CP2110/4 HID-TO-UART API SPECIFICATION

1. Introduction

The Silicon Labs HID-to-UART interface library provides a simple API to configure and operate CP2110 and CP2114 devices. The library provides interface abstraction so that users can develop their application without writing any USB HID code. C libraries implementing the CP2110 and CP2114 Interface Specification are provided for Windows 2000 and later, Mac OS X 10.5 and later, and Linux. Similarly, various include files are provided to import library functions into C#.NET and Visual Basic.NET. Refer to Table 1 for complete details.

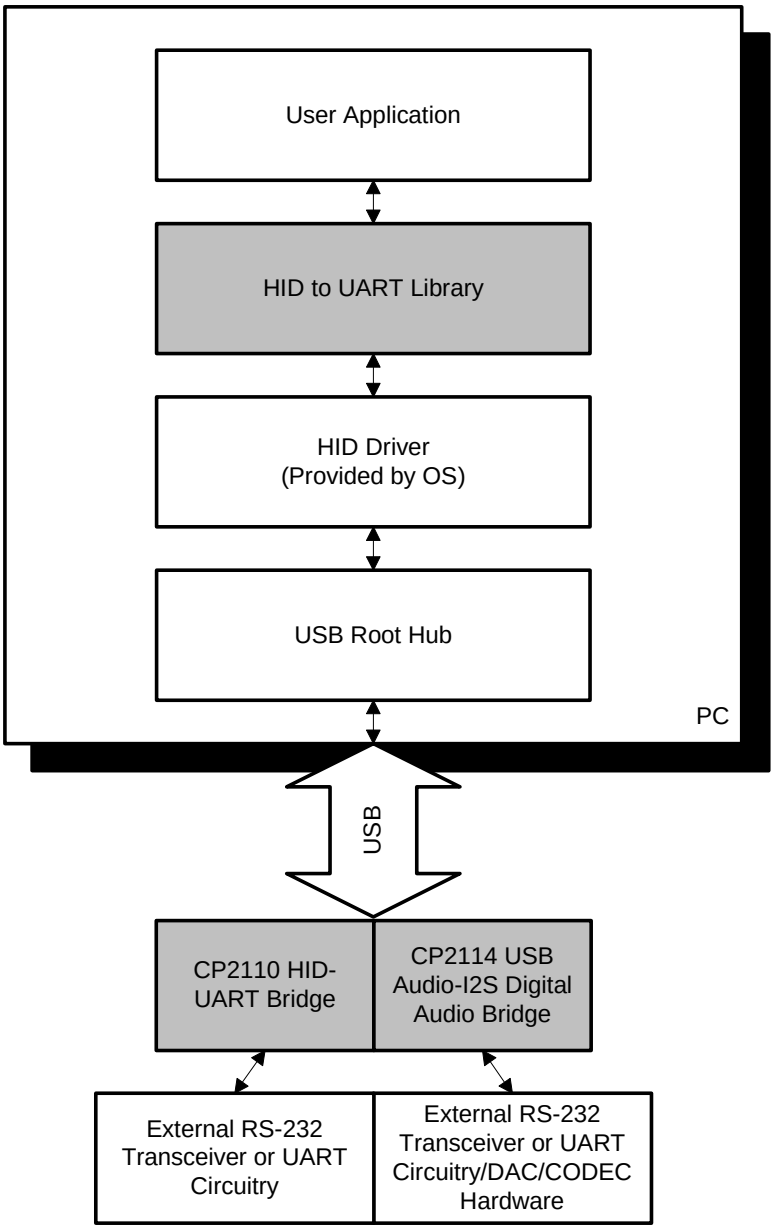

Figure 1. System Architecture Diagram

Table 1. HID-to-UART Include Files

Operating System	Library	Include Files	Version
Windows 2000 and later	SLABHIDtoUART.dll[1]	SLABHIDtoUART.h (C/C++) SLABHIDtoUART.cs (C#.NET) SLABHIDtoUART.vb (VB.NET) SLABCP2110.h (C/C++) SLABCP2110.cs (C#.NET) SLABCP2110.vb (VB.NET) SLABCP2114.h (C/C++) CP2114_common.h (C/C++)	2.0
Mac OS X 10.5 and later	libSLABHIDtoUART.dylib	SLABHIDtoUART.h (C, C++, Obj-C) SLABCP2110.h (C, C++, Obj-C) SLABCP2114.h (C, C++, Obj-C) CP2114_common.h (C, C++, Obj-C) Types.h (C, C++, Obj-C)	2.0
Linux	libslabhidtouart.so[2]	SLABHIDtoUART.h (C/C++) SLABCP2110.h (C/C++) SLABCP2114.h (C/C++) CP2114_common.h (C/C++) Types.h (C/C++)	2.0

Notes:
1. Requires SLABHIDDevice.dll version 1.5 during runtime.
2. Requires libusb-1.0 during runtime.

