

HID LIBRARY API SPECIFICATION

1. Introduction

The Silicon Labs HID library provides an API for communicating with a Human Interface Device (HID). This library provides methods for extracting device information and sending and receiving HID reports. C libraries are provided for Windows 2000 and later. Similarly, various include files are provided to import library functions into C# .NET and Visual Basic .NET. Refer to Table 1 for complete details.

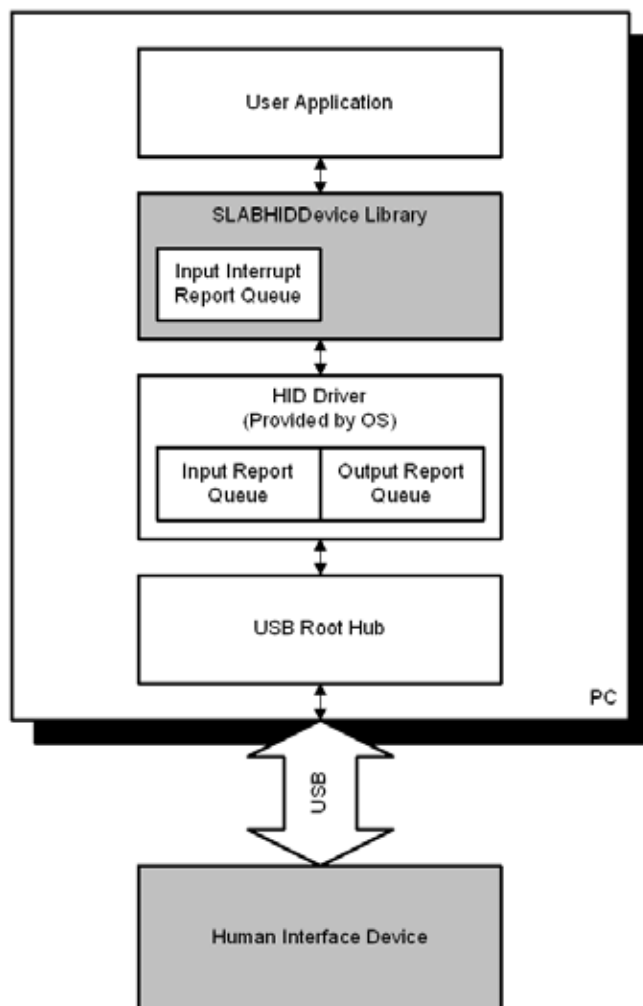


Figure 1. System Architecture Diagram

Table 1. HID Library Include Files

Operating System	Library	Include Files	Version
Windows 2000 and later	SLABHIDDevice.dll	SLABHIDDevice.h SLABHIDDevice.vb (VB .NET) SLABHIDDevice.bas (VB6) SLABHIDDevice.cs (C# .NET)	1.5

2. Library Usage

The HID Library contains the following files:

- SLABHIDDevice.dll—Dynamic link library that exports a C HID library
- SLABHIDDevice.lib—Provides build-time linking for C/C++ projects
- SLABHIDDevice.h—Defines HID library function prototypes and constants
- SLABHIDDevice.vb—Provides DLL imports and constants for Visual Basic .NET
- SLABHIDDevice.bas—Provides DLL imports and constants for Visual Basic 6
- SLABHIDDevice.cs—Provides DLL imports, constants, and an HID class for Visual C#

2.1. C/C++ Projects

The HID Library is designed to be universally compatible with most programming languages and is exported in C. Perform the following steps to use the library in a C/C++ project:

1. Add "SLABHIDDevice.lib" as a linker input dependency or add the following code:
#pragma comment (lib, "SLABHIDDevice.lib")
2. Include "SLABHIDDevice.h" as needed.
3. Provide "SLABHIDDevice.dll" with the executable (.exe file).

2.2. Visual Basic .NET Projects

Perform the following steps to use the library in a Visual Basic .NET project:

1. Include "SLABHIDDevice.vb" in the project to define library constants and library declarations.
2. Provide "SLABHIDDevice.dll" with the executable (.exe file).

2.3. Visual Basic 6 Projects

Perform the following steps to use the library in a Visual Basic 6 project:

1. Include "SLABHIDDevice.bas" in the project to define library constants and library declarations.
2. Provide "SLABHIDDevice.dll" with the executable (.exe file).

2.4. Visual C# Projects

Perform the following steps to use the library in a Visual C# project:

1. Include "SLABHIDDevice.cs" in the project to define library constants and library declarations.
This file also defines an HID class that uses the DLL functions and manages the "HID_DEVICE device" parameter automatically.
2. Provide "SLABHIDDevice.dll" with the executable (.exe file).

3. SLABHIDDevice Library Functions

Table 2. API Functions Table

Definition	Description	Page #
HidDevice_GetNumHidDevices()	Returns the number of devices connected	3
HidDevice_GetHidString()	Returns the specified USB string	4
HidDevice_GetHidIndexedString()	Returns the USB string descriptor by index	5
HidDevice_GetHidAttributes()	Returns the VID/PID/Release Number	6
HidDevice_GetHidGuid()	Returns the HID GUID	6
HidDevice_GetHidLibraryVersion()	Returns the library version and debug/release mode	6

3.1. HidDevice_GetNumHidDevices

Description: This function returns the number of devices connected to the host with matching vendor ID and product ID (VID, PID).

