can obtain that handle by calling WinUsb_GetAssociatedInterface.

If the recipient is an endpoint, the application must set the lowest two bits of **RequestType** to 0x02 and lower byte of Index to the endpoint address. In this case, *InterfaceHandle* is associated with the interface that contains the endpoint. The application can obtain that handle by calling WinUsb_GetAssociatedInterface.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WINUSB_SETUP_PACKET

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_FlushPipe function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_FlushPipe** function discards any data that is cached in a pipe. This is a synchronous operation.

Syntax

```
BOOL WinUsb_FlushPipe(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  UCHAR                   PipeID
);
```

Parameters

`InterfaceHandle`

An opaque handle to the interface with which the specified pipe's endpoint is associated. To clear data in a pipe that is associated with the endpoint on the first (default) interface, use the handle returned by WinUsb_Initialize. For all other interfaces, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`PipeID`

The identifier (ID) of the control pipe. The *PipeID* parameter is an 8-bit value that consists of a 7-bit address and a direction bit. This parameter corresponds to the **bEndpointAddress** field in the endpoint descriptor.

Return value

WinUsb_FlushPipe returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_Free function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_Free** function releases all of the resources that WinUsb_Initialize allocated. This is a synchronous operation.

Syntax

```
BOOL WinUsb_Free(
  WINUSB_INTERFACE_HANDLE InterfaceHandle
);
```

Parameters

`InterfaceHandle`

An opaque handle to an interface in the selected configuration. That handle must be created by a previous call to WinUsb_Initialize or WinUsb_GetAssociatedInterface.

Return value

WinUsb_Free returns **TRUE**.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_GetAdjustedFrameNumber function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_GetAdjustedFrameNumber** function computes what the current USB frame number should be based on the frame number value and timestamp.

Syntax

```
BOOL WinUsb_GetAdjustedFrameNumber(
  PULONG        CurrentFrameNumber,
  LARGE_INTEGER TimeStamp
);
```

Parameters

`CurrentFrameNumber`

The frame number to be adjusted.

`TimeStamp`

The timestamp recorded at the time the frame number was returned.

Return value

WinUsb_GetAdjustedFrameNumber returns TRUE if the operation succeeds. Otherwise this function returns FALSE, and the caller can retrieve the logged error by calling **GetLastError**.

Requirements

Minimum supported client	Windows 8.1
Minimum supported server	Windows Server 2012 R2
Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

Send USB isochronous transfers from a WinUSB desktop app

WinUsb_GetAssociatedInterface function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_GetAssociatedInterface** function retrieves a handle for an associated interface. This is a synchronous operation.

Syntax

```
BOOL WinUsb_GetAssociatedInterface(
  WINUSB_INTERFACE_HANDLE  InterfaceHandle,
  UCHAR                    AssociatedInterfaceIndex,
  PWINUSB_INTERFACE_HANDLE AssociatedInterfaceHandle
);
```

Parameters

`InterfaceHandle`

An opaque handle to the first (default) interface on the device, which is returned by WinUsb_Initialize.

`AssociatedInterfaceIndex`

An index that specifies the associated interface to retrieve. A value of 0 indicates the first associated interface, a value of 1 indicates the second associated interface, and so on.

`AssociatedInterfaceHandle`

A handle for the associated interface. Callers must pass this interface handle to WinUSB Functions exposed by Winusb.dll. To close this handle, call WinUsb_Free.

Return value

WinUsb_GetAssociatedInterface returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

RETURN CODE	DESCRIPTION
ERROR_ALREADY_EXISTS	WinUsb_GetAssociatedInterface has already returned a handle for the interface that *AssociatedInterfaceIndex* specifies.
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.
ERROR_INVALID_PARAMETER	The passed *AssociatedInterfaceIndex* value failed an integer overflow check.
ERROR_NO_MORE_ITEMS	An interface does not exist for the specified *AssociatedInterfaceIndex* value.

ERROR_NOT_ENOUGH_MEMORY	Indicates that there is insufficient memory to perform the operation.

Remarks

The **WinUsb_GetAssociatedInterface** routine retrieves an opaque handle.

The *first associated interface* is the interface that immediately follows the interface whose handle the WinUsb_Initialize routine retrieves.

The handle that **WinUsb_GetAssociatedInterface** returns must be released by calling WinUsb_Free.

Callers of **WinUsb_GetAssociatedInterface**can retrieve only one handle for each interface. If a caller attempts to retrieve more than one handle for the same interface, the routine will fail with an error of ERROR_ALREADY_EXISTS.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_GetCurrentAlternateSetting function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_GetCurrentAlternateSetting** function gets the current alternate interface setting for an interface. This is a synchronous operation.

Syntax

```
BOOL WinUsb_GetCurrentAlternateSetting(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  PUCHAR                  SettingNumber
);
```

Parameters

`InterfaceHandle`

An opaque handle to an interface in the selected configuration. To get the current alternate setting in the first (default) interface on the device, use the interface handle returned by WinUsb_Initialize. For all other interfaces, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`SettingNumber`

A pointer to an unsigned character that receives an integer that indicates the current alternate setting.

Return value

WinUsb_GetCurrentAlternateSetting returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_GetCurrentFrameNumber function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_GetCurrentFrameNumber** function gets the current frame number for the bus.

Syntax

```
BOOL WinUsb_GetCurrentFrameNumber(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  PULONG                  CurrentFrameNumber,
  LARGE_INTEGER           *TimeStamp
);
```

Parameters

`InterfaceHandle`

The handle to the device that **CreateFile** returned.

`CurrentFrameNumber`

The current frame number value.

`TimeStamp`

The time stamp value when the current frame was read.

Return value

WinUsb_GetCurrentFrameNumber returns TRUE if the operation succeeds. Otherwise this function returns FALSE, and the caller can retrieve the logged error by calling **GetLastError**.

Remarks

The caller may compare the PerformanceCount with the value returned by the Win32 function QueryPerformanceCounter to determine if there has been a delay in transitioning back to user-mode after the frame number was read. The caller can then adjust the starting frame number as needed.

Requirements

Minimum supported client	Windows 8.1
Minimum supported server	Windows Server 2012 R2
Target Platform	Universal
Header	winusb.h (include Winusb.h)

Library	Winusb.lib
DLL	Winusb.dll

See also

Send USB isochronous transfers from a WinUSB desktop app

WinUSB Functions

WinUsb_GetCurrentFrameNumberAndQpc function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_GetCurrentFrameNumberAndQpc** function retrieves the system query performance counter (QPC) value synchronized with the frame and microframe.

Syntax

```
BOOL WinUsb_GetCurrentFrameNumberAndQpc(
  WINUSB_INTERFACE_HANDLE                          InterfaceHandle,
  PUSB_FRAME_NUMBER_AND_QPC_FOR_TIME_SYNC_INFORMATION FrameQpcInfo
);
```

Parameters

`InterfaceHandle`

An opaque handle retrieved in the previous call to WinUsb_Initialize.

`FrameQpcInfo`

A pointer to a USB_FRAME_NUMBER_AND_QPC_FOR_TIME_SYNC_INFORMATION structure. On output, **CurrentQueryPerformanceCounter** set to the system QPC value (in microseconds) predicted by the USB driver stack. Optionally, on input, the caller can specify a frame and microframe number for which to retrieve the QPC value.

On output, the **QueryPerformanceCounterAtInputFrameOrMicroFrame** member is set to the QPC value for that frame or microframe.