2. API Functions

The following API functions apply to the CP2110 and CP2114.

Definition	Description	Page #
HidUart_GetNumDevices()	Returns the number of devices connected	4
HidUart_GetString()	Returns a string for a device by index	5
HidUart_GetOpenedString()	Returns a string for a device by device object pointer	6
HidUart_GetIndexedString()	Returns an indexed USB string descriptor by index (Windows/Linux only)	7
HidUart_GetOpenedIndexedString()	Returns an indexed USB string descriptor by device object pointer (Windows/Linux only)	7
HidUart_GetAttributes()	Returns the VID, PID, and release number for a device by index.	8
HidUart_GetOpenedAttributes()	Returns the VID, PID, and release number for a device by device object pointer.	8
HidUart_Open()	Opens a device and returns a device object pointer	9
HidUart_Close()	Cancels pending IO and closes a device	9
HidUart_IsOpened()	Returns the device opened status	9
HidUart_SetUartEnable()	Enables/disables the UART	10
HidUart_GetUartEnable()	Gets UART status	10
HidUart_Read()	Reads a block of data from a device	11
HidUart_Write()	Writes a block of data to a device	12
HidUart_FlushBuffers()	Flushes the TX and RX buffers for a device	13
HidUart_CancelIo()	Cancels pending HID reads and writes (Windows only)	13
HidUart_SetTimeouts()	Sets read and write block timeouts for a device	13
HidUart_GetTimeouts()	Gets read and write block timeouts for a device	14
HidUart_GetUartStatus()	Returns the number of bytes in the device transmit and receive FIFOs and parity/overrun errors	14
HidUart_SetUartConfig()	Sets baud rate, parity, flow control, data bits, and stop bits	15
HidUart_GetUartConfig()	Gets baud rate, parity, flow control, data bits, and stop bits	16
HidUart_StartBreak()	Starts transmission of the line break for the specified duration	17
HidUart_StopBreak()	Stops transmission of the line break	17
HidUart_Reset()	Resets the device with re-enumeration	17
HidUart_ReadLatch()	Gets the port latch value from a device	18

2.1. HidUart_GetNumDevices

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

Prototype: `HID_UART_STATUS HidUart_GetNumDevices (DWORD* numDevices, WORD vid, WORD pid)`

Parameters:
1. *numDevices*—Returns the number of devices connected on return.
2. *vid*—Filter 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*—Filter 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_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_PARAMETER

2.2. HidUart_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 *HidUart_GetNumDevices()* – 1.

Prototype: `HID_UART_STATUS HidUart_GetString (DWORD deviceNum, WORD vid, WORD pid, char* deviceString, DWORD options)`

Parameters:

1. *deviceNum*—Index of the device for which the string is desired.
2. *vid*—Filter 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*—Filter 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*—Variable of type `HID_UART_DEVICE_STRING` 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 and Linux.
5. *options*—Determines if *deviceString* contains a vendor ID string, product ID string, serial string, device path string, manufacturer string, or product string.

Definition	Value	Length	Description
HID_UART_GET_VID_STR	0x01	5	Vendor ID
HID_UART_GET_PID_STR	0x02	5	Product ID
HID_UART_GET_PATH_STR	0x03	260/512	Device path
HID_UART_GET_SERIAL_STR	0x04	256	Serial string
HID_UART_GET_MANUFACTURER_STR	0x05	256	Manufacturer string
HID_UART_GET_PRODUCT_STR	0x06	256	Product string

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_DEVICE_NOT_FOUND
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_ACCESS_ERROR

2.3. HidUart_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_UART_STATUS HidUart_GetOpenedString (HID_UART_DEVICE device, char* deviceString, DWORD options)`

Parameters:

1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *deviceString*—Variable of type HID_UART_DEVICE_STRING 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 and Linux.
3. *options*—Determines if *deviceString* contains a vendor ID string, product ID string, serial string, device path string, manufacturer string, or product string.

