CP2112 HID USB-to-SMBus API Specification

1. Introduction

The Silicon Labs HID USB-to-SMBus interface library provides a simple API to configure and operate CP2112 devices. The library provides interface abstraction so that users can develop their application without writing any USB HID Code. C libraries implementing the CP2112 Interface Specification are provided for Windows 2000 and later and Mac OS X 10.5 and later. Similarly, various include files are provided to import library functions into C# .NET, and Visual Basic .NET. Refer to the table below for complete details.

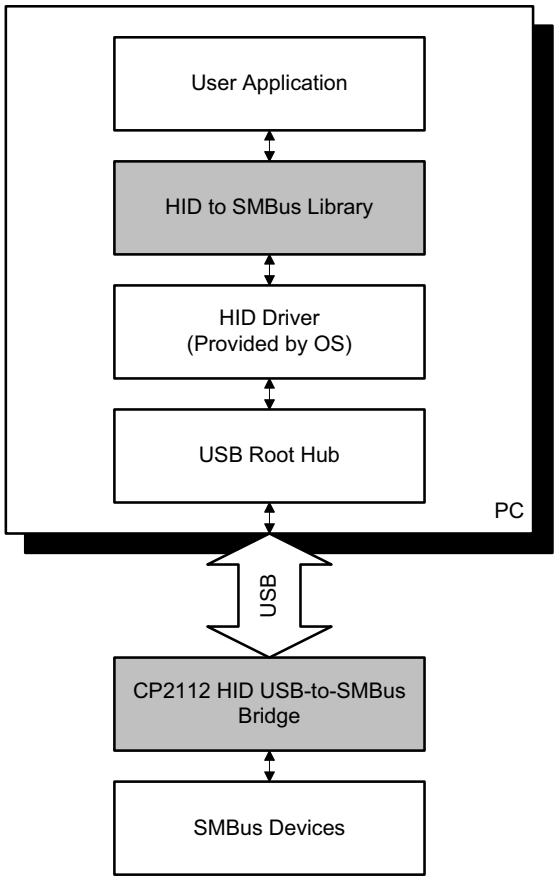

Figure 1. System Architecture Diagram

Table 1. CP2112 Include Files

Operating System	Library	Include Files	Version
Windows 2000 and later	SLABHIDtoSMBus.dll*	SLABCP2112.h (C/C++) SLABCP2112.cs (C# .NET) SLABCP2112.vb (VB .NET)	1.2
Mac OS X 10.5 and later	libSLABHIDtoSMBus.dylib	SLABCP2112.h (C, C++, Obj-C) Types.h (Compatibility)	1.0
***Note:** Requires SLABHIDDevice.dll version 1.5 during runtime.			

2. API Functions

Table 2. API Functions Table

Definition	Description	Page #
HidSmbus_GetNumDevices()	Returns the number of devices connected	3
HidSmbus_GetString()	Returns a string for a device by index	4
HidSmbus_GetOpenedString()	Returns a string for a device by device object pointer	5
HidSmbus_GetIndexedString()	Returns an indexed USB string descriptor by index (Windows Only)	6
HidSmbus_GetOpenedIndexedString()	Returns an indexed USB string descriptor by device object pointer (Windows Only)	6
HidSmbus_GetAttributes()	Returns the VID, PID, and release number for a device by index	7
HidSmbus_GetOpenedAttributes()	Returns the VID, PID and release number for a device by device object pointer	7
HidSmbus_Open()	Opens a device and returns a device object pointer	8
HidSmbus_Close()	Cancels pending IO and closes a device	8
HidSmbus_IsOpened()	Returns the device opened status	9
HidSmbus_ReadRequest()	Initiates a fixed length read request to the desired slave device	9
HidSmbus_AddressReadRequest()	Initiates a fixed length read request to the desired slave device specifying the memory address to read	10
HidSmbus_ForceReadResponse()	Forces the device to generate and send a read response	11
HidSmbus_GetReadResponse()	Returns a read response if available	12
HidSmbus_WriteRequest()	Initiates a write request to the desired slave device	13
HidSmbus_TransferStatusRequest()	Requests the status of the current read or write request	13
HidSmbus_GetTransferStatusResponse()	Returns the status of the current read or write request	14
HidSmbus_CancelTransfer()	Cancels the current read or write request	15
HidSmbus_CancelIo()	Cancels pending HID reads and writes (Windows Only)	16
HidSmbus_SetTimeouts()	Sets the response timeouts for a device	16
HidSmbus_GetTimeouts()	Gets the response timeouts for a device	16
HidSmbus_SetSmbusConfig()	Sets the bit rate, master address, timeouts, and transfer settings	17
HidSmbus_GetSmbusConfig()	Gets the bit rate, master address, timeouts, and transfer settings	18

Table 2. API Functions Table (Continued)

Definition	Description	Page #
HidSmbus_Reset()	Resets the device with re-enumeration	18
HidSmbus_SetGpioConfig()	Sets GPIO direction and mode configuration	19
HidSmbus_GetGpioConfig()	Gets GPIO direction and mode configuration	20
HidSmbus_ReadLatch()	Gets the GPIO latch value	21
HidSmbus_WriteLatch()	Sets the GPIO latch value using a bitmask	21
HidSmbus_GetPartNumber()	Gets the device part number and version	22
HidSmbus_GetLibraryVersion()	Gets the DLL Library version	22
HidSmbus_GetHidLibraryVersion()	Gets the HID Device Interface Library version	22
HidSmbus_GetHidGuid()	Gets the HID GUID (Windows® only)	22

2.1. HidSmbus_GetNumDevices

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

Prototype: `HID_SMBUS_STATUS HidSmbus_GetNumDevices (DWORD* numDevices, WORD vid, WORD pid)`

Parameters:
1. *numDevices* returns the number of devices connected on return.
2. *vid* filters device results by vendor ID. If both *vid* and *pid* are set to 0x0000, then HID devices will not be filtered by VID/PID.
3. *pid* filters device results by product ID. If both *vid* and *pid* are set to 0x0000, then HID devices will not be filtered by VID/PID.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_PARAMETER

2.2. HidSmbus_GetString

Description: This function returns a null-terminated vendor ID string, product ID string, serial string, device path string, manufacturer string, or product string for the device specified by an index passed in *deviceNum*. The index for the first device is 0, and the last device is the value returned by *HidSmbus_GetNumDevices()* – 1.

Prototype: `HID_SMBUS_STATUS HidSmbus_GetString (DWORD deviceNum, WORD vid, WORD pid, char* deviceString, DWORD options)`

Parameters:

1. *deviceNum* is the index of the device for which the string is desired.
2. *vid* filters device results by vendor ID. If both *vid* and *pid* are set to 0x0000, then HID devices will not be filtered by VID/PID.
3. *pid* filters device results by product ID. If both *vid* and *pid* are set to 0x0000, then HID devices will not be filtered by VID/PID.
4. *deviceString* is a variable of type HID_SMBUS_DEVICE_STR, which will contain a null-terminated ASCII device string on return. The string is 260 bytes on Windows and 512 bytes on Mac OS X.
5. *options* determines if *deviceString* will contain a vendor ID string, product ID string, serial string, device path string, manufacturer string, or product string.