Prototype: `DWORD HidDevice_GetNumHidDevices (WORD vid, WORD pid)`

Parameters:

1. *vid*—Filter device results by vendor ID. If both *vid* and *pid* are set to 0x0000, devices will not be filtered by VID/PID.
2. *pid*—Filter device results by product ID. If both *vid* and *pid* are set to 0x0000, devices will not be filtered by VID/PID.

Return Value: A return value of 0 indicates that no devices are available. Otherwise returns the number of connected devices. When referring to a device by *deviceIndex*, the index may range from 0 to (*HidDevice_GetNumHidDevices()* – 1).

Remarks: This function returns the number of connected devices. This does not imply that a device is available for use. Users must call *HidDevice_GetNumHidDevices()* before calling any function that takes a device index as a parameter in order to build an up-to-date device list. If a device is installed or removed after calling *HidDevice_GetNumHidDevices()*, then the device list will be out of date, and the results may be unpredictable. Currently, *HidDevice_GetHidString()*, *HidDevice_GetHidIndexedString()*, *HidDevice_GetHidAttributes()*, and *HidDevice_Open()* are the only functions that take a device index parameter.

3.2. HidDevice_GetHidString

Description: This function returns a null-terminated vendor ID, product ID, device path, serial, manufacturer, or product string for the device specified by an index passed in *deviceIndex*.

Prototype: `BYTE HidDevice_GetHidString (DWORD deviceIndex, WORD vid, WORD pid, BYTE hidStringType, char* deviceString, DWORD deviceStringLength)`

Parameters:

1. *deviceIndex*—Index specifying which device to retrieve the string from. This value ranges from 0 to (*HidDevice_GetNumHidDevices()* – 1).
2. *vid*—Filter device results by vendor ID. If both *vid* and *pid* are set to 0x0000, devices will not be filtered by VID/PID.
3. *pid*—Filter device results by product ID. If both *vid* and *pid* are set to 0x0000, devices will not be filtered by VID/PID.
4. *hidStringType*—Determines if deviceString contains a vendor ID, product ID, device path, serial, manufacturer, or product string.

Definition	Value	Length	Description
HID_VID_STRING	0x01	5	Vendor ID
HID_PID_STRING	0x02	5	Product ID
HID_PATH_STRING	0x03	260	Device path
HID_SERIAL_STRING	0x04	256	Serial string
HID_MANUFACTURER_STRING	0x05	256	Manufacturer String
HID_PRODUCT_STRING	0x06	256	Product String

5. *deviceString*—Pointer to a user-defined, ASCII string that will contain a NULL-terminated device string on return. The string must be allocated with enough space to return the desired string.
6. *deviceStringLength*—The length of *deviceString* in bytes.

Return Value: HID_DEVICE_SUCCESS
HID_DEVICE_NOT_FOUND
HID_DEVICE_INVALID_BUFFER_SIZE
HID_DEVICE_SYSTEM_CODE
HID_DEVICE_ALREADY_OPENED
HID_DEVICE_CANNOT_GET_HID_INFO

3.3. HidDevice_GetHidIndexedString

Description: This function returns an ASCII, null-terminated USB string descriptor using the specified string descriptor index for the device specified by *deviceIndex*.

Prototype: BYTE HidDevice_GetHidIndexedString (DWORD deviceIndex, WORD vid, WORD, pin, DWORD stringIndex, char* deviceString, DWORD deviceStringLength)

Parameters:

1. *deviceIndex*—Index specifying from which device to retrieve the string. This value ranges from 0 to (*HidDevice_GetHidNumDevices()* – 1).
2. *vid*—Filter device results by vendor ID. If both *vid* and *pid* are set to 0x0000, devices will not be filtered by VID/PID.
3. *pid*—Filter device results by product ID. If both *vid* and *pid* are set to 0x0000, devices will not be filtered by VID/PID.
4. *stringIndex*—Index specifying which USB string descriptor to retrieve.
5. *deviceString*—Pointer to a user-defined, ASCII string that will contain a NULL-terminated device string on return. The string must be allocated with enough space to return the desired string and must be at least 256 bytes.
6. *deviceStringLength*—The length of *deviceString* in bytes.

Return Value: HID_DEVICE_SUCCESS
HID_DEVICE_NOT_FOUND
HID_DEVICE_INVALID_BUFFER_SIZE
HID_DEVICE_SYSTEM_CODE
HID_DEVICE_ALREADY_OPENED

3.4. HidDevice_GetHidAttributes

Description: This function returns the USB vendor ID, product ID, and device release number for the device specified by *deviceIndex*.