Return value

WinUsb_GetCurrentFrameNumberAndQpc returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return one of the following error codes.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* or *FrameQpcInfo* parameter.

Requirements

Minimum supported client	Windows 10
Minimum supported server	Windows Server 2016

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_StartTrackingForTimeSync

WinUsb_GetDescriptor function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_GetDescriptor** function returns the requested descriptor. This is a synchronous operation.

Syntax

```
BOOL WinUsb_GetDescriptor(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  UCHAR                   DescriptorType,
  UCHAR                   Index,
  USHORT                  LanguageID,
  PUCHAR                  Buffer,
  ULONG                   BufferLength,
  PULONG                  LengthTransferred
);
```

Parameters

`InterfaceHandle`

An opaque handle to an interface in the selected configuration.

To retrieve the device or configuration descriptor, use the handle returned by WinUsb_Initialize.

To retrieve the interface descriptor of the first interface, use the handle returned by WinUsb_Initialize.

To retrieve the endpoint descriptor of an endpoint in the first interface, use the handle returned by WinUsb_Initialize.

To retrieve descriptors of all other interfaces and their related endpoints, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`DescriptorType`

A value that specifies the type of descriptor to return. This parameter corresponds to the **bDescriptorType** field of a standard device descriptor, whose values are described in the *Universal Serial Bus* specification. Some of these values are listed in the description of the **DescriptorType** member of the _URB_CONTROL_DESCRIPTOR_REQUEST structure.

`Index`

The descriptor index. For an explanation of the descriptor index, see the *Universal Serial Bus* specification (www.usb.org).

`LanguageID`

A value that specifies the language identifier, if the requested descriptor is a string descriptor.

`Buffer`

A caller-allocated buffer that receives the requested descriptor.

`BufferLength`

The length, in bytes, of *Buffer*.

`LengthTransferred`

The number of bytes that were copied into *Buffer*.

Return value

WinUsb_GetDescriptor returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.

Remarks

If the output buffer pointed to by the *Buffer* parameter is large enough, **WinUsb_GetDescriptor** creates a copy of the specified descriptor into the output buffer. No data is copied if the buffer is not large enough to hold descriptor data. The descriptor is created during the WinUsb_Initialize call or it may be retrieved at this point from the device.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

_URB_CONTROL_DESCRIPTOR_REQUEST

WinUsb_GetOverlappedResult function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_GetOverlappedResult** function retrieves the results of an overlapped operation on the specified file.

Syntax

```
BOOL WinUsb_GetOverlappedResult(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  LPOVERLAPPED            lpOverlapped,
  LPDWORD                 lpNumberOfBytesTransferred,
  BOOL                    bWait
);
```

Parameters

`InterfaceHandle`

An opaque handle to the first interface on the device, which is returned by WinUsb_Initialize.

`lpOverlapped`

A pointer to an **OVERLAPPED** structure that was specified when the overlapped operation was started.

`lpNumberOfBytesTransferred`

A pointer to a variable that receives the number of bytes that were actually transferred by a read or write operation.

`bWait`

If this parameter is **TRUE**, the function does not return until the operation has been completed. If this parameter is **FALSE** and the operation is still pending, the function returns **FALSE** and the **GetLastError** function returns ERROR_IO_INCOMPLETE.

Return value

If the function succeeds, the return value is any number other than zero. If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

This function is like the Win32 API routine, **GetOverlappedResult**, with one difference—instead of passing a file handle that is returned from **CreateFile**, the caller passes an interface handle that is returned from WinUsb_Initialize. The caller can use either API routine, if the appropriate handle is passed. The **WinUsb_GetOverlappedResult** function extracts the file handle from the interface handle and then calls **GetOverlappedResult**.

The results that are reported by the **WinUsb_GetOverlappedResult** function are those from the specified handle's last overlapped operation to which the specified **OVERLAPPED** structure was provided, and for which the operation's results were pending. A pending operation is indicated when the function that started the

operation returns **FALSE**, and the **GetLastError** routine returns ERROR_IO_PENDING. When an I/O operation is pending, the function that started the operation resets the **hEvent** member of the **OVERLAPPED** structure to the nonsignaled state. Then when the pending operation has been completed, the system sets the event object to the signaled state.

The caller can specify that an event object is manually reset in the **OVERLAPPED** structure. If an automatic reset event object is used, the event handle must not be specified in any other wait operation in the interval between starting the overlapped operation and the call to **WinUsb_GetOverlappedResult**. For example, the event object is sometimes specified in one of the wait routines to wait for the operation to be completed. When the wait routine returns, the system sets an auto-reset event's state to nonsignaled, and a successive call to **WinUsb_GetOverlappedResult** with the *bWait* parameter set to **TRUE** causes the function to be blocked indefinitely.

If the *bWait* parameter is **TRUE**, **WinUsb_GetOverlappedResult** determines whether the pending operation has been completed by waiting for the event object to be in the signaled state.

If the **hEvent** member of the **OVERLAPPED** structure is **NULL**, the system uses the state of the file handle to signal when the operation has been completed. Do not use file handles for this purpose. It is better to use an event object because of the confusion that can occur when multiple concurrent overlapped operations are performed on the same file. In this situation, you cannot know which operation caused the state of the object to be signaled.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

_URB_CONTROL_DESCRIPTOR_REQUEST

WinUsb_GetPipePolicy function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_GetPipePolicy** function retrieves the policy for a specific pipe associated with an endpoint on the device. This is a synchronous operation.

Syntax

```
BOOL WinUsb_GetPipePolicy(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  UCHAR                   PipeID,
  ULONG                   PolicyType,
  PULONG                  ValueLength,
  PVOID                   Value
);
```

Parameters

`InterfaceHandle`

An opaque handle to an interface that contains the endpoint with which the pipe is associated.

To query the pipe associated with the endpoint in the first interface, use the handle returned by WinUsb_Initialize. For all other interfaces, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`PipeID`

An 8-bit value that consists of a 7-bit address and a direction bit. This parameter corresponds to the **bEndpointAddress** field in the endpoint descriptor.

`PolicyType`

A **ULONG** variable that specifies the policy parameter to retrieve. The current value for the policy parameter is retrieved the *Value* parameter. For information about the behavior of the pipe policies, see WinUSB Functions for Pipe Policy Modification.

`ValueLength`

A pointer to the size, in bytes, of the buffer that *Value* points to. On output, *ValueLength* receives the size, in bytes, of the data that was copied into the *Value*buffer.

`Value`

A pointer to a buffer that receives the specified pipe policy value.

Return value

WinUsb_GetPipePolicy returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION

ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUSB Functions for Pipe Policy Modification

WinUsb_Initialize

WinUsb_ReadPipe

WinUsb_WritePipe

WinUsb_GetPowerPolicy function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_GetPowerPolicy** function retrieves the power policy for a device. This is a synchronous operation.

Syntax

```
BOOL WinUsb_GetPowerPolicy(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  ULONG                   PolicyType,
  PULONG                  ValueLength,
  PVOID                   Value
);
```

Parameters

`InterfaceHandle`

An opaque handle to the first interface on the device, which is returned by WinUsb_Initialize.

`PolicyType`

A value that specifies the power policy parameter to retrieve in *Value*. The following table describes symbolic constants that are defined in *Winusbio.h*.