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

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_ACCESS_ERROR

2.4. HidUart_GetIndexedString

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

Prototype: `HID_UART_STATUS HidUart_GetIndexedString (DWORD deviceNum, WORD vid, WORD pid, DWORD stringIndex, char* deviceString)`

Parameters:

1. *deviceNum*—Index of the device for which the string is desired.
2. *vid*—Filter 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*—Filter 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*—Variable of type HID_UART_DEVICE_STRING which will contain a NULL terminated device descriptor string on return. The string is 260 bytes on Windows and 512 bytes on Linux.

Return Value: HID_UART_STATUS= HID_UART_SUCCESS
HID_UART_DEVICE_NOT_FOUND
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_ACCESS_ERROR

2.5. HidUart_GetOpenedIndexedString

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

Prototype: `HID_UART_STATUS HidUart_GetOpenedIndexedString (HID_UART_DEVICE device, DWORD stringIndex, char* deviceString)`

Parameters:

1. *deviceNum*—Device object pointer as returned by HidUart_Open().
2. *stringIndex*—Specifies the device-specific index of the USB string descriptor to return.
3. *deviceString* —Variable of type HID_UART_DEVICE_STRING which will contain a NULL terminated device descriptor string on return. The string is 260 bytes on Windows and 512 bytes on Linux.

Return Value: HID_UART_STATUS= HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_ACCESS_ERROR

2.6. HidUart_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_UART_STATUS HidUart_GetAttributes (DWORD deviceNum, WORD vid, WORD pid, WORD* deviceVid, WORD* devicePid, WORD* deviceReleaseNumber)`

Parameters:
1. *deviceNum*—Index of the device for which the string is desired.
2. *vid*—Filter 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*—Filter 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_UART_STATUS= HID_UART_SUCCESS
HID_UART_DEVICE_NOT_FOUND
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_ACCESS_ERROR

2.7. HidUart_GetOpenedAttributes

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

Prototype: `HID_UART_STATUS HidUart_GetOpenedAttributes (HID_UART_DEVICE device, WORD* deviceVid, WORD* devicePid, WORD* deviceReleaseNumber)`

Parameters:
1. *device*—Device object pointer as returned by HidUart_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_UART_STATUS= HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_ACCESS_ERROR

2.8. HidUart_Open

Description: Opens a device using a device number between 0 and *HidUart_GetNumDevices()–1*, enables the UART, and returns a device object pointer which will be used for subsequent accesses.

Prototype: `HID_UART_STATUS HidUart_Open (HID_UART_DEVICE* device, DWORD deviceNum, WORD vid, WORD pid)`

Parameters:

1. *device*—Returns a pointer to a HID-to-UART device object. This pointer will be used by all subsequent accesses to the device.
2. *deviceNum* —Zero-based device index, between 0 and *(HidUart_GetNumDevices()* – 1).
3. *vid*—Filter 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*—Filter 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_UART_STATUS= HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_DEVICE_NOT_FOUND
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_ACCESS_ERROR
HID_UART_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 actually a CP211x, use of this library will cause undesirable results. The best course of action would be to designate a unique VID/PID for CP211x devices only. The application should then filter devices using this VID/PID.

2.9. HidUart_Close

Description: Closes an opened device using the device object pointer provided by *HidUart_Open()*.

Prototype: `HID_UART_STATUS HidUart_Close (HID_UART_DEVICE device)`

Parameters: *device*—Device object pointer as returned by *HidUart_Open()*.

Return Value: HID_UART_STATUS= HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_HANDLE
HID_UART_DEVICE_ACCESS_ERROR

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

2.10. HidUart_IsOpened

Description: Returns the device opened status.

Prototype: `HID_UART_STATUS HidUart_IsOpened (HID_UART_DEVICE device, BOOL* opened)`

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

Return Value: HID_UART_STATUS= HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER

2.11. HidUart_SetUartEnable

Description: Enables or disables the UART.

Prototype: `HID_UART_STATUS HidUart_SetUartEnable (HID_UART_DEVICE, BOOL enable)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *enable*—Set to *TRUE* to enable the UART. Set to *FALSE* to disable the UART.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_DEVICE_IO_FAILED

Remarks: Enabling or disabling the UART will flush the UART FIFOs if the *flushBuffers* parameter is enabled by calling *HidUart_SetUsbConfig()*.