Prototype: `BYTE HidDevice_GetHidAttributes (DWORD deviceIndex, WORD vid, WORD pid, WORD* deviceVid, WORD* devicePid, WORD* deviceReleaseNumber)`

Parameters:

1. *deviceIndex*—Index specifying from which device to retrieve attributes. This value ranges from 0 to (*HidDevice_GetNumHidDevices*() – 1).
2. *vid*—Filter device results by vendor ID. If both *vid* and *pid* are set to 0x0000, devices will not be filtered by VID/PID.
3. *pid*—Filter device results by product ID. If both *vid* and *pid* are set to 0x0000, devices will not be filtered by VID/PID.
4. *deviceVid*—Returns the USB device vendor ID.
5. *devicePid*—Returns the USB device product ID.
6. *deviceReleaseNumber*—Returns the USB device bcdVersion, or device release number, in binary-coded decimal.

Return Value: HID_DEVICE_SUCCESS
HID_DEVICE_NOT_FOUND
HID_DEVICE_SYSTEM_CODE
HID_DEVICE_ALREADY_OPENED

3.5. HidDevice_GetHidGuid

Description: This function returns the HID GUID.

Prototype: `void HidDevice_GetHidGuid (void* hidGuid)`

Parameters: 1. *hidGuid*—Returns the HID GUID.

Remarks: See "8. Surprise Removal" on page 21 for more information on surprise removal.

3.6. HidDevice_GetLibraryVersion

Description: This function returns the library version and whether the build is a debug or release version.

Prototype: `BYTE HidDevice_GetLibraryVersion (BYTE* major, BYTE* minor, BOOL* release)`

Parameters:

1. *major*—Returns the library major version number. This value ranges from 0 to 255.
2. *minor*—Returns the library minor version number. This value ranges from 0 to 255.
3. *release*—Returns *TRUE* if the library was built in release mode. Returns *FALSE* if the library was built in debug mode.

Return Value: HID_DEVICE_SUCCESS
HID_DEVICE_INVALID_BUFFER_SIZE

4. SLABHIDDevice Library Functions for Opened Devices

Table 3. API Functions for Opened Devices

Definition	Description	Page #
HidDevice_Open()	Opens an HID using a device index	8
HidDevice_IsOpened()	Returns the opened state	9
HidDevice_GetHandle()	Returns the HID handle for the currently opened device	9
HidDevice_GetString()	Returns the specified USB string for the currently opened device	9
HidDevice_GetIndexedString()	Returns the USB string descriptor by index for the currently opened device	10
HidDevice_GetAttributes()	Returns the VID/PID/Release Number for the currently opened device	10
HidDevice_SetFeatureReport_Control()	Sends an HID feature report from the host to the device over the control endpoint	11
HidDevice_GetFeatureReport_Control()	Receives an HID feature report from the device to the host over the control endpoint	11
HidDevice_SetOutputReport_Interrupt()	Sends an HID output report from the host to the device over the interrupt endpoint	11
HidDevice_GetInputReport_Interrupt()	Receives HID input reports from the device to the host over the interrupt endpoint	12
HidDevice_SetOutputReport_Control()	Sends an HID output report from the host to the device over the control endpoint	13
HidDevice_GetInputReport_Control()	Receives an HID input report from the device to the host over the control endpoint	13
HidDevice_GetInputReportBufferLength()	Returns the maximum input report size including the report ID	14
HidDevice_GetOutputReportBufferLength()	Returns the maximum output report size including the report ID	14
HidDevice_GetFeatureReportBufferLength()	Returns the maximum feature report size including the report ID	14
HidDevice_GetMaxReportRequest()	Returns the maximum number of input reports that can be queued in the HID driver	14
HidDevice_FlushBuffers()	Deletes all pending input reports in the HID driver queue and the HID library	15
HidDevice_Cancello()	Cancels all pending input and output operations issued by the calling thread	15
HidDevice_GetTimeouts()	Returns the input and output report timeouts over the interrupt endpoint	15
HidDevice_SetTimeouts()	Sets the input and output report timeouts over the interrupt endpoint	16
HidDevice_Close()	Closes the currently opened device	16
Note: These functions require an additional " <i>HID_DEVICE device</i> " parameter at the beginning of the argument list. This parameter is an HID class object pointer as returned by <i>HidDevice_Open()</i> .		

4.1. HidDevice_Open

Description: Opens a device using a device index and returns a device object pointer which will be used for subsequent access.

Prototype: `BYTE HidDevice_Open (HID_DEVICE* device, DWORD deviceIndex, WORD vid, WORD pid, DWORD numInputBuffers)`

Parameters:

1. *device*—Returns a pointer to an HID class object used for subsequent device access.
2. *deviceIndex*—Zero-based device index ranging from 0 to (*HidDevice_GetNumDevices()* – 1).
3. *vid*—Filter device results by vendor ID. If both *vid* and *pid* are set to 0x0000, then devices will not be filtered by VID/PID.
4. *pid*—Filter device results by product ID. If both *vid* and *pid* are set to 0x0000, then devices will not be filtered by VID/PID.
5. *numInputBuffers*—Specifies the number of input report buffers to queue in the HID driver.