POLICY TYPE	DESCRIPTION
AUTO_SUSPEND (0x81)	If the caller specifies a power policy of AUTO_SUSPEND, **WinUsb_GetPowerPolicy** returns the value of the auto suspend policy parameter in the *Value* parameter. If *Value* is **TRUE** (that is, nonzero), the USB stack suspends the device when no transfers are pending or the only transfers pending are IN transfers on an interrupt or bulk endpoint. The value of the **DefaultIdleState** registry value determines the default value of the auto suspend policy parameter. The *Value* parameter must point to a UCHAR variable.
SUSPEND_DELAY (0x83)	If the caller specifies a power policy of SUSPEND_DELAY, **WinUsb_GetPowerPolicy** returns the value of the suspend delay policy parameter in *Value*. The suspend delay policy parameter specifies the minimum amount of time, in milliseconds, that the WinUSB driver must wait after any transfer before it can suspend the device. *Value* must point to a ULONG variable.

`ValueLength`

A pointer to the size of the buffer that *Value*. On output, *ValueLength* receives the size of the data that was copied into the *Value* buffer.

`Value`

A buffer that receives the specified power policy parameter. For more information, see *PolicyType*.

Return value

WinUsb_GetPowerPolicy returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUSB Power Management

WinUsb_Initialize

WinUsb_Initialize function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_Initialize** function creates a WinUSB handle for the device specified by a file handle.

Syntax

```
BOOL WinUsb_Initialize(
  HANDLE                   DeviceHandle,
  PWINUSB_INTERFACE_HANDLE InterfaceHandle
);
```

Parameters

`DeviceHandle`

The handle to the device that **CreateFile** returned. WinUSB uses overlapped I/O, so FILE_FLAG_OVERLAPPED must be specified in the *dwFlagsAndAttributes* parameter of **CreateFile** call for *DeviceHandle* to have the characteristics necessary for **WinUsb_Initialize** to function properly.

`InterfaceHandle`

Receives an opaque handle to the first (default) interface on the device. This handle is required by other WinUSB routines that perform operations on the default interface. To release the handle, call the WinUSB_Free function.

Return value

WinUsb_Initialize returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** or an invalid handle in the *DeviceHandle* parameter; FILE_FLAG_OVERLAPPED was not set in the file handle.
ERROR_NOT_ENOUGH_MEMORY	Indicates that there is insufficient memory to perform the operation.
ERROR_BAD_DEVICE	Indicates that the default interface descriptor could not be found for the device.

Remarks

When **WinUsb_Initialize** is called, the policy settings of the interface are reset to the default values.

The **WinUsb_Initialize** call queries the underlying USB stack for various descriptors and allocates enough memory to store the retrieved descriptor data.

WinUsb_Initialize first retrieves the device descriptor and then gets the associated configuration descriptor. From the configuration descriptor, the call derives the associated interface descriptors and stores them in an array. The interfaces in the array are identified by zero-based indexes. An index value of 0 indicates the first interface (the default interface), a value of 1 indicates the second associated interface, and so on. **WinUsb_Initialize** parses the default interface descriptor for the endpoint descriptors and caches information such as the associated pipes or state specific data. The handle received in the *InterfaceHandle* parameter is a pointer to the memory block allocated for the first interface in the array.

If an application wants to use another interface on the device, it must call WinUsb_GetAssociatedInterface, specify the index of the interface, and retrieve a handle to the memory block allocated for the specified interface.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUSB_Free

WinUsb_QueryDeviceInformation function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_QueryDeviceInformation** function gets information about the physical device that is associated with a WinUSB interface handle.

Syntax

```
BOOL WinUsb_QueryDeviceInformation(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  ULONG                   InformationType,
  PULONG                  BufferLength,
  PVOID                   Buffer
);
```

Parameters

`InterfaceHandle`

An opaque handle to the first interface on the device, which is returned by WinUsb_Initialize.

`InformationType`

A value that specifies which interface information value to retrieve.

On input, *InformationType* must have the following value: DEVICE_SPEED (0x01).

`BufferLength`

The maximum number of bytes to read. This number must be less than or equal to the size, in bytes, of *Buffer*. On output, *BufferLength* is set to the actual number of bytes that were copied into *Buffer*.

`Buffer`

A caller-allocated buffer that receives the requested value.

If *InformationType* is DEVICE_SPEED, on successful return, *Buffer* indicates the operating speed of the device. 0x03 indicates high-speed or higher; 0x01 indicates full-speed or lower.

Return value

WinUsb_QueryDeviceInformation returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_QueryInterfaceSettings function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_QueryInterfaceSettings** function retrieves the interface descriptor for the specified alternate interface settings for a particular interface handle.

Syntax

```
BOOL WinUsb_QueryInterfaceSettings(
  WINUSB_INTERFACE_HANDLE   InterfaceHandle,
  UCHAR                     AlternateInterfaceNumber,
  PUSB_INTERFACE_DESCRIPTOR UsbAltInterfaceDescriptor
);
```

Parameters

`InterfaceHandle`

An opaque handle to an interface in the selected configuration.

To retrieve the settings of the first interface, use the handle returned by WinUsb_Initialize. For all other interfaces, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`AlternateInterfaceNumber`

A value that indicates which alternate settings to return. A value of 0 indicates the first alternate setting, a value of 1 indicates the second alternate setting, and so on.

`UsbAltInterfaceDescriptor`

A pointer to a caller-allocated USB_INTERFACE_DESCRIPTOR structure that contains information about the interface that *AlternateSettingNumber* specified.

Return value

WinUsb_QueryInterfaceSettings returns **TRUE** if the operation succeeds. Otherwise, it returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.
ERROR_NO_MORE_ITEMS	The specified alternate interface was not found.

Remarks

WinUsb_QueryInterfaceSettings parses the configuration descriptor previously retrieved by

WinUsb_Initialize. For more information, see the Remarks section for **WinUsb_Initialize**.

The **WinUsb_QueryInterfaceSettings** call searches the interface array for the alternate interface specified by the interface index passed by the caller in the *AlternateSettingNumber*. If the specified interface is found, the function populates the caller-allocated USB_INTERFACE_DESCRIPTOR structure. If the specified interface is not found, then the call fails with the ERROR_NO_MORE_ITEMS code.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

USB_INTERFACE_DESCRIPTOR

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_QueryPipe function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_QueryPipe** function retrieves information about the specified endpoint and the associated pipe for an interface.

Syntax

```
BOOL WinUsb_QueryPipe(
  WINUSB_INTERFACE_HANDLE  InterfaceHandle,
  UCHAR                    AlternateInterfaceNumber,
  UCHAR                    PipeIndex,
  PWINUSB_PIPE_INFORMATION PipeInformation
);
```

Parameters

`InterfaceHandle`

An opaque handle to an interface that contains the endpoint with which the pipe is associated.

To query the pipe associated with an endpoint in the first interface, use the handle returned by WinUsb_Initialize. For all other interfaces, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`AlternateInterfaceNumber`

A value that specifies the alternate interface to return the information for.

`PipeIndex`

A value that specifies the pipe to return information about. This value is not the same as the **bEndpointAddress** field in the endpoint descriptor. A *PipeIndex* value of 0 signifies the first endpoint that is associated with the interface, a value of 1 signifies the second endpoint, and so on. *PipeIndex* must be less than the value in the **bNumEndpoints** field of the interface descriptor.