2.12. HidUart_GetUartEnable

Description: Returns the UART enable status.

Prototype: `HID_UART_STATUS HidUart_GetUartEnable (HID_UART_DEVICE, BOOL* enable)`

Parameters:
1. *device* —Device object pointer as returned by *HidUart_Open()*.
2. *enable* —Returns *TRUE* if the UART is enabled. Returns *FALSE* if the UART is disabled.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_DEVICE_IO_FAILED
HID_UART_INVALID_PARAMETER

2.13. HidUart_Read

Description: Reads the available number of bytes into the supplied buffer and returns the number of bytes read which can be less than the number of bytes requested. This function returns synchronously after reading the requested number of bytes or after the timeout duration has elapsed. Read and write timeouts can be set using *HidUart_SetTimeouts()* described in "2.17. HidUart_SetTimeouts" on page 13.

Prototype:
```
HID_UART_STATUS HidUart_Read (HID_UART_DEVICE device, BYTE* buffer,
DWORD numBytesToRead, DWORD* numBytesRead)
```

Parameters:

1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *buffer*—Address of a buffer to be filled with read data.
3. *numBytesToRead*—Number of bytes to read from the device into the buffer (1–32768). This value must be less than or equal to the size of *buffer.*
4. *numBytesRead*—Returns the number of bytes actually read into the buffer on completion.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_READ_ERROR
HID_UART_INVALID_PARAMETER
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_READ_TIMED_OUT
HID_UART_INVALID_REQUEST_LENGTH

Remarks: *HidUart_Read()* returns HID_UART_READ_TIMED_OUT if the number of bytes read is less than the number of bytes requested. This will only occur after the read timeout has elapsed. If the number of bytes read matches the number of bytes requested, this function will return HID_UART_SUCCESS.

2.14. HidUart_Write

Description: Write the specified number of bytes from the supplied buffer to the device. This function returns synchronously after writing the requested number of bytes or after the timeout duration has elapsed. Read and write timeouts can be set using *HidUart_SetTimeouts()* described in "2.17. HidUart_SetTimeouts" on page 13.

Prototype:
```
HID_UART_STATUS HidUart_Write (HID_UART_DEVICE device, BYTE* buffer,
DWORD numBytesToWrite, DWORD* numBytesWritten)
```

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *buffer*—Address of a buffer to be sent to the device.
3. *numBytesToWrite*—Number of bytes to write to the device (1–4096 bytes). *This value must be less than or equal to the size of buffer.*
4. *numBytesWritten*—Returns the number of bytes actually written to the device.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_WRITE_ERROR
HID_UART_INVALID_PARAMETER
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_WRITE_TIMED_OUT
HID_UART_INVALID_REQUEST_LENGTH

Remarks: *HidUart_Write()* returns HID_UART_WRITE_TIMED_OUT if the number of bytes written is less than the number of bytes requested. Data is broken down into HID interrupt reports between 1 – 63 bytes in size and transmitted. Each report will be given a specific amount of time to complete. This report timeout is determined by *writeTimeout* in *HidUart_SetTimeouts()*. Each interrupt report is given the max timeout to complete because a timeout at the interrupt report level is considered an unrecoverable error (the IO is canceled in an unknown state). If the HID set interrupt report times out, *HidUart_Write()* returns HID_UART_WRITE_ERROR. The *HidUart_Write()* timeout may take up to twice as long as the timeout specified to allow each interrupt report to complete.

2.15. HidUart_FlushBuffers

Description: This function flushes the receive buffer in the device and the HID driver and/or the transmit buffer in the device.

Prototype: `HID_UART_STATUS HidUart_FlushBuffers (HID_UART_DEVICE device, BOOL flushTransmit, BOOL flushReceive)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *flushTransmit* —Set to *TRUE* to flush the device transmit buffer.
3. *flushReceive* —Set to *TRUE* to flush the device receive buffer and HID receive buffer.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_DEVICE_IO_FAILED

2.16. HidUart_Cancello

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

Prototype: `HID_UART_STATUS HidUart_CancelIo (HID_UART_DEVICE device)`

Parameters: *device*—Device object pointer as returned by *HidUart_Open()*.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_DEVICE_IO_FAILED

2.17. HidUart_SetTimeouts

Description: Sets the read and write timeouts. Timeouts are used for *HidUart_Read()* and *HidUart_Write()*. The default value for timeouts is 1000 ms, but timeouts can be set to wait for any number of milliseconds between 0 and 0xFFFFFFFF.