Definition	Value	Description
MAX_REPORT_REQUEST_XP	512	Maximum number of input report buffers for Windows XP and later.
MAX_REPORT_REQUEST_2K	200	Maximum number of input report buffers for Windows 2000.
DEFAULT_REPORT_INPUT_BUFFER	0	Use OS-dependent HID driver default.

Return Value: `HID_DEVICE_SUCCESS`
`HID_DEVICE_ALREADY_OPENED`
`HID_DEVICE_CANNOT_GET_HID_INFO`
`HID_DEVICE_NOT_FOUND`

Remarks: For Windows 2000, the maximum number of queued input reports is 200.
For Windows XP and later, the maximum number of queued input reports is 512.
For Windows XP and later, the OS default number of queued input reports is 32.
In most cases, call this function with *numInputBuffers* set to `MAX_REPORT_REQUEST_XP` to achieve the best input report throughput.
If *numInputBuffers* is set to a value higher than the OS maximum, then the OS maximum value will be used.
It is very common to list all connected devices by serial string. After calling *HidDevice_GetNumHidDevices()*, call *HidDevice_GetHidString()* to get each device's serial string, making sure to check the return code. Since *HidDevice_GetHidString()* may fail (indicating that the device is unavailable) it is not appropriate to simply add each device's serial string to a list and use the list index to open. Similarly, a device index may have changed since the last time the device list was updated. In this case, the user should select a device solely by serial string and query all devices for a matching serial string.
This function allocates the dynamic report buffers, sets timeouts to their default values, and clears the report queue.

4.2. HidDevice_IsOpened

Description: This function returns the device opened state.

Prototype: `BOOL HidDevice_IsOpened (HID_DEVICE device)`

Parameters: 1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.

Return Value: Returns *TRUE* if a device is opened. Returns *FALSE* if the device is invalid or if a device is not opened.

4.3. HidDevice_GetHandle

Description: This function returns the HID handle for the currently-opened device

Prototype: `HANDLE HidDevice_GetHandle (HID_DEVICE device)`

Parameters: 1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.

Return Value: Returns a Windows HANDLE created by `::CreateFile()` in *HidDevice_Open()*. Returns *INVALID_HANDLE_VALUE* if a device is not open or if there was an error opening a device.

4.4. HidDevice_GetString

Description: This function returns an ASCII, null-terminated vendor ID, product ID, device path, serial, manufacturer, or product string for the device specified by *device*.

Prototype: `BYTE HidDevice_GetString (HID_DEVICE device, BYTE hidStringType, char* deviceString, DWORD deviceStringLength)`

Parameters: 1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.
2. *hidStringType*—Determines if *deviceString* contains a vendor ID, product ID, device path, serial, manufacturer, or product string.

Definition	Value	Length	Description
HID_VID_STRING	0x01	5	Vendor ID
HID_PID_STRING	0x02	5	Product ID
HID_PATH_STRING	0x03	260	Device path
HID_SERIAL_STRING	0x04	256	Serial string
HID_MANUFACTURER_STRING	0x05	256	Manufacturer String
HID_PRODUCT_STRING	0x06	256	Product String

5. *deviceString*—Pointer to a user-defined, ASCII string which will contain a NULL-terminated device string on return. The string must be allocated with enough space to return the desired string.

6. *deviceStringLength*—The length of *deviceString* in bytes.

Return Value: HID_DEVICE_SUCCESS
HID_DEVICE_NOT_FOUND
HID_DEVICE_NOT_OPENED
HID_DEVICE_INVALID_BUFFER_SIZE
HID_DEVICE_SYSTEM_CODE
HID_DEVICE_ALREADY_OPENED
HID_DEVICE_CANNOT_GET_HID_INFO

Remarks: Once a device is opened, *deviceIndex* is no longer valid. Similarly, to retrieve a USB string, *HidDevice_GetString()* must be used rather than *HidDevice_GetHidString()*.

4.5. HidDevice_GetIndexedString

Description: This function returns an ASCII, null-terminated USB string descriptor using the specified string descriptor index for the currently-open device.

Prototype: `BYTE HidDevice_GetIndexedString (HID_DEVICE_ device, DWORD stringIndex, char* deviceString, DWORD deviceStringLength)`

Parameters:

1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.
2. *stringIndex*—Index specifying which USB string descriptor to retrieve.
3. *deviceString*—Pointer to a user-defined, ASCII string which will contain a NULL-terminated device string on return. The string must be allocated with enough space to return the desired string and must be at least 256 bytes.
4. *deviceStringLength*—The length of *deviceString* in bytes.

Return Value: HID_DEVICE_SUCCESS
HID_DEVICE_NOT_FOUND
HID_DEVICE_NOT_OPENED
HID_DEVICE_INVALID_BUFFER_SIZE
HID_DEVICE_SYSTEM_CODE

4.6. HidDevice_GetAttributes

Description: This function returns the USB vendor ID, product ID, and device release number for the currently-open device.

Prototype: `BYTE HidDevice_GetAttributes (HID_DEVICE device, WORD* deviceVid, WORD* devicePid, WORD* deviceReleaseNumber)`

Parameters:

1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.
2. *deviceVid*—Returns the USB device vendor ID.
3. *devicePid*—Returns the USB device product ID.
4. *deviceReleaseNumber*—Returns the USB device bcdVersion, or device release number, in binary-coded decimal.