`PipeInformation`

A pointer, on output, to a caller-allocated WINUSB_PIPE_INFORMATION structure that contains pipe information.

Return value

WinUsb_QueryPipe returns **TRUE** if the operation succeeds. Otherwise, this function returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.

ERROR_INVALID_PARAMETER	The caller passed **NULL** in the *PipeInformation* parameter; interface descriptor could not be found for the handle specified in *InterfaceHandle*.
ERROR_NO_MORE_ITEMS	The value passed in the *PipeIndex* parameter is greater than the **bNumEndpoints** value of the interface descriptor; endpoint descriptor could not be found for the specified interface.

Remarks

The **WinUsb_QueryPipe** function does not retrieve information about the control pipe.

Each interface on the USB device can have multiple endpoints. To communicate with each of these endpoints, the bus driver creates pipes for each endpoint on the interface. The pipe indices are zero-based. Therefore for *n* number of endpoints, the pipes' indices are set from *n*-1. **WinUsb_QueryPipe** parses the configuration descriptor to get the interface specified by the caller. It searches the interface descriptor for the endpoint descriptor associated with the caller-specified pipe. If the endpoint is found, the function populates the caller-allocated WINUSB_PIPE_INFORMATION structure with information from the endpoint descriptor.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WINUSB_PIPE_INFORMATION

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_QueryPipeEx function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_QueryPipeEx** function retrieves extended information about the specified endpoint and the associated pipe for an interface.

Syntax

```
BOOL WinUsb_QueryPipeEx(
  WINUSB_INTERFACE_HANDLE     InterfaceHandle,
  UCHAR                       AlternateSettingNumber,
  UCHAR                       PipeIndex,
  PWINUSB_PIPE_INFORMATION_EX PipeInformationEx
);
```

Parameters

`InterfaceHandle`

An opaque handle to an interface that contains the endpoint with which the pipe is associated.

To query the pipe associated with an endpoint in the first interface, use the handle returned by WinUsb_Initialize. For all other interfaces, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`AlternateSettingNumber`

A value that specifies the alternate interface to return the information for.

`PipeIndex`

A value that specifies the pipe to return information about. This value is not the same as the **bEndpointAddress** field in the endpoint descriptor. A *PipeIndex* value of 0 signifies the first endpoint that is associated with the interface, a value of 1 signifies the second endpoint, and so on. *PipeIndex* must be less than the value in the **bNumEndpoints** field of the interface descriptor.

`PipeInformationEx`

A pointer, on output, to a caller-allocated WINUSB_PIPE_INFORMATION_EX structure that contains pipe information.

Return value

WinUsb_QueryPipeEx returns **TRUE** if the operation succeeds. Otherwise, this function returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.

ERROR_INVALID_PARAMETER	The caller passed **NULL** in the *PipeInformation* parameter; interface descriptor could not be found for the handle specified in *InterfaceHandle*.
ERROR_NO_MORE_ITEMS	The value passed in the *PipeIndex* parameter is greater than the **bNumEndpoints** value of the interface descriptor; endpoint descriptor could not be found for the specified interface.

Remarks

The **WinUsb_QueryPipeEx** function does not retrieve information about the control pipe.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

Send USB isochronous transfers from a WinUSB desktop app

WINUSB_PIPE_INFORMATION

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_ReadIsochPipe function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_ReadIsochPipe** function reads data from an isochronous OUT endpoint.

Syntax

```
BOOL WinUsb_ReadIsochPipe(
  WINUSB_ISOCH_BUFFER_HANDLE  BufferHandle,
  ULONG                       Offset,
  ULONG                       Length,
  PULONG                      FrameNumber,
  ULONG                       NumberOfPackets,
  PUSBD_ISO_PACKET_DESCRIPTOR IsoPacketDescriptors,
  LPOVERLAPPED                Overlapped
);
```

Parameters

`BufferHandle`

An opaque handle to the transfer buffer that was registered by a previous call to WinUsb_RegisterIsochBuffer.

`Offset`

Offset into the buffer relative to the start the transfer.

`Length`

Length in bytes of the transfer buffer.

`FrameNumber`

On input, indicates the starting frame number for the transfer. On output, contains the frame number of the frame that follows the last frame used in the transfer.

`NumberOfPackets`

Total number of isochronous packets required to hold the transfer buffer. Also indicates the number of elements in the array pointed to by *IsoPacketDescriptors*.

`IsoPacketDescriptors`

An array of USBD_ISO_PACKET_DESCRIPTOR structures. After the transfer completes, each element contains the status and size of the isochronous packet.

`Overlapped`

Pointer to an OVERLAPPED structure used for asynchronous operations.

Return value

WinUsb_ReadIsochPipe returns TRUE if the operation succeeds. Otherwise this function returns FALSE, and the caller can retrieve the logged error by calling **GetLastError**.

Remarks

WinUsb_ReadIsochPipe packetizes the transfer buffer so that in each 1ms interval, the host can receive the maximum bytes allowed per interval. The maximum bytes is as specified by the endpoint descriptor for full and high-speed endpoints, and endpoint companion descriptor for SuperSpeed endpoints. If the caller submits multiple read requests to stream data from the device, the transfer size should be a multiple of the maximum bytes per interval (as returned by WinUsb_QueryPipeEx) * 8 / interval.

Because of the transfer packaging used in the underlying kernel-mode interface, the lowest latency notification to an application or driver is 1ms intervals.

Requirements

Minimum supported client	Windows 8.1
Minimum supported server	Windows Server 2012 R2
Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

Send USB isochronous transfers from a WinUSB desktop app

WinUSB Functions

WinUsb_ReadIsochPipeAsap function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_ReadIsochPipeAsap** function submits a request that reads data from an isochronous OUT endpoint.

Syntax

```
BOOL WinUsb_ReadIsochPipeAsap(
  WINUSB_ISOCH_BUFFER_HANDLE  BufferHandle,
  ULONG                       Offset,
  ULONG                       Length,
  BOOL                        ContinueStream,
  ULONG                       NumberOfPackets,
  PUSBD_ISO_PACKET_DESCRIPTOR IsoPacketDescriptors,
  LPOVERLAPPED                Overlapped
);
```

Parameters

`BufferHandle`

An opaque handle to the transfer buffer that was registered by a previous call to WinUsb_RegisterIsochBuffer.

`Offset`

Offset into the buffer relative to the start the transfer.

`Length`

Length in bytes of the transfer buffer.

`ContinueStream`

Indicates that the transfer should only be submitted if it can be scheduled in the first frame after the last pending transfer.

`NumberOfPackets`

Total number of isochronous packets required to hold the transfer buffer. Also indicates the number of elements in the array pointed to by *IsoPacketDescriptors*.

`IsoPacketDescriptors`

An array of USBD_ISO_PACKET_DESCRIPTOR that receives the details of each isochronous packet in the transfer.

`Overlapped`

Pointer to an OVERLAPPED structure used for asynchronous operations.

Return value

WinUsb_ReadIsochPipeAsap returns TRUE if the operation succeeds. Otherwise this function returns FALSE, and the caller can retrieve the logged error by calling **GetLastError**.

If the caller sets *ContinueStream* to TRUE, The transfer fails if Winusb.sys is unable to schedule the transfer to continue the stream without dropping one or more frames.

Remarks

WinUsb_ReadIsochPipeAsap allows the USB driver stack to choose the starting frame number for the transfer. If one or more transfers are already pending on the endpoint, the transfer will be scheduled for the frame number immediately following the last frame number of the last currently pending transfer.