Prototype: `HID_UART_STATUS HidUart_SetTimeouts (HID_UART_DEVICE device, DWORD readTimeout, DWORD writeTimeout)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *readTimeout—HidUart_Read()* operation timeout in milliseconds.
3. *writeTimeout—HidUart_Write()* operation timeout in milliseconds.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT

Remarks: If read timeouts are set to a large value and no data is received, then the application may appear unresponsive. It is recommended to set timeouts appropriately before using the device.

2.18. HidUart_GetTimeouts

Description: Returns the current read and write timeouts specified in milliseconds.

Prototype: `HID_UART_STATUS HidUart_GetTimeouts (HID_UART_DEVICE device, DWORD* readTimeout, DWORD* writeTimeout)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *readTimeout—HidUart_Read()* operation timeout in milliseconds.
3. *writeTimeout—HidUart_Write()* operation timeout in milliseconds.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_PARAMETER
HID_UART_INVALID_DEVICE_OBJECT

Remarks: Read and write timeouts are maintained for each device but are not persistent across *HidUart_Open()*/*HidUart_Close()*.

2.19. HidUart_GetUartStatus

Description: Returns the number of bytes held in the device receive and transmit FIFO. Returns the parity/error status and line break status.

Prototype: `HID_UART_STATUS HidUart_GetUartStatus (HID_UART_DEVICE device, WORD* transmitFifoSize, WORD* receiveFifoSize, BYTE* errorStatus, BYTE* lineBreakStatus)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *transmitFifoSize*—Returns the number of bytes currently held in the device transmit FIFO.
3. *receiveFifoSize*—Returns the number of bytes currently held in the device receive FIFO.
4. *errorStatus*—Returns an error status bitmap describing parity and overrun errors. Calling this function clears the errors.

Definition	Value	Description
HID_UART_MASK_PARITY_ERROR	0x01	Parity error
HID_UART_MASK_OVERRUN_ERROR	0x02	Overrun error

5. *lineBreakStatus*—Returns 0x01 if line break is currently active and 0x00 otherwise.

Definition	Value	Description
HID_UART_LINE_BREAK_INACTIVE	0x00	Line break inactive
HID_UART_LINE_BREAK_ACTIVE	0x01	Line break active

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_PARAMETER
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_DEVICE_IO_FAILED

Remarks: The *transmitFifoSize* and *receiveFifoSize* only apply to data held in the device FIFOs; they do not include data queued in the HID driver or interface library.

2.20. HidUart_SetUartConfig

Description: Sets the baud rate, data bits, parity, stop bits, and flow control. Refer to the device data sheet for a list of supported configuration settings.

Prototype: `HID_UART_STATUS HidUart_SetUartConfig (HID_UART_DEVICE device, DWORD baudRate, BYTE dataBits, BYTE parity, BYTE stopBits, BYTE flowControl)`

Parameters:

1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *baudRate* —The baud rate for UART communication. Default: 115200
3. *dataBits*—The number of data bits for UART communication. Default: 8

Definition	Value	Description
HID_UART_FIVE_DATA_BITS	0x00	5 data bits
HID_UART_SIX_DATA_BITS	0x01	6 data bits
HID_UART_SEVEN_DATA_BITS	0x02	7 data bits
HID_UART_EIGHT_DATA_BITS	0x03	8 data bits

4. *parity*—The parity for UART communication. Default: No parity

Definition	Value	Description
HID_UART_NO_PARITY	0x00	No parity
HID_UART_ODD_PARITY	0x01	Odd parity (sum of data bits is odd)
HID_UART_EVEN_PARITY	0x02	Even parity (sum of data bits is even)
HID_UART_MARK_PARITY	0x03	Mark parity (always 1)
HID_UART_SPACE_PARITY	0x04	Space parity (always 0)

5. *stopBits*—The number of stop bits for UART communication. Default: 1

Definition	Value	Description
HID_UART_SHORT_STOP_BIT	0x00	1 stop bit
HID_UART_LONG_STOP_BIT	0x01	5 data bits: 1.5 stop bits 6-8 data bits: 2 stop bits

6. *flowControl*—The type of flow control for UART communication. Default: No flow control

Definition	Value	Description
HID_UART_NO_FLOW_CONTROL	0x00	No flow control
HID_UART_RTS_CTS_FLOW_CONTROL	0x01	RTS/CTS hardware flow control

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_PARAMETER
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_DEVICE_IO_FAILED

2.21. HidUart_GetUartConfig

Description: Gets the baud rate, data bits, parity, stop bits, and flow control. Refer to the device data sheet for a list of supported baud rates. See "2.20. HidUart_SetUartConfig" on page 15.