Return Value: HID_DEVICE_SUCCESS
HID_DEVICE_NOT_FOUND
HID_DEVICE_NOT_OPENED
HID_DEVICE_SYSTEM_CODE

4.7. HidDevice_SetFeatureReport_Control

Description: This function sends an HID feature report from the host to the device over the control endpoint.

Prototype: `BYTE HidDevice_SetFeatureReport_Control (HID_DEVICE device, BYTE* buffer, DWORD bufferSize)`

Parameters:

1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.
2. *buffer*—A byte array containing a feature report. The first byte specifies the report ID.
3. *bufferSize*—The size of the feature report including the report ID.

Return Value: `HID_DEVICE_SUCCESS`
`HID_DEVICE_NOT_FOUND`
`HID_DEVICE_INVALID_BUFFER_SIZE`
`HID_DEVICE_NOT_OPENED`
`HID_DEVICE_TRANSFER_FAILED`

4.8. HidDevice_GetFeatureReport_Control

Description: This function receives an HID feature report from the host to the device over the control endpoint.

Prototype: `BYTE HidDevice_GetFeatureReport_Control (HID_DEVICE device, BYTE* buffer, DWORD bufferSize)`

Parameters:

1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.
2. *buffer*—Returns a byte array containing a feature report with report ID. The user must call this function with the first byte in *buffer* set to the report ID of the report to receive. *Buffer* must be large enough to hold the feature report including report ID and, in most cases, should be allocated with a size equal to *HidDevice_GetFeatureReportBufferLength()*.
3. *bufferSize*—The size of *buffer* in bytes.

Return Value: `HID_DEVICE_SUCCESS`
`HID_DEVICE_NOT_FOUND`
`HID_DEVICE_INVALID_BUFFER_SIZE`
`HID_DEVICE_NOT_OPENED`
`HID_DEVICE_TRANSFER_FAILED`

4.9. HidDevice_SetOutputReport_Interrupt

Description: This function sends an HID output report from the host to the device over the interrupt endpoint.

Prototype: `BYTE HidDevice_SetOutputReport_Interrupt (HID_DEVICE device, BYTE* buffer, DWORD bufferSize)`

Parameters:

1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.
2. *buffer*—A byte array containing an output report. The first byte specifies the report ID.
3. *bufferSize*—The size of the output report including the report ID.

Return Value: `HID_DEVICE_SUCCESS`
`HID_DEVICE_NOT_FOUND`
`HID_DEVICE_INVALID_BUFFER_SIZE`
`HID_DEVICE_NOT_OPENED`
`HID_DEVICE_TRANSFER_FAILED`
`HID_DEVICE_TRANSFER_TIMEOUT`

4.10. HidDevice_GetInputReport_Interrupt

Description: This function receives an HID input report from the host to the device over the interrupt endpoint.
Prototype: `BYTE HidDevice_GetInputReport_Interrupt (HID_DEVICE device, BYTE* buffer, DWORD bufferSize, DWORD numReports, DWORD* bytesReturned)`

Parameters:

1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.
2. *buffer*—Returns a byte array of data containing up to *numReports* number of input reports. Each report occupies *HidDevice_GetInputReportBufferLength()* number of bytes. The first byte of each report buffer contains the report ID.
3. *bufferSize*—The size of *buffer* in bytes.
4. *numReports*—The maximum number of input reports to return.
5. *bytesReturned*—The number of bytes returned in *buffer*.

Return Value: `HID_DEVICE_SUCCESS`
`HID_DEVICE_NOT_FOUND`
`HID_DEVICE_INVALID_BUFFER_SIZE`
`HID_DEVICE_NOT_OPENED`
`HID_DEVICE_TRANSFER_FAILED`
`HID_DEVICE_TRANSFER_TIMEOUT`

Remarks: Set input report interrupt timeouts by settings *getReportTimeout* in *HidDevice_SetTimeouts()*. Each input report returned in *buffer* is stored on boundaries set by the maximum input report buffer size. For example, if the maximum input report buffer size as returned by *HidDevice_GetInputReportBufferLength()* is 64 and *HidDevice_GetInputReport_Interrupt()* returns 2 input reports, then *bytesReturned* will return with 128 bytes. The first input report starts at *buffer[0]*, and the second input report starts at *buffer[64]* regardless of the actual size of the report. To retrieve the maximum number of input reports possible, call *HidDevice_GetInputReport_Interrupt()* with *bufferSize* set to $(HidDevice_GetInputReportBufferLength() * HidDevice_GetMaxReportRequest())$ and *numReports* set to *HidDevice_GetMaxReportRequest()*.

4.11. HidDevice_SetInputReport_Control

Description: This function sends an HID input report from the host to the device over the control endpoint.

Prototype: `BYTE HidDevice_SetInputReport_Control (HID_DEVICE device, BYTE* buffer, DWORD bufferSize)`

Parameters:

1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.
2. *buffer*—A byte array containing an output report. The first byte specifies the report ID.
3. *bufferSize*—The size of the output report including the report ID.