WinUsb_ReadIsochPipeAsap packetizes the transfer buffer so that in each interval, the host can receive the maximum bytes allowed per interval. The maximum bytes is as specified by the endpoint descriptor for full and high-speed endpoints, and endpoint companion descriptor for SuperSpeed endpoints. If the caller submits multiple read requests to stream data from the device, the transfer size should be a multiple of the maximum bytes per interval (as returned by WinUsb_QueryPipeEx) * 8 / interval.

Because of the transfer packaging used in the underlying kernel-mode interface, the lowest latency notification to an application or driver is 1ms intervals.

Requirements

Minimum supported client	Windows 8.1
Minimum supported server	Windows Server 2012 R2
Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

Send USB isochronous transfers from a WinUSB desktop app

WinUSB Functions

WinUsb_ReadPipe function (winusb.h)

2/1/2021 • 3 minutes to read • Edit Online

The **WinUsb_ReadPipe** function reads data from the specified pipe.

Syntax

```
BOOL WinUsb_ReadPipe(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  UCHAR                   PipeID,
  PUCHAR                  Buffer,
  ULONG                   BufferLength,
  PULONG                  LengthTransferred,
  LPOVERLAPPED            Overlapped
);
```

Parameters

`InterfaceHandle`

An opaque handle to the interface that contains the endpoint with which the pipe is associated.

To read data from the pipe associated with an endpoint in the first interface, use the handle returned by WinUsb_Initialize. For all other interfaces, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`PipeID`

PipeID corresponds to the **bEndpointAddress** field in the endpoint descriptor. For information about the layout of this field, see **Table 9-13** in "Universal Serial Bus Specification Revision 2.0" at USB Technology. In the **bEndpointAddress** field, Bit 7 indicates the direction of the endpoint: 0 for OUT; 1 for IN.

`Buffer`

A caller-allocated buffer that receives the data that is read.

`BufferLength`

The maximum number of bytes to read. This number must be less than or equal to the size, in bytes, of *Buffer*.

`LengthTransferred`

A pointer to a ULONG variable that receives the actual number of bytes that were copied into *Buffer*. For more information, see Remarks.

`Overlapped`

An optional pointer to an OVERLAPPED structure that is used for asynchronous operations. If this parameter is specified, **WinUsb_ReadPipe** returns immediately rather than waiting synchronously for the operation to complete before returning. An event is signaled when the operation is complete.

Return value

WinUsb_ReadPipe returns **TRUE** if the operation succeeds. Otherwise, this function returns **FALSE**, and the

caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.
ERROR_IO_PENDING	An overlapped I/O operation is in progress but has not completed. If the overlapped operation cannot be completed immediately, the function returns **FALSE** and the **GetLastError** function returns ERROR_IO_PENDING, indicating that the operation is executing in the background. Call WinUsb_GetOverlappedResult to check the success or failure of the operation.
ERROR_NOT_ENOUGH_MEMORY	There is insufficient memory to perform the operation.
ERROR_SEM_TIMEOUT	The read operation initiated by WinUsb_ReadPipe in the USB stack timed out before the operation could be completed.

Remarks

If the data returned by the device is greater than a maximum transfer length, WinUSB divides the request into smaller requests of maximum transfer length and submits them serially. If the transfer length is not a multiple of the endpoint's maximum packet size (retrievable through the WINUSB_PIPE_INFORMATION structure's **MaximumPacketSize** member), WinUSB increases the size of the transfer to the next multiple of **MaximumPacketSize**.

USB packet size does not factor into the transfer for a read request. If the device responds with a packet that is too large for the client buffer, the behavior of the read request corresponds to the type of policy set on the pipe. If policy type for the pipe is ALLOW_PARTIAL_READS, WinUSB adds the remaining data to the beginning of the next transfer. If ALLOW_PARTIAL_READS is not set, the read request fails. For more information about policy types, see WinUSB Functions for Pipe Policy Modification.

If an application passes **NULL** in the *Overlapped* parameter (synchronous operation), the application must make sure that *LengthTransferred* is not **NULL**, even when the read operation produces no output data.

If *Overlapped* is not **NULL** (asynchronous operation), *LengthTransferred* can be set to **NULL**. For an overlapped operation (and if *LengthTransferred* is a non-**NULL** value), the value received in *LengthTransferred* after **WinUsb_ReadPipe** returns is meaningless until the overlapped operation has completed. To retrieve the actual number of bytes read from the pipe, call WinUsb_GetOverlappedResult.

When no data is available in the endpoint (pipe is empty), **WinUsb_ReadPipe** does not return until there is data in the pipe. If an error condition occurs or the application-specified timeout expires, **WinUsb_ReadPipe** always returns FALSE. To determine the actual reason for that return value, always call **GetLastError**. For example, in these cases the **GetLastError** error value indicates the actual reason:

- If the application specified a timeout value in the pipe policy and that timeout expires, **WinUsb_ReadPipe** returns FALSE and **GetLastError** returns ERROR_SEM_TIMEOUT.
- If an error condition occurs while reading data from the pipe, **WinUsb_ReadPipe** returns FALSE and **GetLastError** returns ERROR_GEN_FAILURE.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_RegisterIsochBuffer function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_RegisterIsochBuffer** function registers a buffer to be used for isochronous transfers.

Syntax

```
BOOL WinUsb_RegisterIsochBuffer(
  WINUSB_INTERFACE_HANDLE     InterfaceHandle,
  UCHAR                       PipeID,
  PUCHAR                      Buffer,
  ULONG                       BufferLength,
  PWINUSB_ISOCH_BUFFER_HANDLE IsochBufferHandle
);
```

Parameters

`InterfaceHandle`

An opaque handle to an interface in the selected configuration. That handle must be created by a previous call to WinUsb_Initialize or WinUsb_GetAssociatedInterface.

`PipeID`

Derived from Bit 3...0 of the **bEndpointAddress** field in the endpoint descriptor.

`Buffer`

Pointer to the transfer buffer to be registered.

`BufferLength`

Length, in bytes, of the transfer buffer pointed to by *Buffer*.

`IsochBufferHandle`

Receives an opaque handle to the registered buffer. This handle is required by other WinUSB functions that perform isochronous transfers. To release the handle, call the WinUsb_UnregisterIsochBuffer function.

Return value

WinUsb_RegisterIsochBuffer returns TRUE if the operation succeeds. Otherwise this function returns FALSE, and the caller can retrieve the logged error by calling **GetLastError**.

If the caller sets *ContinueStream* to TRUE, The transfer fails if Winusb.sys is unable to schedule the transfer to continue the stream without dropping one or more frames.

Remarks

Prior to initiating isochronous transfers to or from a buffer, the caller must register the buffer with **WinUsb_RegisterIsochBuffer**. This call allows the Winusb.sys to pre-map and lock the buffer after for all subsequent transfers using the buffer.

Requirements

Minimum supported client	Windows 8.1
Minimum supported server	Windows Server 2012 R2
Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

Send USB isochronous transfers from a WinUSB desktop app

WinUSB Functions

WinUsb_UnregisterIsochBuffer

WinUsb_ResetPipe function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_ResetPipe** function resets the data toggle and clears the stall condition on a pipe.

Syntax

```
BOOL WinUsb_ResetPipe(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  UCHAR                   PipeID
);
```

Parameters

`InterfaceHandle`

An opaque handle to the interface that contains the endpoint with which the pipe is associated.