Definition	Value	Length	Description
HID_SMBUS_GET_VID_STR	0x01	5	Vendor ID
HID_SMBUS_GET_PID_STR	0x02	5	Product ID
HID_SMBUS_GET_PATH_STR	0x03	260/512	Device path
HID_SMBUS_GET_SERIAL_STR	0x04	256	Serial string
HID_SMBUS_GET_MANUFACTURER_STR	0x05	256	Manufacturer String
HID_SMBUS_GET_PRODUCT_STR	0x06	256	Product String

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_DEVICE_NOT_FOUND
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_ACCESS_ERROR

2.3. HidSmbus_GetOpenedString

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

Prototype: `HID_SMBUS_STATUS HidSmbus_GetOpenedString (HID_SMBUS_DEVICE device, char* deviceString, DWORD options)`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *deviceString* is a variable of type HID_SMBUS_DEVICE_STR, which will contain a null-terminated ASCII device string on return. The string is 260 bytes on Windows and 512 bytes on Mac OS X.
3. *options* determines if *deviceString* will contain a vendor ID string, product ID string, serial string, device path string, manufacturer string, or product string.

Definition	Value	Length	Description
HID_SMBUS_ GET_VID_STR	0x01	5	Vendor ID
HID_SMBUS_ GET_PID_STR	0x02	5	Product ID
HID_SMBUS_ GET_PATH_STR	0x03	260/512	Device path
HID_SMBUS_ GET_SERIAL_STR	0x04	256	Serial string
HID_SMBUS_GET_MANUFACTURER_STR	0x05	256	Manufacturer String
HID_SMBUS_GET_PRODUCT_STR	0x06	256	Product String

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_ACCESS_ERROR

2.4. HidSmbus_GetIndexedString

Description: This function returns a null-terminated USB string descriptor for the device specified by an index passed in *deviceNum* (Windows Only).

Prototype: `HID_SMBUS_STATUS HidSmbus_GetIndexedString (DWORD deviceNum, WORD vid, WORD pid, DWORD stringIndex, char* deviceString)`

Parameters:

1. *deviceNum* is the index of the device for which the string is desired.
2. *vid* filters device results by vendor ID. If both *vid* and *pid* are set to 0x0000, then HID devices will not be filtered by VID/PID.
3. *pid* filters device results by product ID. If both *vid* and *pid* are set to 0x0000, then HID devices will not be filtered by VID/PID.
4. *stringIndex* specifies the device-specific index of the USB string descriptor to return.
5. *deviceString* is a variable of type HID_SMBUS_DEVICE_STR (260-byte ASCII string), which will contain a NULL terminated device descriptor string on return.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_DEVICE_NOT_FOUND
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_ACCESS_ERROR

2.5. HidSmbus_GetOpenedIndexedString

Description: This function returns a null-terminated USB string descriptor for the device specified by *device* (Windows Only).

Prototype: `HID_SMBUS_STATUS HidSmbus_GetOpenedIndexedString (HID_SMBUS_DEVICE device, DWORD stringIndex, char* deviceString)`

Parameters:

1. *deviceNum* is the device object pointer as returned by *HidSmbus_Open()*.
2. *stringIndex* specifies the device-specific index of the USB string descriptor to return.
3. *deviceString* is a variable of type HID_SMBUS_DEVICE_STR (260-byte ASCII string), which will contain a NULL terminated device descriptor string on return.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_ACCESS_ERROR

2.6. HidSmbus_GetAttributes

Description: This function returns the device vendor ID, product ID, and release number for the device specified by an index passed in *deviceNum*.

Prototype: `HID_SMBUS_STATUS HidSmbus_GetAttributes (DWORD deviceNum, WORD vid, WORD pid, WORD* deviceVid, WORD* devicePid, WORD* deviceReleaseNumber)`

Parameters:

1. *deviceNum* is the index of the device for which the string is desired.
2. *vid* filters device results by vendor ID. If both vid and pid are set to 0x0000, then HID devices will not be filtered by VID/PID.
3. *pid* filters device results by product ID. If both vid and pid are set to 0x0000, then HID devices will not be filtered by VID/PID.
4. *deviceVid* returns the device vendor ID.
5. *devicePid* returns the device product ID.
6. *deviceReleaseNumber* returns the USB device release number in binary-coded decimal.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_DEVICE_NOT_FOUND
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_ACCESS_ERROR

2.7. HidSmbus_GetOpenedAttributes

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

Prototype: `HID_SMBUS_STATUS HidSmbus_GetOpenedAttributes (HID_SMBUS_DEVICE device, WORD* deviceVid, WORD* devicePid, WORD* deviceReleaseNumber)`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *deviceVid* returns the device vendor ID.
3. *devicePid* returns the device product ID.
4. *deviceReleaseNumber* returns the USB device release number in binary-coded decimal.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_ACCESS_ERROR

2.8. HidSmbus_Open

Description: This function opens a device using a device number between 0 and *HidSmbus_GetNumDevices()* – 1 and returns a device object pointer that is used for subsequent accesses.

Prototype: `HID_SMBUS_STATUS HidSmbus_Open (HID_SMBUS_DEVICE* device, DWORD deviceNum, WORD vid, WORD pid)`

Parameters:

1. *device* returns a pointer to an HID USB-to-SMBus device object. This pointer will be used by all subsequent accesses to the device.
2. *deviceNum* is a zero-based device index, between 0 and *(HidSmbus_GetNumDevices()* – 1).
3. *vid* filters device results by vendor ID. If both *vid* and *pid* are set to 0x0000, then HID devices will not be filtered by VID/PID.
4. *pid* filters device results by product ID. If both *vid* and *pid* are set to 0x0000, then HID devices will not be filtered by VID/PID.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_DEVICE_NOT_FOUND
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED
HID_SMBUS_DEVICE_ACCESS_ERROR
HID_SMBUS_DEVICE_NOT_SUPPORTED

Remarks: Be careful when opening a device. Any HID device may be opened by this library. However, if the device is not a CP2112, use of this library will cause undesirable results. The best course of action is to designate a unique VID/PID for CP2112 devices only. The application should then filter devices using this VID/PID.

2.9. HidSmbus_Close

Description: This function closes an opened device using the device object pointer provided by *HidSmbus_Open()*.

Prototype: `HID_SMBUS_STATUS HidSmbus_Close (HID_SMBUS_DEVICE device)`

Parameters: *device* is the device object pointer as returned by *HidSmbus_Open()*.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_HANDLE
HID_SMBUS_DEVICE_ACCESS_ERROR

Remarks: *device* is invalid after calling *HidSmbus_Close()*. Set *device* to NULL.