Return Value: `HID_DEVICE_SUCCESS`
`HID_DEVICE_NOT_FOUND`
`HID_DEVICE_UNSPUPPORTED_FUNCTION`
`HID_DEVICE_INVALID_BUFFER_SIZE`
`HID_DEVICE_NOT_OPENED`
`HID_DEVICE_TRANSFER_FAILED`

Remarks: Attempting to call this function on Windows 2000 or earlier will fail and return *HID_DEVICE_UNSUPPORTED_FUNCTION*.

4.12. HidDevice_GetInputReport_Control

Description: This function receives an HID input report from the host to the device over the control endpoint.

Prototype: `BYTE HidDevice_GetInputReport_Control (HID_DEVICE device, BYTE* buffer, DWORD bufferSize)`

Parameters:

1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.
2. *buffer*—Returns a byte array containing an input report with report ID. The user must call this function with the first byte in *buffer* set to the report ID of the report to receive. *Buffer* must be large enough to hold the input report including report ID and, in most cases, should be allocated with a size equal to *HidDevice_GetInputReportBufferLength()*.
3. *bufferSize*—The size of *buffer* in bytes.

Return Value: `HID_DEVICE_SUCCESS`
`HID_DEVICE_NOT_FOUND`
`HID_DEVICE_UNSPUPPORTED_FUNCTION`
`HID_DEVICE_INVALID_BUFFER_SIZE`
`HID_DEVICE_NOT_OPENED`
`HID_DEVICE_TRANSFER_FAILED`

Remarks: Attempting to call this function on Windows 2000 or earlier will fail and return *HID_DEVICE_UNSUPPORTED_FUNCTION*.

4.13. HidDevice_GetInputReportBufferLength

Description: This function returns the maximum input report size, including the report ID.

Prototype: WORD HidDevice_GetInputReportBuffferLength (HID_DEVICE device)

Parameters: 1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.

Return Value: A return value of zero indicates that the specified device cannot be found. A non-zero value specifies the maximum input report size including the report ID. This function should be called to determine the appropriate buffer size for *HidDevice_GetInputReport_Interrupt()* and *HidDevice_GetInputReport_Control()*.

4.14. HidDevice_GetOutputReportBufferLength

Description: This function returns the maximum output report size including the report ID.

Prototype: WORD HidDevice_GetOutputReportBuffferLength (HID_DEVICE device)

Parameters: 1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.

Return Value: A return value of zero indicates that the specified device cannot be found. A non-zero value specifies the maximum output report size, including the report ID.

4.15. HidDevice_GetFeatureReportBufferLength

Description: This function returns the maximum feature report size, including the report ID.

Prototype: WORD HidDevice_GetFeatureReportBuffferLength (HID_DEVICE device)

Parameters: 1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.

Return Value: A return value of zero indicates that the specified device cannot be found. A non-zero value specifies the maximum feature report size, including the report ID. This function should be called to determine the appropriate buffer size for *HidDevice_GetFeatureReport_Control()*.

4.16. HidDevice_GetMaxReportRequest

Description: This function returns the maximum number of input reports that can be queued in the HID driver.

Prototype: WORD HidDevice_GetMaxReportRequest (HID_DEVICE device)

Parameters: 1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.

Return Value: A return value of zero indicates that the specified device cannot be found. A non-zero value specifies the maximum size of the HID driver input report queue in number of reports. This function should be called to determine the appropriate buffer size for *HidDevice_GetInputReport_Interrupt()*.

For Windows XP and later, this function returns up to 512.

For Windows 2000, this function returns up to 200.

This function cannot be called before a device is opened. See "4.1. HidDevice_Open" on page 8 for more information

4.17. HidDevice_FlushBuffers

Description: This function deletes all pending input reports in the HID driver queue and the HID library.

Prototype: BYTE HidDevice_FlushBuffers (HID_DEVICE device)

Parameters: 1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.

Return Value: Returns *TRUE* if the function completed successfully.

Returns *FALSE* if the function failed or if the device does not exist.

Remarks: This function first cancels pending overlapped input report reads (*::Cancello()*). Next, it flushes the HID driver input report queue. Finally, it flushes the library's internal input report queue.

4.18. HidDevice_CancelIo

Description: This function cancels all pending input and output operations issued by the calling thread.

Prototype: BYTE HidDevice_CancelIo (HID_DEVICE device)

Parameters: 1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.

Return Value: Returns *TRUE* if the function completed successfully.

Returns *FALSE* if the function failed or if the device does not exist.

Remarks: *::Cancello()* called in one thread cannot cancel reads and writes issued in another thread. Therefore, it is necessary to call this function in all threads that call *HidDevice_GetInputReport_Interrupt()* prior to call *HidDevice_Close()*. See "7. Thread Read Access Models" on page 19 for more information.

4.19. HidDevice_GetTimeouts

Description: This function returns the input and output report timeouts over the interrupt endpoint.

Prototype: BYTE HidDevice_GetTimeouts (HID_DEVICE device, DWORD* getReportTimeout, DWORD* setReportTimeout)

Parameters: 1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.