To reset a pipe associated with an endpoint in the first interface, use the handle returned by WinUsb_Initialize. For all other interfaces, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`PipeID`

The identifier (ID) of the control pipe. The *PipeID* parameter is an 8-bit value that consists in a 7-bit address and a direction bit. This parameter corresponds to the **bEndpointAddress** field in the endpoint descriptor.

Return value

WinUsb_ResetPipe returns **TRUE** if the operation succeeds. Otherwise, this function returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_SetCurrentAlternateSetting function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_SetCurrentAlternateSetting** function sets the alternate setting of an interface.

Syntax

```
BOOL WinUsb_SetCurrentAlternateSetting(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  UCHAR                   SettingNumber
);
```

Parameters

`InterfaceHandle`

An opaque handle to an interface, which defines the alternate setting to set.

To set an alternate setting in the first interface on the device, use the interface handle returned by WinUsb_Initialize. For all other interfaces, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`SettingNumber`

The value that is contained in the **bAlternateSetting** member of the USB_INTERFACE_DESCRIPTOR structure. This structure is populated by the WinUsb_QueryInterfaceSettings routine.

Return value

WinUsb_SetCurrentAlternateSetting returns **TRUE** if the operation succeeds. Otherwise, this function returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.

Remarks

WinUsb_SetCurrentAlternateSetting fails if outstanding I/O requests are present on the interface.

Requirements

Target Platform	Universal

Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

USB_INTERFACE_DESCRIPTOR

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_QueryInterfaceSettings

WinUsb_SetPipePolicy function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_SetPipePolicy** function sets the policy for a specific pipe associated with an endpoint on the device. This is a synchronous operation.

Syntax

```
BOOL WinUsb_SetPipePolicy(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  UCHAR                   PipeID,
  ULONG                   PolicyType,
  ULONG                   ValueLength,
  PVOID                   Value
);
```

Parameters

`InterfaceHandle`

An opaque handle to an interface that contains the endpoint with which the pipe is associated.

To set policy for the pipe associated with the endpoint in the first interface, use the handle returned by WinUsb_Initialize. For all other interfaces, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`PipeID`

An 8-bit value that consists of a 7-bit address and a direction bit. This parameter corresponds to the **bEndpointAddress** field in the endpoint descriptor.

`PolicyType`

A **ULONG** variable that specifies the policy parameter to change. The *Value* parameter contains the new value for the policy parameter, defined in *winusbio.h*. For information about how to use each of the pipe policies and the resulting behavior, see WinUSB Functions for Pipe Policy Modification.

`ValueLength`

The size, in bytes, of the buffer at *Value*.

`Value`

The new value for the policy parameter that *PolicyType* specifies. The size of this input parameter depends on the policy to change. For information about the size of this parameter, see the description of the *PolicyType* parameter.

Return value

WinUsb_SetPipePolicy returns **TRUE** if the operation succeeds. Otherwise, this function returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.
ERROR_INVALID_PARAMETER	The caller passed an invalid size for the policy parameter buffer in the *ValueLength* parameter.
ERROR_NOT_ENOUGH_MEMORY	Indicates that there is insufficient memory to perform the operation.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUSB Functions for Pipe Policy Modification

WinUsb_GetPipePolicy

WinUsb_Initialize

WinUsb_SetPowerPolicy function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_SetPowerPolicy** function sets the power policy for a device.

Syntax

```
BOOL WinUsb_SetPowerPolicy(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  ULONG                   PolicyType,
  ULONG                   ValueLength,
  PVOID                   Value
);
```

Parameters

`InterfaceHandle`

An opaque handle to the first (default) interface on the device, which is returned by WinUsb_Initialize.

`PolicyType`

A value that specifies the power policy to set. The following table describes symbolic constants that are defined in winusbio.h.

POLICY PARAMETER	DESCRIPTION
AUTO_SUSPEND (0x81)	Specifies the auto-suspend policy type; the power policy parameter must be specified by the caller in the *Value* parameter. For auto-suspend, the *Value* parameter must point to a UCHAR variable. If *Value* is **TRUE** (nonzero), the USB stack suspends the device if the device is idle. A device is idle if there are no transfers pending, or if the only pending transfers are IN transfers to interrupt or bulk endpoints. The default value is determined by the value set in the **DefaultIdleState** registry setting. By default, this value is **TRUE**.
SUSPEND_DELAY (0x83)	Specifies the suspend-delay policy type; the power policy parameter must be specified by the caller in the *Value* parameter. For suspend-delay, *Value* must point to a ULONG variable. *Value* specifies the minimum amount of time, in milliseconds, that the WinUSB driver must wait post transfer before it can suspend the device. The default value is determined by the value set in the **DefaultIdleTimeout** registry setting. By default, this value is five seconds.

`ValueLength`

The size, in bytes, of the buffer at *Value*.

`Value`

The new value for the power policy parameter. Datatype and value for *Value* depends on the type of power policy passed in *PolicyType*. For more information, see *PolicyType*.

Return value

WinUsb_SetPowerPolicy returns **TRUE** if the operation succeeds. Otherwise, this function returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.
ERROR_INVALID_PARAMETER	The caller passed an invalid size for the policy parameter buffer in the *ValueLength* parameter.
ERROR_NOT_ENOUGH_MEMORY	Indicates that there is insufficient memory to perform the operation.

Remarks

The following list summarizes the effects of changes to power management states:

- All pipe handles, interface handles, locks, and alternate settings are preserved across power management events.
- Any transfers that are in progress are suspended when a device transfers to a low power state, and they are resumed when the device is restored to a working state.
- The device and system must be in a working state before the client can restore a device-specific configuration. Clients can determine whether the device and system are in a working state from the WM_POWERBROADCAST message.
- The client can indicate that an interface is idle by calling **WinUsb_SetPowerPolicy**.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUSB Power Management

WinUsb_Initialize

WINUSB_SETUP_PACKET structure (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WINUSB_SETUP_PACKET** structure describes a USB setup packet.

Syntax

```
typedef struct _WINUSB_SETUP_PACKET {
  UCHAR  RequestType;
  UCHAR  Request;
  USHORT Value;
  USHORT Index;
  USHORT Length;
} WINUSB_SETUP_PACKET, *PWINUSB_SETUP_PACKET;
```

Members

`RequestType`

The request type. The values that are assigned to this member are defined in Table 9.2 of section 9.3 of the Universal Serial Bus (USB) specification (www.usb.org).

`Request`

The device request. The values that are assigned to this member are defined in Table 9.3 of section 9.4 of the Universal Serial Bus (USB) specification.

`Value`

The meaning of this member varies according to the request. For an explanation of this member, see the Universal Serial Bus (USB) specification.

`Index`

The meaning of this member varies according to the request. For an explanation of this member, see the Universal Serial Bus (USB) specification.

`Length`

The number of bytes to transfer.

Remarks

Callers of the WinUsb_ControlTransfer routine must pass in a **WINUSB_SETUP_PACKET** structure.

Requirements

Header	winusb.h (include Winusbio.h)

See also

USB Structures

WinUsb_ControlTransfer

WinUsb_StartTrackingForTimeSync function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_StartTrackingForTimeSync** function starts the time synchronization feature in the USB driver stack that gets the associated system QPC time for USB bus frames and microframes.