2.10. HidSmbus_IsOpened

Description: This function returns the device opened status.

Prototype: `HID_SMBUS_STATUS HidSmbus_IsOpened (HID_SMBUS_DEVICE device, BOOL* opened)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *opened* returns *TRUE* if the device object pointer is valid and the device has been opened using *HidSmbus_Open()*.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER

2.11. HidSmbus_ReadRequest

Description: This function intiates a read transfer to the specified slave device address. Read and write timeouts as well as transfer retries can be set using *HidSmbus_SetSmbusConfig()* as described in "HidSmbus_SetSmbusConfig" on page 17.

Prototype: `HID_SMBUS_STATUS HidSmbus_ReadRequest (HID_SMBUS_DEVICE device, BYTE slaveAddress, WORD numBytesToRead)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *slaveAddress* is the address of the slave device to read from. This value must be between 0x02 - 0xFE. The least significant bit is the read/write bit for the SMBus transaction and must be 0.
3. *numBytesToRead* is the number of bytes to read from the device (1–512).

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_INVALID_REQUEST_LENGTH
HID_SMBUS_DEVICE_IO_FAILED

Remarks: *HidSmbus_ReadRequest()* initiates a read transfer. SMBus is a half-duplex bus, which means that only one read, address read, or write transfer can be active at a time. The device will attempt to read up to *transferRetries* number of times and for *readTimeout* milliseconds before timing out. See *HidSmbus_SetSmbusConfig()* for more information on configuring read timeouts. If the autoReadRespond setting is enabled, then call *HidSmbus_GetReadResponse()* to return the results of the read transfer.

2.12. HidSmbus_AddressReadRequest

Description: This function intiates a read transfer to the specified slave device address and specifies the address to read from on the device. Read and write timeouts as well as transfer retries can be set using *HidSmbus_SetSmbusConfig()* as described in "HidSmbus_SetSmbusConfig" on page 17.

Prototype: `HID_SMBUS_STATUS HidSmbus_AddressReadRequest (HID_SMBUS_DEVICE device, BYTE slaveAddress, WORD numBytesToRead, BYTE targetAddressSize, BYTE targetAddress[16])`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *slaveAddress* is the address of the slave device to read from. This value must be between 0x02 - 0xFE. The least significant bit is the read/write bit for the SMBus transaction and must be 0.
3. *numBytesToRead* is the number of bytes to read from the device (1–512).
4. *targetAddressSize* is the size of the target address in bytes (1-16).
5. *targetAddress* is the address to read from the slave device.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_INVALID_REQUEST_LENGTH
HID_SMBUS_DEVICE_IO_FAILED

Remarks: *HidSmbus_AddressReadRequest()* initiates a read transfer. SMBus is a half-duplex bus which means that only one read, address read, or write transfer can be active at a time. The device will attempt to read up to *transferRetries* number of times and for *readTimeout* milliseconds before timing out. See *HidSmbus_SetSmbusConfig()* for more information on configuring read timeouts. If the autoReadRespond setting is enabled, then call *HidSmbus_GetReadResponse()* to return the results of the read transfer. The device will transmit the target address on the bus after the slave device has acknowledged its address. This function is designed to read from EEPROMs with an SMBus interface.

2.13. HidSmbus_ForceReadResponse

Description: This function causes the device to send a read response to the host after a read transfer has been issued.

Prototype: `HID_SMBUS_STATUS HidSmbus_ForceReadResponse (HID_SMBUS_DEVICE device, WORD numBytesToRead)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *numBytesToRead* is the number of bytes to read from the device (1–512).

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_REQUEST_LENGTH
HID_SMBUS_DEVICE_IO_FAILED

Remarks: *HidSmbus_ForceReadResponse()* should only be called if *autoReadRespond* is disabled using *HidSmbus_SetSmbusConfig()*. This allows the user to read data in a polled mode.
Call *HidSmbus_ReadRequest()* or *HidSmbus_AddressReadRequest()* first.
Next, call *HidSmbus_TransferStatusRequest()* and *HidSmbus_TransferStatusResponse()* to check if the device has received data.
Next, call *HidSmbus_ForceReadResponse()*.
Finally, call *HidSmbus_GetReadResponse()* repeatedly until all read data is returned.
Typically, this procedure is not necessary as users should enable the *autoReadRespond* setting.

2.14. HidSmbus_GetReadResponse

Description: This function returns the read response to a read request. Read and write timeouts as well as transfer retries can be set using *HidSmbus_SetSmbusConfig()* as described in "HidSmbus_SetSmbusConfig" on page 17.

Prototype: `HID_SMBUS_STATUS HidSmbus_GetReadResponse (HID_SMBUS_DEVICE device, HID_SMBUS_S0* status, BYTE* buffer, BYTE bufferSize, BYTE* numBytesRead)`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *status* returns the status of the read request.

Definition	Value	Description
HID_SMBUS_S0_IDLE	0x00	No transfers are currently active on the bus.
HID_SMBUS_S0_BUSY	0x01	A read or write transfer is in progress.
HID_SMBUS_S0_COMPLETE	0x02	A read or write transfer completed without error and without retry.
HID_SMBUS_S0_ERROR	0x03	A read or write transfer completed with an error.

3. *buffer* returns up to 61 read data bytes.
4. *bufferSize* is the size of *buffer* and must be at least 61 bytes.
5. *numBytesRead* returns the number of valid data bytes returned in *buffer*.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_READ_TIMED_OUT
HID_SMBUS_READ_ERROR

Remarks: *HidSmbus_GetReadResponse()* waits for up to *readTimeout* milliseconds for the device to send a read response interrupt report to the host. This function should be called repeatedly until all read data has been received or an error occurs. Call *HidSmbus_ReadRequest()* or *HidSmbus_AddressReadRequest()* followed by *HidSmbus_GetReadResponse()* to read data when *autoReadResponse* is enabled using *HidSmbus_SetSmbusConfig()*. *HidSmbus_GetReadResponse()* will wait for up to *responseTimeout* milliseconds before returning *HID_SMBUS_READ_TIMED_OUT*

2.15. HidSmbus_WriteRequest

Description: This function writes the specified number of bytes from the supplied buffer to the specified slave device and returns immediately after sending the request to the CP2112. Read and write timeouts can be set using *HidSmbus_SetTimeouts()* as described in "2.20. HidSmbus_SetTimeouts" on page 16.

Prototype: `HID_SMBUS_STATUS HidSmbus_WriteRequest (HID_SMBUS_DEVICE device, BYTE slaveAddress, BYTE* buffer, BYTE numBytesToWrite)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *slaveAddress* is the address of the slave device to write to. This value must be between 0x02 - 0xFE. The least significant bit is the read/write bit for the SMBus transaction and must be 0.
3. *buffer* is the address of a buffer to be sent to the device.
4. *numBytesToWrite* is the number of bytes to write to the device (1–61). This value must be less than or equal to the size of buffer.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_INVALID_REQUEST_LENGTH
HID_SMBUS_DEVICE_IO_FAILED

Remarks: Call *HidSmbus_TransferStatusRequest()*/*HidSmbus_GetTransferStatusResponse()* to wait for the write transfer to complete before issuing another transfer request. The device waits for up to *transferRetries* number of retries and *writeTimeout* number of milliseconds before timing out.