2. *getReportTimeout*—Returns the read timeout for *HidDevice_GetInputReport_Interrupt()* in milliseconds. This timeout specifies the number of milliseconds that must elapse before *HidDevice_GetInputReport_Interrupt()* will return with fewer bytes than requested with an *HID_DEVICE_TRANSFER_TIMEOUT* error. A timeout of 0 ms will return immediately with any available data.

3. *setReportTimeout*—Returns the write timeout for *HidDevice_SetOutputReport_Interrupt()* in milliseconds. This timeout specifies the number of milliseconds that must elapse for an output report before *HidDevice_SetOutputReport_Interrupt()* returns with an *HID_DEVICE_TRANSFER_TIMEOUT* error.

Remarks: Read and write timeouts are maintained for each device but are not persistent across *HidDevice_Open()/HidDevice_Close()*. Allow for sufficient write timeouts to make sure that an output report can be transmitted successfully.

The default get report and set report timeouts are both 1000 ms.

4.20. HidDevice_SetTimeouts

Description: This function sets the input and output report timeouts over the interrupt endpoint. Timeouts are used for *HidDevice_GetInputReport_Interrupt()* and *HidDevice_SetOutputReport_Interrupt()*.

Prototype: `BYTE HidDevice_SetTimeouts(HID_DEVICE device, DWORD getReportTimeout, DWORD setReportTimeout)`

Parameters:

1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.
2. *getReportTimeout*—Sets the timeout used in *HidDevice_GetInputReport_Interrupt()* in milliseconds. This timeout specifies the number of milliseconds that must elapse before *HidDevice_GetInputReport_Interrupt()* will return fewer bytes than requested with an *HID_DEVICE_TRANSFER_TIMEOUT* error. A timeout of 0 ms will return immediately with any available data.
3. *setReportTimeout*—Sets the timeout used in *HidDevice_SetOutputReport_Interrupt()* in milliseconds. This timeout specifies the number of milliseconds that must elapse during an output report before *HidDevice_SetOutputReport_Interrupt()* will return with an *HID_DEVICE_TRANSFER_TIMEOUT* error.

Remarks: If get report timeouts are set to a large value and no data is received, the application may appear unresponsive. Therefore, it is recommended to set timeouts to an appropriate value before reading from the device.

The default value for get report and set report timeouts is 1000 ms and can be set to any number of milliseconds from 0 to 0xFFFFFFFF.

4.21. HidDevice_Close

Description: This function closes the currently-open device.

Prototype: `BYTE HidDevice_Close (HID_DEVICE device)`

Parameters: 1. *device*—A pointer to an HID class object returned by *HidDevice_Open()*.

Return Value: `HID_DEVICE_SUCCESS`
`HID_DEVICE_HANDLE_ERROR`
`HID_DEVICE_NOT_FOUND`
`HID_DEVICE_NOT_OPENED`

Remarks: This function deletes the internal library input report buffer and cancels any pending input report reads via *::Cancello()*. This function is responsible for deallocating the HID class object allocated in *HidDevice_Open()*.

5. Library Returns Codes

Each library function returns a *BYTE* return code to indicate that the function returned successfully or to describe an error. Table 4 describes each return code.

Table 4. Return Code Descriptions

Definition	Value	Description
HID_DEVICE_SUCCESS	0x00	Function returned successfully.
HID_DEVICE_NOT_FOUND	0x01	Indicates that the specified device index was invalid or the device does not exist or is inaccessible.
HID_DEVICE_NOT_OPENED	0x02	Indicates that the device must be opened prior to calling the function.
HID_DEVICE_ALREADY_OPENED	0x03	Indicates that the device is already opened and cannot be re-opened.
HID_DEVICE_TRANSFER_TIMEOUT	0x04	Indicates that a get or set report function returned due to a timeout. <i>HidDevice_GetInputReport_Interrupt()</i> returned with less bytes than requested.
HID_DEVICE_TRANSFER_FAILED	0x05	The host failed to communicate with the device or function parameters are incorrect.
HID_DEVICE_CANNOT_GET_HID_INFO	0x06	Cannot retrieve device path or <i>hidStringType</i> is invalid.
HID_DEVICE_HANDLE_ERROR	0x07	HID :: <i>CreateFile()</i> handle is invalid.
HID_DEVICE_INVALID_BUFFER_SIZE	0x08	Specified buffer is not large enough to return requested data.
HID_DEVICE_SYSTEM_CODE	0x09	Indicates that a system error occurred. Call <i>GetLastError()</i> to retrieve the system error code.
HID_DEVICE_UNSUPPORTED_FUNCTION	0x0A	The function is only supported on certain Windows versions.
HID_DEVICE_UNKNOWN_ERROR	0xFF	This is the default return code value. This value should never be returned.

6. Thread Safety

The HID library and associated functions are not thread safe. This means that calling library functions simultaneously from multiple threads may have undesirable effects.