Syntax

```
BOOL WinUsb_StartTrackingForTimeSync(
  WINUSB_INTERFACE_HANDLE                      InterfaceHandle,
  PUSB_START_TRACKING_FOR_TIME_SYNC_INFORMATION StartTrackingInfo
);
```

Parameters

`InterfaceHandle`

An opaque handle retrieved in the previous call to WinUsb_Initialize.

`StartTrackingInfo`

A pointer to a USB_START_TRACKING_FOR_TIME_SYNC_INFORMATION structure. Set **TimeTrackingHandle** to INAVLID_HANDLE. Set **IsStartupDelayTolerable** to TRUE if the initial startup latency of up to 2.048 seconds is tolerable. FALSE, the registration is delayed until the USB driver stack is able to detect a valid frame or microframe boundary.

Return value

WinUsb_StartTrackingForTimeSync returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return one of the following error codes.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* or *StartTrackingInfo* parameter.

Requirements

Minimum supported client	Windows 10
Minimum supported server	Windows Server 2016
Target Platform	Universal

Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_StopTrackingForTimeSync function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_StopTrackingForTimeSync** function tops the time synchronization feature in the USB driver stack that gets the associated system QPC time for USB bus frames and microframes.

Syntax

```
BOOL WinUsb_StopTrackingForTimeSync(
  WINUSB_INTERFACE_HANDLE                      InterfaceHandle,
  PUSB_STOP_TRACKING_FOR_TIME_SYNC_INFORMATION StopTrackingInfo
);
```

Parameters

`InterfaceHandle`

An opaque handle retrieved in the previous call to WinUsb_Initialize.

`StopTrackingInfo`

A pointer to a USB_STOP_TRACKING_FOR_TIME_SYNC_INFORMATION structure. Set **TimeTrackingHandle** to the handle received in the previous call to WinUsb_StartTrackingForTimeSync.

Return value

WinUsb_StopTrackingForTimeSync returns **TRUE** if the operation succeeds. Otherwise, this routine returns **FALSE**, and the caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return one of the following error codes.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* or *StopTrackingInfo* parameter.

Requirements

Minimum supported client	Windows 10
Minimum supported server	Windows Server 2016
Target Platform	Universal
Header	winusb.h (include Winusb.h)

Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

WinUsb_StartTrackingForTimeSync

WinUsb_UnregisterIsochBuffer function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_UnregisterIsochBuffer** function releases all of the resources that WinUsb_RegisterIsochBuffer allocated for isochronous transfers. This is a synchronous operation.

Syntax

```
BOOL WinUsb_UnregisterIsochBuffer(
  WINUSB_ISOCH_BUFFER_HANDLE IsochBufferHandle
);
```

Parameters

`IsochBufferHandle`

An opaque handle to the transfer buffer that was registered by a previous call to WinUsb_RegisterIsochBuffer.

Return value

WinUsb_UnregisterIsochBuffer returns TRUE if the operation succeeds. Otherwise this function returns FALSE, and the caller can retrieve the logged error by calling **GetLastError**.

Remarks

The caller must ensure that there are no pending transfers that is currently using the buffer before calling **WinUsb_UnregisterIsochBuffer**. If there are pending transfers, **WinUsb_UnregisterIsochBuffer** blocks until those transfers are complete.

Requirements

Minimum supported client	Windows 8.1
Minimum supported server	Windows Server 2012 R2
Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

Send USB isochronous transfers from a WinUSB desktop app

WinUSB Functions

WinUsb_RegisterIsochBuffer

WinUsb_WriteIsochPipe function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_WriteIsochPipe** function writes the contents of a caller-supplied buffer to an isochronous OUT endpoint, starting on a specified frame number.

Syntax

```
BOOL WinUsb_WriteIsochPipe(
  WINUSB_ISOCH_BUFFER_HANDLE BufferHandle,
  ULONG                      Offset,
  ULONG                      Length,
  PULONG                     FrameNumber,
  LPOVERLAPPED               Overlapped
);
```

Parameters

`BufferHandle`

An opaque handle to the transfer buffer that was registered by a previous call to WinUsb_RegisterIsochBuffer.

`Offset`

Offset into the buffer relative to the start the transfer.

`Length`

Length in bytes of the transfer buffer.

`FrameNumber`

On input, indicates the starting frame number for the transfer. On output, contains the frame number of the frame that follows the last frame used in the transfer.

`Overlapped`

Pointer to an OVERLAPPED structure used for asynchronous operations.

Return value

WinUsb_WriteIsochPipe returns TRUE if the operation succeeds. Otherwise this function returns FALSE, and the caller can retrieve the logged error by calling **GetLastError**.

Remarks

WinUsb_WriteIsochPipe packetizes the transfer buffer so that in each 1ms interval, the host can send the maximum bytes allowed per interval. The maximum bytes is as specified by the endpoint descriptor for full and high-speed endpoints, and endpoint companion descriptor for SuperSpeed endpoints. If the caller submits multiple write requests to stream data to the device, the transfer size should be a multiple of the maximum bytes per interval (as returned by WinUsb_QueryPipeEx) * 8 / interval.

Because of the transfer packaging used in the underlying kernel-mode interface, the lowest latency notification

to an application or driver is 1ms intervals.

Requirements

Minimum supported client	Windows 8.1
Minimum supported server	Windows Server 2012 R2
Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

Send USB isochronous transfers from a WinUSB desktop app

WinUSB Functions

WinUsb_WriteIsochPipeAsap function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_WriteIsochPipeAsap** submits a request for writing the contents of a buffer to an isochronous OUT endpoint.

Syntax

```
BOOL WinUsb_WriteIsochPipeAsap(
  WINUSB_ISOCH_BUFFER_HANDLE BufferHandle,
  ULONG                      Offset,
  ULONG                      Length,
  BOOL                       ContinueStream,
  LPOVERLAPPED               Overlapped
);
```

Parameters

`BufferHandle`

An opaque handle to the transfer buffer that was registered by a previous call to WinUsb_RegisterIsochBuffer.

`Offset`

Offset into the buffer relative to the start the transfer.

`Length`

Length in bytes of the transfer buffer.

`ContinueStream`

Indicates that the transfer should only be submitted if it can be scheduled in the first frame after the last pending transfer.

`Overlapped`

Pointer to an OVERLAPPED structure used for asynchronous operations.

Return value

WinUsb_WriteIsochPipeAsap returns TRUE if the operation succeeds. Otherwise this function returns FALSE, and the caller can retrieve the logged error by calling **GetLastError**.

If the caller sets *ContinueStream* to TRUE, The transfer fails if Winusb.sys is unable to schedule the transfer to continue the stream without dropping one or more frames.

Remarks

WinUsb_WriteIsochPipeAsap allows the USB driver stack to choose the starting frame number for the transfer. If one or more transfers are already pending on the endpoint, the transfer will be scheduled for the frame number immediately following the last frame number of the last currently pending transfer.

WinUsb_WriteIsochPipeAsap packetizes the transfer buffer so that in each 1ms interval, the host can send

the maximum bytes allowed per interval. The maximum bytes is as specified by the endpoint descriptor for full and high-speed endpoints, and endpoint companion descriptor for SuperSpeed endpoints. If the caller submits multiple write requests to stream data to the device, the transfer size should be a multiple of the maximum bytes per interval (as returned by WinUsb_QueryPipeEx) * 8 / interval.