2.16. HidSmbus_TransferStatusRequest

Description: This function requests the status of the current read or write transfer.

Prototype: `HID_SMBUS_STATUS HidSmbus_TransferStatusRequest (HID_SMBUS_DEVICE device)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_DEVICE_IO_FAILED

Remarks: Call *HidSmbus_TransferStatusRequest()* followed by *HidSmbus_GetTransferStatusResponse()* to get the status of the current read or write transfer.

2.17. HidSmbus_GetTransferStatusResponse

Description: This function returns the status of the current read or write transfer.

Prototype: `HID_SMBUS_STATUS HidSmbus_GetTransferStatusResponse (HID_SMBUS_DEVICE device, HID_SMBUS_S0* status, HID_SMBUS_S1* detailedStatus, WORD* numRetries, WORD* bytesRead)`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *status* returns the status of the read or write transfer.

Definition	Value	Description
HID_SMBUS_S0_IDLE	0x00	No transfers are currently active on the bus
HID_SMBUS_S0_BUSY	0x01	A read or write transfer is in progress
HID_SMBUS_S0_COMPLETE	0x02	A read or write transfer completed without error and without retry
HID_SMBUS_S0_ERROR	0x03	A read or write transfer completed with an error

3. *detailedStatus* returns the extended status of the read or write transfer.

Definition	Value	Description
HID_SMBUS_S1_BUSY_ADDRESS_ACKED	0x00	The slave address was acknowledged
HID_SMBUS_S1_BUSY_ADDRESS_NACKED	0x01	The slave address has not been acknowledged
HID_SMBUS_S1_BUSY_READING	0x02	Read data phase in progress
HID_SMBUS_S1_BUSY_WRITING	0x03	Write data phase in progress

Detailed Status for Status = HID_SMBUS_S0_ERROR		
Definition	**Value**	**Description**
HID_SMBUS_S1_ERROR_TIMEOUT_NACK	0x00	Tranfer timeout: SMBus slave address was NACKed
HID_SMBUS_S1_ERROR_TIMEOUT_BUS_NOT_FREE	0x01	Tranfser timeout: SMBus not free (or SCL low timeout occurred)
HID_SMBUS_S1_ERROR_ARB_LOST	0x02	Bus arbitration was lost
HID_SMBUS_S1_ERROR_READ_INCOMPLETE	0x03	Read was incomplete
HID_SMBUS_S1_ERROR_WRITE_INCOMPLETE	0x04	Write was incomplete
HID_SMBUS_S1_ERROR_SUCCESS_AFTER_RETRY	0x05	Transfer completed after *numRetries* number of retries

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_READ_TIMED_OUT
HID_SMBUS_READ_ERROR

Remarks: Call *HidSmbus_TransferStatusRequest()* followed by *HidSmbus_GetTransferStatusResponse()* to get the status of the current read or write transfer. *HidSmbus_GetTransferStatusResponse()* will wait for up to *responseTimeout* milliseconds before returning *HID_SMBUS_READ_TIMED_OUT*.

2.18. HidSmbus_CancelTransfer

Description: This function cancels the current read or write transfer.

Prototype: `HID_SMBUS_STATUS HidSmbus_TransferStatusRequest (HID_SMBUS_DEVICE device)`

Parameters: 1.*device* is the device object pointer as returned by *HidSmbus_Open()*.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_DEVICE_IO_FAILED

Remarks: This function will clear any read responses received.

2.19. HidSmbus_Cancello

Description: This function cancels any pending HID reads and writes (Windows Only).

Prototype: `HID_SMBUS_STATUS HidSmbus_CancelIo (HID_SMBUS_DEVICE device)`

Parameters: *device* is the device object pointer as returned by *HidSmbus_Open()*.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_DEVICE_IO_FAILED

2.20. HidSmbus_SetTimeouts

Description: This function sets the response timeouts. Response timeouts are used by *HidSmbus_GetReadResponse()* and *HidSmbus_GetTransferStatusResponse()*. The default value for response timeouts is 1000 ms, but timeouts can be set to wait for any number of milliseconds between 1 and 0xFFFFFFFF. Specifying a response timeout of 0, will wait indefinitely.

Prototype: `HID_SMBUS_STATUS HidSmbus_SetTimeouts (HID_SMBUS_DEVICE device, DWORD responseTimeout)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *responseTimeout* is the *HidSmbus_GetReadResponse()* and *HidSmbus_GetTransferStatusResponse()* timeout.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT

Remarks: If timeouts are set to a large value and no data is received, the application may appear unresponsive. It is recommended to set timeouts appropriately before using the device. Typically, users will want to specify a response timeout that is greater than the read and write timeouts.

2.21. HidSmbus_GetTimeouts

Description: This function returns the current response timeouts specified in milliseconds. A response timeout value of 0 indicates an infinite timeout.

Prototype: `HID_SMBUS_STATUS HidSmbus_GetTimeouts (HID_SMBUS_DEVICE device, DWORD* responseTimeout)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *responseTimeout* is the response operation timeout in milliseconds.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER

Remarks: Timeouts are maintained for each device but are not persistent across *HidSmbus_Open()*/ *HidSmbus_Close()*.

2.22. HidSmbus_SetSmbusConfig

Description: This function sets the SMBus bit rate, address, and transfer settings such as timeouts and retries. Refer to the device data sheet for a list of supported configuration settings.

Prototype: `HID_SMBUS_STATUS HidSmbus_SetSmbusConfig (HID_SMBUS_DEVICE device, DWORD bitRate, BYTE address, BOOL autoReadRespond, WORD writeTimeout, WORD readTimeout, BOOL sclLowTimeout, WORD transferRetries)`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *bitRate* is the bit rate for SMBus communication. The default is 100 kHz. This value must be non-zero.
3. *address* is the device's slave address (0x02– 0xFE). The device will only acknowledge this address. The default is 0x02. The least significant bit is the read/write bit for the SMBus transaction and must be 0.
4. *autoReadRespond* controls the read response behavior of the device. If enabled, the device will automatically send read response interrupt reports to the device after initiating a read transfer. If disabled, the user must call *HidSmbus_ForceReadResponse()* before read response interrupt reports will be sent to the host. The default is *FALSE* (0).
5. *writeTimeout* is the time limit in milliseconds (0–1000) before the device will automatically cancel a write transfer. A value of 0 indicates an infinite timeout. In this case, a write transfer will wait indefinitely for a write to complete or until *HidSmbus_CancelTransfer()* is called. The default is 0.
6. *readTimeout* is the time limit in milliseconds (0 - 1000) before the device will automatically cancel a read transfer. A value of 0 indicates an infinite timeout. In this case, a read transfer will wait indefinitely for a read to complete or until *HidSmbus_CancelTransfer()* is called. The default is 0.
7. *sclLowTimeout* is a timeout that will reset the SMBus if the SCL line is held low for more than 25 ms. If enabled and an SCL Low Timeout occurs, the status byte of the Transfer Status Response command will be set appropriately. The default is *FALSE* (0).
8. *transferRetries* is the number of times to retry (0 - 1000) a failed read or write transfer. A value of 0 indicates an infinite number of retries until the specified read or write timeout has elapsed. The default is 0.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

2.23. HidSmbus_GetSmbusConfig

Description: This function gets the SMBus bit rate, address, and transfer settings such as timeouts and retries. Refer to the device data sheet for a list of supported configuration settings.