Prototype:
```
HID_UART_STATUS HidUart_GetUartConfig (HID_UART_DEVICE device, DWORD* baudRate, BYTE* dataBits, BYTE* parity, BYTE* stopBits, BYTE* flowControl)
```

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *baudRate*—Returns the baud rate for UART communication.
3. *dataBits*—Returns the number of data bits for UART communication.
4. *parity*—Returns the parity for UART communication.
5. *stopBits*—Returns the number of stop bits for UART communication.
6. *flowControl*—Returns the type of flow control for UART communication.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_PARAMETER
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_DEVICE_IO_FAILED

2.22. HidUart_StartBreak

Description: Causes the device to transmit a line break, holding the TX pin low, for the specified duration in milliseconds.

Prototype: `HID_UART_STATUS HidUart_StartBreak (HID_UART_DEVICE device, BYTE duration)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *duration*—The length of time in milliseconds to transmit the line break (1–125 ms). A value of 0x00 indicates that the line break will be transmitted indefinitely until *HidUart_StopBreak()* is called.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_PARAMETER
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

2.23. HidUart_StopBreak

Description: Stops the device from transmitting a line break.

Prototype: `HID_UART_STATUS HidUart_StopBreak (HID_UART_DEVICE device)`

Parameters: *device*—Device object pointer as returned by *HidUart_Open()*.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

Remarks: This function is ignored if the device is not transmitting a line break.

2.24. HidUart_Reset

Description: Initiates a full device reset. Transmit and receive FIFOs will be cleared, UART settings will be reset to default values, and the device will re-enumerate.

Prototype: `HID_UART_STATUS HidUart_Reset (HID_UART_DEVICE device)`

Parameters: *device*—Device object pointer as returned by *HidUart_Open()*.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_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. Default UART settings are as follows: 115200 8N1, no flow control.

2.25. HidUart_ReadLatch

Description: Gets the current port latch value from the device.

Prototype: `HID_UART_STATUS HidUart_ReadLatch (HID_UART_DEVICE device, WORD* latchValue)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *latchValue*—Returns the port latch value (Logic High = 1, Logic Low = 0). If a pin is configured as a GPIO input or flow control pin that is an input, then the corresponding bit represents the input value. If a pin is configured as a GPIO output pin or a flow control pin that is an output, then the corresponding bit represents the logic level driven on the pin.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

Remarks: See "6. Port Latch Pin Definition" on page 46 for more information on configuring GPIO and flow control pins. Bits 9 and 15 of *latchValue* are ignored.

2.26. HidUart_WriteLatch

Description: Sets the current port latch value to the device.

Prototype: `HID_UART_STATUS HidUart_WriteLatch (HID_UART_DEVICE device, WORD latchValue, WORD latchMask)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *latchValue*—Value to write to the port latch (Logic High = 1, Logic Low = 0). This function is used to set the values for GPIO pins or flow control pins that are configured as outputs. This function will not affect any pins that are not configured as outputs.
3. *latchMask*—Determines which pins to change (Change = 1, Leave = 0).

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

Remarks: See "6. Port Latch Pin Definition" on page 46 for more information on configuring GPIO and flow control pins. Bits 9 and 15 or *latchValue* and *latchMask* are ignored. Pins TX, RX, Suspend, and / Suspend cannot be written to using this function.

2.27. HidUart_GetPartNumber

Description: Retrieves the part number and version of the CP211x device.

Prototype: `HID_UART_STATUS HidUart_GetPartNumber (HID_UART_DEVICE device, BYTE* partNumber, BYTE* version)`

Parameters: 1. *device*—Device object pointer as returned by *HidUart_Open()*.

2. *partNumber*—Returns the device part number.