To use the library functions in more than one thread, the user should do the following:

1. Call HID library functions from within a critical section such that only a single function is being called at any given time. If a function is being called in one thread, the user must prevent another thread from calling any function until the first function returns.
2. *HidDevice_GetInputReport_Interrupt()* issues a pending read request that cannot be canceled from another thread. If the user calls *HidDevice_Close()* in a thread other than the one in which the read request was created, the device will not be accessible after calling *HidDevice_Close()*. The thread that issued the pending read request must return/terminate successfully before the device can be accessed again. See "7. Thread Read Access Models" on page 19 for more information.

7. Thread Read Access Models

There are several common read access models when using the HID library functions, and, due to certain limitations with overlapped I/O, certain precautions must be taken. `::Cancello()` can only cancel pending I/O (reads/writes) issued in the same thread in which `::Cancello()` is called. Due to this limitation, the user is responsible for cancelling pending I/O in each thread before closing the device. Failure to do so will result in an inaccessible device until the thread releases access to the device handle.

`HidDevice_Close()` calls `::Cancello()` prior to calling `::CloseHandle()`. `HidDevice_GetInputReport_Interrupt()` issues a pending read request. The request completes if at least one input report is read. The request is still pending if the operation times out. `::Cancello()` cancels any pending I/O requests issued by the calling thread.

Tables 5 through 9 describe five common access models and the expected behavior:

Notes:

1. Read*—Read is still pending and was issued in the specified thread
2. Read? —Read is still pending and was issued in one of the threads (indeterminate)
3. Open—Call `HidDevice_Open()`
4. Read—Call `HidDevice_GetInputReport_Interrupt()`
5. Close—Call `HidDevice_Close()`
6. Cancel—Call `HidDevice_Cancello()`

Table 5. Single Thread Access Model (Safe)

Thread A	Thread B	Result
<i>Open</i>	—	—
<i>Read*</i>	—	—
<i>Close</i>	—	OK

Table 6. Split Thread Access Model (Unsafe)

Thread A	Thread B	Result
<i>Open</i>	—	—
—	<i>Read*</i>	—
<i>Close</i>	—	Error: Device inaccessible
—	Terminate Thread	OK: Thread relinquishes device access

Table 7. Split Thread Access Mode (Safe)

Thread A	Thread B	Result
<i>Open</i>	—	—
—	<i>Read*</i>	—
—	<i>Cancel</i>	—
<i>Close</i>	—	OK

Table 8. Multi-Thread Access Model (Unsafe)

Thread A	Thread B	Result
<i>Open</i>	—	—
<i>Read?</i>	<i>Read?</i>	—
<i>Close</i>	—	<i>If read is pending in Thread A:</i> OK <i>If read is pending in Thread B:</i> Error: Device inaccessible
—	Terminate Thread	OK: Thread relinquishes device access

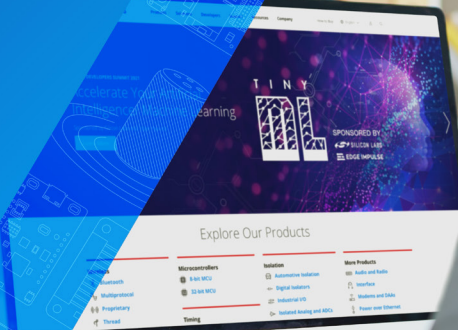
Table 9. Multi-Thread Access Model (Safe)

Thread A	Thread B	Result
<i>Open</i>	—	—
<i>Read?</i>	<i>Read?</i>	—
—	<i>Cancel</i>	—
<i>Close</i>	—	OK

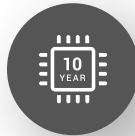
8. Surprise Removal

HidDevice_GetHidGuid() returns the HID GUID so that Windows applications or services can register for the *WM_DEVICECHANGE* Windows message. Once registered, the application will receive device arrival and removal notifications for HID devices. The application should retrieve the device path to filter devices. Similarly, if a *DBT_DEVICEREMOVECOMPLETE* message is received, the application should check to see if the device path matches the device path of any connected devices. If this is the case, the device was removed, and the application should close the device. In addition, if a *DBT_DEVICEARRIVAL* message is received, then the application might add the new device to a device list so that users can select any HID matching the required VID/PID. See accompanying example code for information on how to implement surprise removal and device arrival. Search for Knowledge Base Article # [000005173](#), [137957](#), [137956](#), [114055](#), and [311153](#) for programming examples for C++(MFC), Visual Basic .NET, Visual C# WinForms, Visual C# WPF, Visual C# .NET.

Smart. Connected. Energy-Friendly.



IoT Portfolio
www.silabs.com/products



Quality
www.silabs.com/quality



Support & Community
www.silabs.com/community

Disclaimer

Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.

Note: This content may contain offensive terminology that is now obsolete. Silicon Labs is replacing these terms with inclusive language wherever possible. For more information, visit www.silabs.com/about-us/inclusive-lexicon-project

Trademark Information

Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Redpine Signals®, WiSeConnect®, n-Link®, ThreadArch®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, Gecko OS, Gecko OS Studio, Precision32®, Simplicity Studio®, Telegesis, the Telegesis Logo®, USBXpress®, Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.



Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com