Prototype: `HID_SMBUS_STATUS HidSmbus_GetSmbusConfig (HID_SMBUS_DEVICE device, DWORD* bitRate, BYTE* address, BOOL* autoReadRespond, WORD* writeTimeout, WORD* readTimeout, BOOL* sclLowTimeout, WORD* transferRetries)`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *bitRate* returns the bit rate for SMBus communication. The default is 100 kHz. This value must be non-zero.
3. *address* returns the device's slave address (0x02–0xFE). The device will only acknowledge this address. The default is 0x02.
4. *autoReadRespond* returns the read response behavior of the device. If auto read respond is enabled, then the device will automatically send read response interrupt reports to the device after initiating a read transfer. If disabled, the user must call *HidSmbus_ForceReadResponse()* before read response interrupt reports will be sent to the host. The default is *FALSE* (0).
5. *writeTimeout* returns the time limit in milliseconds (0–1000) before the device will automatically cancel a write transfer. A value of 0 indicates an infinite timeout. In this case, a write transfer will wait indefinitely for a write to complete or until *HidSmbus_CancelTransfer()* is called. The default is 0.
6. *readTimeout* returns the time limit in milliseconds (0–1000) before the device will automatically cancel a read transfer. A value of 0 indicates an infinite timeout. In this case, a read transfer will wait indefinitely for a read to complete or until *HidSmbus_CancelTransfer()* is called. The default is 0.
7. *sclLowTimeout* is a timeout that will reset the SMBus if the SCL line is held low for more than 25 ms. If enabled and an SCL Low Timeout occurs, the status byte of the Transfer Status Response command will be set appropriately. The default is *FALSE* (0).
8. *transferRetries* returns the number of times to retry (0–1000) a failed read or write transfer. A value of 0 indicates an infinite number of retries until the specified read or write timeout has elapsed. The default is 0.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

2.24. HidSmbus_Reset

Description: This function initiates a full device reset. All configuration settings will be reset to their default values after the device re-enumerates.

Prototype: `HID_SMBUS_STATUS HidSmbus_Reset (HID_SMBUS_DEVICE device)`

Parameters: *device* is the device object pointer as returned by *HidSmbus_Open()*.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_DEVICE_IO_FAILED

Remarks: Resetting the device will make the device's handle stale. Users must close the device using the old handle before proceeding to reconnect to the device. See more information on surprise removal. See *HidSmbus_SetSmbusConfig()* and *HidSmbus_SetGpioConfig()* for default configuration settings.

2.25. HidSmbus_SetGpioConfig

Description: This function configures the GPIO pins' directions and modes.

Prototype: `HID_SMBUS_STATUS HidSmbus_SetGpioConfig (HID_SMBUS_DEVICE device, BYTE direction, BYTE mode, BYTE special, BYTE clkDiv)`

Parameters: 1. *device* is the device object pointer as returned by *HidSmbus_Open()*.

2. *direction* is a bitmask that specifies each GPIO pin's direction.

.

Definition	Bit Value	Description
HID_SMBUS_DIRECTION_INPUT	0	Input
HID_SMBUS_DIRECTION_OUTPUT	1	Output

3. *mode* is a bitmask that specifies each GPIO pin's mode.

Definition	Bit Value	Description
HID_SMBUS_MODE_OPEN_DRAIN	0	Open-Drain
HID_SMBUS_MODE_PUSH_PULL	1	Push-Pull

4. *special* is a bitmask that specifies the special behavior of GPIO.0, GPIO.1, and GPIO.7

Definition	Value	Description
HID_SMBUS_MASK_FUNCTION_GPIO_7_CLK	0x01	Enables or disables the clock output function of GPIO.7
HID_SMBUS_MASK_FUNCTION_GPIO_0_TXT	0x02	Enables or disables the TX toggle function of GPIO.0
HID_SMBUS_MASK_FUNCTION_GPIO_1_RXT	0x04	Enables or disables the RX toggle function of GPIO.1

Definition	Bit Value	Description
HID_SMBUS_GPIO_FUNCTION	0	GPIO function as configured using *direction* and *mode*
HID_SMBUS_SPECIAL_FUNCTION	1	Special function: GPIO.0 - TX Toggle (push-pull output) GPIO.1 - RX Toggle (push-pull output) GPIO.7 - Clock Output (push-pull output)

5. *clkDiv* is the clock output divider value used for GPIO.7 when configured in clock output mode. The frequency is equal to 48 MHz / (2 x *clkDiv*) when *clkDiv* is between 1 and 255 and equal to 48 MHz when *clkDiv* is 0.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_DEVICE_IO_FAILED

2.26. HidSmbus_GetGpioConfig

Description: This function returns the GPIO pins' directions and modes.

Prototype: `HID_SMBUS_STATUS HidSmbus_GetGpioConfig (HID_SMBUS_DEVICE device, BYTE* direction, BYTE* mode, BYTE* special, BYTE* clkDiv)`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *direction* returns a bitmask that specifies each GPIO pin's direction.
3. *mode* returns a bitmask that specifies each GPIO pin's mode.

Definition	Bit Value	Description
HID_SMBUS_MODE_OPEN_DRAIN	0	Open-Drain
HID_SMBUS_MODE_PUSH_PULL	1	Push-Pull

4. *special* returns a bitmask that specifies the special behavior of GPIO.0, GPIO.1, and GPIO.7.

Definition	Value	Description
HID_SMBUS_MASK_FUNCTION_GPIO_7_CLK	0x01	Enables or disables the clock output function of GPIO.7
HID_SMBUS_MASK_FUNCTION_GPIO_0_TXT	0x02	Enables or disables the TX toggle function of GPIO.0
HID_SMBUS_MASK_FUNCTION_GPIO_1_RXT	0x04	Enables or disables the RX toggle function of GPIO.1

Definition	Bit Value	Description
HID_SMBUS_GPIO_FUNCTION	0	GPIO function as configured using *direction* and *mode*
HID_SMBUS_SPECIAL_FUNCTION	1	Special function: GPIO.0 - TX Toggle (push-pull output) GPIO.1 - RX Toggle (push-pull output) GPIO.7 - Clock Output (push-pull output)

5. *clkDiv* returns the clock output divider value used for GPIO.7 when configured in clock output mode. The frequency is equal to 48 MHz / (2 x *clkDiv*) when *clkDiv* is between 1 and 255 and equal to 48 MHz when *clkDiv* is 0.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

2.27. HidSmbus_ReadLatch

Description: This function returns the current GPIO latch value.