Definition	Value	Description
HID_UART_PART_CP2110	0x0A	CP2110
HID_UART_PART_CP2114	0x0E	CP2114

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

Return Value: HID_UART_STATUS=HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED

2.28. HidUart_GetLibraryVersion

Description: Returns the HID-to-UART Interface Library version.

Prototype: `HID_UART_STATUS HidUart_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_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_PARAMETER

2.29. HidUart_GetHidLibraryVersion

Description: Returns the version of the HID Device Interface Library that is currently in use.

Prototype: `HID_UART_STATUS HidUart_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_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_PARAMETER

2.30. HidUart_GetHidGuid

Description: Obtains the HID GUID. This can be used to register for surprise removal notifications. (Windows only)

Prototype: `HID_UART_STATUS HidUart_GetHidGuid (void* guid)`

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

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_PARAMETER

3. User Customization API Functions

The following parameters are programmable on the device. Different functions are provided to program these parameters. Each of these functions can only be called once for each device and apply to the CP2110 and CP2114.

Name	Size (Bytes)	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
Release Version	2	Major and Minor release version
Flush Buffers	1	Purge FIFOs on enable/disable
Manufacturer String	126	Product Manufacturer (English Unicode)
Product Description String	126	Product Description (English Unicode)
Serial String	62	Serialization String (English Unicode)

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

Definition	Description	Page #
HidUart_SetLock()	Prevents further OTP programming/customization	22
HidUart_GetLock()	Gets the OTP lock status	23
HidUart_SetUsbConfig()	Sets VID, PID, power, power mode, release version, and flush buffers settings	24
HidUart_GetUsbConfig()	Gets VID, PID, power, power mode, release version, and flush buffers settings	26
HidUart_SetManufacturingString()	Sets the USB manufacturing string	27
HidUart_GetManufacturingString()	Gets the USB manufacturing string	27
HidUart_SetProductString()	Sets the USB product string	28
HidUart_GetProductString()	Gets the USB product string	28
HidUart_SetSerialString()	Sets the USB serial string	29
HidUart_GetSerialString()	Gets the USB serial string	29
Note: "4. CP2110 User-Customizable API Functions" on page 30 and "5. CP2114 User-Customizable API Functions" on page 35 contain additional user customization/one-time programmable functions specific to the CP2110 and CP2114.		

3.1. HidUart_SetLock

Description: Permanently locks/disables device customization.

Prototype: `HID_UART_STATUS HidUart_SetLock (HID_UART_DEVICE device, WORD lock)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *lock*—Bitmask specifying which fields can be customized/programmed and which fields are already customized.

Bit	Definition	Mask	Description
0	HID_UART_LOCK_PRODUCT_STR_1	0x0001	Product String
1	HID_UART_LOCK_PRODUCT_STR_2	0x0002	Product String
2	HID_UART_LOCK_SERIAL_STR	0x0004	Serial String
3	HID_UART_LOCK_PIN_CONFIG	0x0008	Pin Config
4	N/A		
5	N/A		
6	N/A		
7	N/A		
8	HID_UART_LOCK_VID	0x0100	VID
9	HID_UART_LOCK_PID	0x0200	PID
10	HID_UART_LOCK_POWER	0x0400	Power
11	HID_UART_LOCK_POWER_MODE	0x0800	Power Mode
12	HID_UART_LOCK_RELEASE_VERSION	0x1000	Release Version
13	HID_UART_LOCK_FLUSH_BUFFERS	0x2000	Flush Buffers
14	HID_UART_LOCK_MFG_STR_1	0x4000	Manufacturing String
15	HID_UART_LOCK_MFG_STR_2	0x8000	Manufacturing String

Definition	Bit Value	Description
HID_UART_LOCK_UNLOCKED	1	Field can be customized
HID_UART_LOCK_LOCKED	0	Field has already been customized or has been locked

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_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_UART_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. HidUart_GetLock

Description: Returns the device customization lock status.

Prototype: `HID_UART_STATUS HidUart_GetLock (HID_UART_DEVICE device, WORD* lock)`

Parameters: 1. *device*—Device object pointer as returned by *HidUart_Open()*.

2. *lock*—Returns a bitmask specifying which fields are locked.