Because of the transfer packaging used in the underlying kernel-mode interface, the lowest latency notification to an application or driver is 1ms intervals.

Requirements

Minimum supported client	Windows 8.1
Minimum supported server	Windows Server 2012 R2
Target Platform	Universal
Header	winusb.h (include Winusb.h)
Library	Winusb.lib
DLL	Winusb.dll

See also

Send USB isochronous transfers from a WinUSB desktop app

WinUSB Functions

WinUsb_WritePipe function (winusb.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WinUsb_WritePipe** function writes data to a pipe.

Syntax

```
BOOL WinUsb_WritePipe(
  WINUSB_INTERFACE_HANDLE InterfaceHandle,
  UCHAR                   PipeID,
  PUCHAR                  Buffer,
  ULONG                   BufferLength,
  PULONG                  LengthTransferred,
  LPOVERLAPPED            Overlapped
);
```

Parameters

`InterfaceHandle`

An opaque handle to the interface that contains the endpoint with which the pipe is associated.

To write to a pipe that is associated with an endpoint in the first interface, use the handle returned by WinUsb_Initialize. For all other interfaces, use the handle to the target interface, retrieved by WinUsb_GetAssociatedInterface.

`PipeID`

PipeID corresponds to the **bEndpointAddress** field in the endpoint descriptor. For information about the layout of this field, see **Table 9-13** in "Universal Serial Bus Specification Revision 2.0" at USB Technology. In the **bEndpointAddress** field, Bit 7 indicates the direction of the endpoint: 0 for OUT; 1 for IN.

`Buffer`

A caller-allocated buffer that contains the data to write.

`BufferLength`

The number of bytes to write. This number must be less than or equal to the size, in bytes, of *Buffer*.

`LengthTransferred`

A pointer to a ULONG variable that receives the actual number of bytes that were written to the pipe. For more information, see Remarks.

`Overlapped`

An optional pointer to an OVERLAPPED structure, which is used for asynchronous operations. If this parameter is specified, **WinUsb_WritePipe** immediately returns, and the event is signaled when the operation is complete.

Return value

WinUsb_WritePipe returns **TRUE** if the operation succeeds. Otherwise, this function returns **FALSE**, and the

caller can retrieve the logged error by calling **GetLastError**.

GetLastError can return the following error code.

RETURN CODE	DESCRIPTION
ERROR_INVALID_HANDLE	The caller passed **NULL** in the *InterfaceHandle* parameter.
ERROR_IO_PENDING	Indicates that an overlapped I/O operation is in progress but has not completed. If the overlapped operation cannot be completed immediately, the function returns **FALSE** and the **GetLastError** function returns ERROR_IO_PENDING, indicating that the operation is executing in the background. Call WinUsb_GetOverlappedResult to check the success or failure of the operation.
ERROR_NOT_ENOUGH_MEMORY	Indicates that there is insufficient memory to perform the operation.
ERROR_SEM_TIMEOUT	The write operation initiated by WinUsb_WritePipe in the USB stack timed out before the operation could be completed.

Remarks

To create a write request, your the application must allocate a buffer, fill it with the data that you want to write to the device, and send the buffer to the host controller by calling **WinUsb_WritePipe**.

The following restrictions apply to the size of the buffer if RAW_IO is set:

- The buffer length must be a multiple of the maximum endpoint packet size.
- The length must be less than or equal to the value of MAXIMUM_TRANSFER_SIZE retrieved by WinUsb_GetPipePolicy.

There are no restrictions on the size of the buffer if RAW_IO is not set as the pipe's policy type. If the size of the buffer is greater than the maximum transfer length reported by MAXIMUM_TRANSFER_SIZE, WinUSB divides the request into smaller requests and submits them serially to the host controller.

A write request that contains zero-length data is forwarded down the USB stack.

If an application passes **NULL** in the *Overlapped* parameter (synchronous operation), it must ensure that *LengthTransferred* is not **NULL**, even when an operation produces no output data.

If *Overlapped* is not **NULL** (asynchronous operation), *LengthTransferred* can be set to **NULL**. For an overlapped operation (and if *LengthTransferred* is a non-**NULL** value), the value received in *LengthTransferred* after **WinUsb_WritePipe** returns is meaningless until the overlapped operation has completed. To retrieve the actual number of bytes returned, call WinUsb_GetOverlappedResult.

Requirements

Target Platform	Universal
Header	winusb.h (include Winusb.h)

Library	Winusb.lib
DLL	Winusb.dll

See also

WinUSB

WinUSB Functions

WinUsb_Initialize

winusbio.h header

2/1/2021 • 2 minutes to read • Edit Online

This header is used by USB driver reference. For more information, see:

- USB driver reference winusbio.h contains the following programming interfaces:

Structures

TITLE	DESCRIPTION
WINUSB_PIPE_INFORMATION	The WINUSB_PIPE_INFORMATION structure contains pipe information that the WinUsb_QueryPipe routine retrieves.
WINUSB_PIPE_INFORMATION_EX	The WINUSB_PIPE_INFORMATION_EX structure contains pipe information that the WinUsb_QueryPipeEx routine retrieves.

WINUSB_PIPE_INFORMATION structure (winusbio.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WINUSB_PIPE_INFORMATION** structure contains pipe information that the WinUsb_QueryPipe routine retrieves.

Syntax

```
typedef struct _WINUSB_PIPE_INFORMATION {
  USBD_PIPE_TYPE PipeType;
  UCHAR          PipeId;
  USHORT         MaximumPacketSize;
  UCHAR          Interval;
} WINUSB_PIPE_INFORMATION, *PWINUSB_PIPE_INFORMATION;
```

Members

`PipeType`

A USBD_PIPE_TYPE-type enumeration value that specifies the pipe type.

`PipeId`

The pipe identifier (ID).

`MaximumPacketSize`

The maximum size, in bytes, of the packets that are transmitted on the pipe.

`Interval`

The pipe interval.

Requirements

Header	winusbio.h (include Winusbio.h)

See also

USB Structures

USBD_PIPE_TYPE

WinUsb_QueryPipe

WINUSB_PIPE_INFORMATION_EX structure (winusbio.h)

2/1/2021 • 2 minutes to read • Edit Online

The **WINUSB_PIPE_INFORMATION_EX** structure contains pipe information that the WinUsb_QueryPipeEx routine retrieves.

Syntax

```
typedef struct _WINUSB_PIPE_INFORMATION_EX {
  USBD_PIPE_TYPE PipeType;
  UCHAR          PipeId;
  USHORT         MaximumPacketSize;
  UCHAR          Interval;
  ULONG          MaximumBytesPerInterval;
} WINUSB_PIPE_INFORMATION_EX, *PWINUSB_PIPE_INFORMATION_EX;
```

Members

`PipeType`

A USBD_PIPE_TYPE-type enumeration value that specifies the pipe type.

`PipeId`

The pipe identifier (ID).

`MaximumPacketSize`

The maximum size, in bytes, of the packets that are transmitted on the pipe.

`Interval`

The pipe interval.

`MaximumBytesPerInterval`

The maximum number of bytes that can be transmitted in single interval. This value may be a larger than the **MaximumPacketSize** value on high-bandwidth, high-speed periodic endpoints and SuperSpeed periodic endpoints, such as isochronous endpoints.

Requirements

Header	winusbio.h (include Winusbio.h)

See also

USB Structures

USBD_PIPE_TYPE

WinUsb_QueryPipe