Prototype: `HID_SMBUS_STATUS HidSmbus_ReadLatch (HID_SMBUS_DEVICE device, BYTE* latchValue)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *latchValue* returns the current GPIO latch value.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

Remarks: If a pin is configured as an input, then the *latchValue* bit represents the logical voltage level received on the pin. If a pin is configured as an output, then the *latchValue* bit represents the logical voltage level driven on the pin.

2.28. HidSmbus_WriteLatch

Description: This function sets the current GPIO latch value for the specified bits.

Prototype: `HID_SMBUS_STATUS HidSmbus_WriteLatch (HID_SMBUS_DEVICE device, BYTE latchValue, BYTE latchMask)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *latchValue* is the output value to drive on GPIO pins configured as outputs.
3. *latchMask* is the bitmask specifying which bits to modify.

Definition	Value
HID_SMBUS_MASK_GPIO_0	0x01
HID_SMBUS_MASK_GPIO_1	0x02
HID_SMBUS_MASK_GPIO_2	0x04
HID_SMBUS_MASK_GPIO_3	0x08
HID_SMBUS_MASK_GPIO_4	0x10
HID_SMBUS_MASK_GPIO_5	0x20
HID_SMBUS_MASK_GPIO_6	0x40
HID_SMBUS_MASK_GPIO_7	0x80

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_DEVICE_IO_FAILED

Remarks: Only GPIO pins configured as outputs with their corresponding *latchMask* bits set can be written to.

2.29. HidSmbus_GetPartNumber

Description: This function retrieves the part number and version of the CP2112 device.

Prototype: `HID_SMBUS_STATUS HidSmbus_GetPartNumber (HID_SMBUS_DEVICE device, BYTE* partNumber, BYTE* version)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *partNumber* returns the device part number.

Definition	Value	Description
HID_SMBUS_PART_CP2112	0x0C	CP2112

3. *version* returns the version. This value is not user-programmable.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

2.30. HidSmbus_GetLibraryVersion

Description: This function returns the HID USB-to-SMBus Interface Library version.

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

Parameters:
1. *major* returns the major library version number. This value ranges from 0 to 255.
2. *minor* returns the minor library version number. This value ranges from 0 to 255.
3. *release* returns *TRUE* if the library is a release build; otherwise, the library is a Debug build.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_PARAMETER

2.31. HidSmbus_GetHidLibraryVersion

Description: This function returns the version of the HID Device Interface Library that is currently in use.

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

Parameters:
1. *major* returns the major library version number. This value ranges from 0 to 255.
2. *minor* returns the minor library version number. This value ranges from 0 to 255.
3. *release* returns *TRUE* if the library is a release build; otherwise, the library is a Debug build.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_PARAMETER

2.32. HidSmbus_GetHidGuid

Description: This function obtains the HID GUID. This can be used to register for surprise removal notifications (Windows Only).

Prototype: `HID_SMBUS_STATUS HidSmbus_GetHidGuid (void* guid)`

Parameters:
1. *guid* returns the HID GUID.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_PARAMETER

3. User Customization API Functions

The following parameters are programmable on the device. Different functions are provided to program these parameters. Each parameter may only be programmed once and only if the parameter is not locked.

Name	Size	Short Description
VID	2	USB Vendor ID
PID	2	USB Product ID
Power	1	Power request in mA/2
Power Mode	1	Bus Powered Self Powered - Regulator Off Self Powered - Regulator On
Release Version	2	Major and Minor release version
Manufacturer String	60	Product Manufacturer (English Unicode)
Product Description String	60	Product Description (English Unicode)
Serial String	60	Serialization String (English Unicode)

The following API functions are provided to allow user customization/one-time programming:

Definition	Description	Page #
HidSmbus_SetLock()	Prevents further OTP programming/customization	24
HidSmbus_GetLock()	Gets the OTP lock status	25
HidSmbus_SetUsbConfig()	Sets VID, PID, power, power mode, and release version	26
HidSmbus_GetUsbConfig()	Gets VID, PID, power, power mode, and release version	27
HidSmbus_SetManufacturingString()	Sets the USB manufacturing string	28
HidSmbus_GetManufacturingString()	Gets the USB manufacturing string	28
HidSmbus_SetProductString()	Sets the USB product string	28
HidSmbus_GetProductString()	Gets the USB product string	29
HidSmbus_SetSerialString()	Sets the USB serial string	29
HidSmbus_GetSerialString()	Gets the USB serial string	29

3.1. HidSmbus_SetLock

Description: This function permanently locks/disables device customization.
Prototype: `HID_SMBUS_STATUS HidSmbus_SetLock (HID_SMBUS_DEVICE device, BYTE lock)`
Parameters: 1. *device* is the device object pointer as returned by *HidSmbus_Open()*.

2. *lock* is the bitmask specifying which fields can be customized/programmed and which fields are already customized.

Bit	Definition	Mask	Description
0	HID_SMBUS_LOCK_VID	0x01	VID
1	HID_SMBUS_LOCK_PID	0x02	PID
2	HID_SMBUS_LOCK_POWER	0x04	Power
3	HID_SMBUS_LOCK_POWER_MODE	0x08	Power Mode
4	HID_SMBUS_LOCK_RELEASE_VERSION	0x10	Release Version
5	HID_SMBUS_LOCK_MFG_STR	0x20	Manufacturing String
6	HID_SMBUS_LOCK_PRODUCT_STR	0x40	Product String
7	HID_SMBUS_LOCK_SERIAL_STR	0x80	Serial String

Definition	Bit Value	Description
HID_SMBUS_LOCK_UNLOCKED	1	Field can be customized
HID_SMBUS_LOCK_LOCKED	0	Field has already been customized or has been locked

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_DEVICE_IO_FAILED

Remarks: When this function is successfully called, the specified fields are fully locked and cannot be further customized. The user customization functions can be called and may return *HID_SMBUS_SUCCESS* even though the device was not programmed. Call the function's corresponding get function to verify that customization was successful. Each field is stored in one time programmable memory (OTP) and can only be customized once. After a field is customized, the corresponding lock bits are set to 0.

3.2. HidSmbus_GetLock

Description: This function returns the device customization lock status.
Prototype: `HID_SMBUS_STATUS HidSmbus_GetLock (HID_SMBUS_DEVICE device, BYTE* lock)`
Parameters: 1. *device* is the device object pointer as returned by *HidSmbus_Open()*.

2. *lock* returns a bitmask specifying which fields are locked.

Bit	Definition	Mask	Description
0	HID_SMBUS_LOCK_VID	0x01	VID
1	HID_SMBUS_LOCK_PID	0x02	PID
2	HID_SMBUS_LOCK_POWER	0x04	Power
3	HID_SMBUS_LOCK_POWER_MODE	0x08	Power Mode
4	HID_SMBUS_LOCK_RELEASE_VERSION	0x10	Release Version
5	HID_SMBUS_LOCK_MFG_STR	0x20	Manufacturing String
6	HID_SMBUS_LOCK_PRODUCT_STR	0x40	Product String
7	HID_SMBUS_LOCK_SERIAL_STR	0x80	Serial String

Definition	Bit Value	Description
HID_SMBUS_LOCK_UNLOCKED	1	Field can be customized
HID_SMBUS_LOCK_LOCKED	0	Field has already been customized or has been locked

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

3.3. HidSmbus_SetUsbConfig

Description: This function allows one-time customization of the USB configuration, which includes vendor ID, product ID, power, power mode, and release version settings. Each field can be independently programmed one time via the mask field.

Prototype: `HID_SMBUS_STATUS HidSmbus_SetUsbConfig (HID_SMBUS_DEVICE device, WORD vid, WORD pid, BYTE power, BYTE powerMode, WORD releaseVersion, BYTE mask)`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *vid* is the vendor ID.
3. *pid* is the product ID.
4. *power* specifies the current requested by the device in milliamps/2. The maximum power setting is 500 mA or 250 (0xFA). This value only applies when the device is configured to be bus-powered.
5. *powerMode* configures the device as bus-powered or self-powered.

Definition	Value	Description
HID_SMBUS_BUS_POWER	0x00	Device is bus powered
HID_SMBUS_SELF_POWER_VREG_DIS	0x01	Device is self powered (voltage regulator disabled)
HID_SMBUS_SELF_POWER_VREG_EN	0x02	Device is self powered (voltage regulator enabled)

6. *releaseVersion* is the user-programmable release version. The MSB is the major revision and the LSB is the minor revision. Both revisions can be programmed to any value from 0 to 255. This version is not the same as the device release number described in the USB device descriptor.
7. *mask* is the bitmask specifying which fields to customize.

Bit	Definition	Mask	Description
0	HID_SMBUS_SET_VID	0x01	VID
1	HID_SMBUS_SET_PID	0x02	PID
2	HID_SMBUS_SET_POWER	0x04	Power
3	HID_SMBUS_SET_POWER_MODE	0x08	Power Mode
4	HID_SMBUS_SET_RELEASE_VERSION	0x10	Release Version

Definition	Bit Value	Description
HID_SMBUS_SET_IGNORE	0	Field will be unchanged
HID_SMBUS_SET_PROGRAM	1	Field will be programmed

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

3.4. HidSmbus_GetUsbConfig

Description: This function retrieves USB configuration, which includes vendor ID, product ID, power, power mode, release version, and flush buffers settings.

Prototype: `HID_SMBUS_STATUS HidSmbus_GetUsbConfig (HID_SMBUS_DEVICE device, WORD* vid, WORD* pid, BYTE* power, BYTE* powerMode, WORD* releaseVersion)`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *vid* returns the vendor ID.
3. *pid* returns the product ID.
4. *power* returns the current requested by the device in milliamps / 2. This value only applies when the device is bus-powered.
5. *powerMode* returns the device power mode.

Definition	Value	Description
HID_SMBUS_BUS_POWER	0x00	Device is bus powered
HID_SMBUS_SELF_POWER_VREG_DIS	0x01	Device is self powered (voltage regulator disabled)
HID_SMBUS_SELF_POWER_VREG_EN	0x02	Device is self powered (voltage regulator enabled)

6. *releaseVersion* returns the user-programmable release version. The MSB is the major revision, and the LSB is the minor revision. Both revisions can be programmed to any value from 0 to 255. This version is not the same as the device release number described in the USB device descriptor.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

3.5. HidSmbus_SetManufacturingString

Description: This function allows one-time customization of the USB manufacturing string.

Prototype: `HID_SMBUS_STATUS HidSmbus_SetManufacturingString (HID_SMBUS_DEVICE device, char* manufacturingString, BYTE strlen)`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *manufacturingString* is a variable of type HID_SMBUS_CP2112_MFG_STR, a 30-byte character buffer containing the ASCII manufacturing string.
3. *strlen* is the length of *manufacturingString* in bytes. The maximum string length is 30 bytes.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

3.6. HidSmbus_GetManufacturingString

Description: This function retrieves the USB manufacturing string.

Prototype: `HID_SMBUS_STATUS HidSmbus_GetManufacturingString (HID_SMBUS_DEVICE device, char* manufacturingString, BYTE* strlen)`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *manufacturingString* is a variable of type HID_SMBUS_CP2112_MFG_STR, a 30-byte character buffer that will contain the ASCII manufacturing string.
3. *strlen* returns the length of the string in bytes.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

3.7. HidSmbus_SetProductString

Description: This function allows one-time customization of the USB product string.

Prototype: `HID_SMBUS_STATUS HidSmbus_SetProductString (HID_SMBUS_DEVICE device, char* productString, BYTE strlen)`

Parameters:

1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *productString* is a variable of type HID_SMBUS_CP2112_PRODUCT_STR, a 30-byte character buffer containing the ASCII product string.
3. *strlen* is the length of *productString* in bytes. The maximum string length is 30 bytes.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

3.8. HidSmbus_GetProductString

Description: This function retrieves the USB product string.

Prototype: `HID_SMBUS_STATUS HidSmbus_GetProductString (HID_SMBUS_DEVICE device, char* productString, BYTE* strlen)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *productString* is a variable of type HID_SMBUS_CP2112_PRODUCT_STR, a 30-byte character buffer that will contain the ASCII product string.
3. *strlen* returns the length of the string in bytes.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

3.9. HidSmbus_SetSerialString

Description: This function allows one-time customization of the USB serial string.

Prototype: `HID_SMBUS_STATUS HidSmbus_SetSerialString (HID_SMBUS_DEVICE device, char* serialString, BYTE strlen)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *serialString* is a variable of type HID_SMBUS_CP2112_SERIAL_STR, a 30-byte character buffer containing the ASCII serial string.
3. *strlen* is the length of *serialString* in bytes. The maximum string length is 30 bytes.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

3.10. HidSmbus_GetSerialString

Description: This function retrieves the USB product string.

Prototype: `HID_SMBUS_STATUS HidSmbus_GetSerialString (HID_SMBUS_DEVICE device, char* serialString, BYTE* strlen)`

Parameters:
1. *device* is the device object pointer as returned by *HidSmbus_Open()*.
2. *serialString* is a variable of type HID_SMBUS_CP2112_SERIAL_STR, a 30-byte character buffer that will contain the ASCII product string.
3. *strlen* returns the length of the string in bytes.

Return Value: HID_SMBUS_STATUS = HID_SMBUS_SUCCESS
HID_SMBUS_INVALID_DEVICE_OBJECT
HID_SMBUS_INVALID_PARAMETER
HID_SMBUS_DEVICE_IO_FAILED

4. HID_SMBUS_STATUS Returns Codes

Each library function returns an HID_SMBUS_STATUS return code to indicate that the function returned successfully or to describe an error. Table 3 describes each error code.

Table 3. Error Code Descriptions

Definition	Value	Description
HID_SMBUS_SUCCESS	0x00	Function returned successfully.*
HID_SMBUS_DEVICE_NOT_FOUND	0x01	Indicates that no devices are connected or that the specified device does not exist.
HID_SMBUS_INVALID_HANDLE	0x02	Indicates that the handle value is NULL or INVALID_HANDLE_VALUE or that the device with the specified handle does not exist.
HID_SMBUS_INVALID_DEVICE_OBJECT	0x03	Indicates that the device object pointer does not match the address of a valid HID USB-to-SMBus device.
HID_SMBUS_INVALID_PARAMETER	0x04	Indicates that a pointer value is NULL or that an invalid setting was specified.
HID_SMBUS_INVALID_REQUEST_LENGTH	0x05	Indicates that the specified number of bytes to read or write is invalid. Check the read and write length limits.
HID_SMBUS_READ_ERROR	0x10	Indicates that the read was not successful and did not time out. This means that the host could not get an input interrupt report.
HID_SMBUS_WRITE_ERROR	0x11	Indicates that the write was not successful. This means that the output interrupt report failed or timed out.
HID_SMBUS_READ_TIMED_OUT	0x12	Indicates that a read failed to return the number of bytes requested before the read timeout elapsed. The read timeout should be increased.
HID_SMBUS_WRITE_TIMED_OUT	0x13	Indicates that a write failed to complete sending the number of bytes requested before the write timeout elapsed. The write timeout should be increased.
HID_SMBUS_DEVICE_IO_FAILED	0x14	Indicates that the host was unable to get or set a feature report. The device might be disconnected.
HID_SMBUS_DEVICE_ACCESS_ERROR	0x15	Indicates that the device or device property could not be accessed. Either the device is not opened, already opened when trying to open, or an error occurred while trying to get HID information.
HID_SMBUS_DEVICE_NOT_SUPPORTED	0x16	Indicates that the current device does not support the corresponding action. Functions listed in this document are for the CP2112 only.
HID_SMBUS_UNKNOWN_ERROR	0xFF	This is the default return code value. This value should never be returned.
***Note:** Set functions may return success, indicating that the device received the request; however, there is no indication that the device actually performed the request (i.e., the setting was invalid). The user must call the corresponding get function to verify that the settings were properly configured.		

5. Thread Safety

The HID USB-to-SMBus 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 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, then the user must prevent another thread from calling any function until the first function returns.
2. HidSmbus_GetReadResponse(), HidSmbus_GetTransferStatusResponse(), HidSmbus_TransferStatusRequest(), and HidSmbus_CancelTransfer() issue pending read requests that cannot be canceled from another thread. If the user calls HidSmbus_Close() in a different thread than the thread in which the read request was created, then the device will not be accessible after calling HidSmbus_Close(). The thread that issued the pending read request must return/terminate successfully before the device can be accessed again. See “6. Thread Read Access Models (For Windows)” on page 32 for more information.

6. Thread Read Access Models (For Windows)

There are several common read access models when using the HID USB-to-SMBus library. There are some restrictions on the valid use of a device handle based on these models. *CancelIo()* can only cancel pending I/O (reads/writes) issued in the same thread in which *CancelIo()* is called. Due to this limitation, the user is responsible for cancelling pending I/O before closing the device. Failure to do so will result in an inaccessible HID USB-to-SMBus device until the thread releases access to the device handle. The following tables describe five common access models and the expected behavior.

Notes:

1. *HidSmbus_Close()* calls *CancelIo()* prior to calling *CloseHandle()*.
2. *QueueInterruptReports()* issues a pending interrupt report read request. The request completes if at least one input report is read. The request is still pending if the operation times out. The following functions call *QueueInterruptReports()*:
   HidSmbus_GetReadResponse()
   HidSmbus_GetTransferStatusResponse()
   HidSmbus_TransferStatusRequest()
   HidSmbus_CancelTransfer()
3. *HidSmbus_CancelIo()* forces any pending requests issued by the same thread to complete (cancelled).
4. ***** indicates that a read is still pending and was issued in the specified thread.
5. **?** indicates that a read is still pending and was issued in one of the threads (indeterminate).

Table 4. Single Thread Access Model (Safe)

Thread A	Thread B	Result
HidSmbus_Open()	—	—
*QueueInterruptReports()**	—	—
HidSmbus_Close()	—	OK

Table 5. Split Thread Access Model (Unsafe)

Thread A	Thread B	Result
HidSmbus_Open()	—	—
—	*QueueInterruptReports()**	—
HidSmbus_Close()	—	Error: Device inaccessible
—	Terminate Thread	OK: Thread relinquishes device access

Table 6. Split Thread Access Mode (Safe)

Thread A	Thread B	Result
HidSmbus_Open()	—	—
—	*QueueInterruptReports()**	—
—	*HidSmbus_CancelIo()*	—
HidSmbus_Close()	—	OK

Table 7. Multi-Thread Access Model (Unsafe)

Thread A	Thread B	Result
HidSmbus_Open()	—	—
QueueInterruptReports()?	*QueueInterruptReports()*?	—
HidSmbus_Close()	—	*QueueInterruptReports()** Thread A: OK *QueueInterruptReports()** Thread B: Error: Device inaccessible
—	Terminate Thread	OK: Thread relinquishes device access

Table 8. Multi-Thread Access Model (Safe)

Thread A	Thread B	Result
HidSmbus_Open()	—	—
QueueInterruptReports()?	*QueueInterruptReports()?*	—
—	*HidSmbus_CancelIo()*	—
HidSmbus_Close()	—	OK

7. Surprise Removal (For Windows)

HidSmbus_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 notices for HID devices. The application must retrieve the device path to filter devices based on VID/PID. Similarly, if a DBT_DEVICEREMOVECOMPLETE message is received, the application must check to see if the device path matches the device path of any connected devices. If this is the case, then the device was removed and the application must close the device. Also, if a DBT_DEVICEARRIVAL message is received, the application might add the new device to a device list so that users can select any HID device 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 # 222649, 311158, and 311153 for programming examples for C++, Visual Basic .NET, and Visual C#.

DOCUMENT CHANGE LIST

Revision 0.1 to Revision 2

- Added Table 1.
- Added support for the Mac OS X dynamic library.
- Removed Appendix.

Contact Information

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Please visit the Silicon Labs Technical Support web page:
https://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.