Bit	Definition	Mask	Description
0	HID_UART_LOCK_PRODUCT_STR_1	0x0001	Product String
1	HID_UART_LOCK_PRODUCT_STR_2	0x0002	Product String
2	HID_UART_LOCK_SERIAL_STR	0x0004	Serial String
3	HID_UART_LOCK_PIN_CONFIG	0x0008	Pin Config
4	N/A		
5	N/A		
6	N/A		
7	N/A		
8	HID_UART_LOCK_VID	0x0100	VID
9	HID_UART_LOCK_PID	0x0200	PID
10	HID_UART_LOCK_POWER	0x0400	Power
11	HID_UART_LOCK_POWER_MODE	0x0800	Power Mode
12	HID_UART_LOCK_RELEASE_VERSION	0x1000	Release Version
13	HID_UART_LOCK_FLUSH_BUFFERS	0x2000	Flush Buffers
14	HID_UART_LOCK_MFG_STR_1	0x4000	Manufacturing String
15	HID_UART_LOCK_MFG_STR_2	0x8000	Manufacturing String

Definition	Bit Value	Description
HID_UART_LOCK_UNLOCKED	1	Field can be customized
HID_UART_LOCK_LOCKED	0	Field has already been customized or has been locked

Return Value:HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED

3.3. HidUart_SetUsbConfig

Description: Allows one-time customization of the USB configuration, which includes vendor ID, product ID, power, power mode, release version, and flush buffers setting. Each field can be independently programmed one time each via the mask field.

Prototype: `HID_UART_STATUS HidUart_SetUsbConfig (HID_UART_DEVICE device, WORD vid, WORD pid, BYTE power, BYTE powerMode, WORD releaseVersion, BYTE flushBuffers, BYTE mask)`

Parameters:

1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *vid*—Vendor ID.
3. *pid*—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_UART_BUS_POWER	0x00	Device is bus powered
HID_UART_SELF_POWER_VREG_DIS	0x01	Device is self-powered and voltage regulator is disabled
HID_UART_SELF_POWER_VREG_EN	0x02	Device is self-powered and voltage regulator is enabled

6. *releaseVersion*—The 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.
7. *flushBuffers*—Bitmask specifying whether the RX and/or TX FIFOs are purged upon a device open and/or close.

Bit	Definition	Mask	Description
0	HID_UART_FLUSH_TX_OPEN	0x01	Flush TX on Open
1	HID_UART_FLUSH_TX_CLOSE	0x02	Flush TX on Close
2	HID_UART_FLUSH_RX_OPEN	0x04	Flush RX on Open
3	HID_UART_FLUSH_RX_CLOSE	0x08	Flush RX on Close

8. *mask*—Bitmask specifying which fields to customize.

Bit	Definition	Mask	Description
0	HID_UART_SET_VID	0x01	VID
1	HID_UART_SET_PID	0x02	PID
2	HID_UART_SET_POWER	0x04	Power
3	HID_UART_SET_POWER_MODE	0x08	Power Mode
4	HID_UART_SET_RELEASE_VERSION	0x10	Release Version
5	HID_UART_SET_FLUSH_BUFFERS	0x20	Flush Buffers
6	N/A		
7	N/A		

Definition	Bit Value	Description
HID_UART_SET_IGNORE	0	Field will be unchanged
HID_UART_SET_PROGRAM	1	Field will be programmed

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED

3.4. HidUart_GetUsbConfig

Description: Retrieves USB configuration, which includes vendor ID, product ID, power, power mode, release version, and flush buffers setting.

Prototype: `HID_UART_STATUS HidUart_GetUsbConfig (HID_UART_DEVICE device, WORD* vid, WORD* pid, BYTE* power, BYTE* powerMode, WORD* releaseVersion, BYTE* flushBuffers)`

Parameters:

1. *device*—Device object pointer as returned by *HidUart_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_UART_BUS_POWER	0x00	Device is bus-powered
HID_UART_SELF_POWER_VREG_DIS	0x01	Device is self-powered and voltage regulator is disabled
HID_UART_SELF_POWER_VREG_EN	0x02	Device is self-powered and voltage regulator is enabled

6. *releaseVersion*—Returns the 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.
7. *flushBuffers*—Returns a bitmask specifying whether the RX and/or TX FIFOs are purged upon a device open and/or close.

Bit	Definition	Mask	Description
0	HID_UART_FLUSH_TX_OPEN	0x01	Flush TX on Open
1	HID_UART_FLUSH_TX_CLOSE	0x02	Flush TX on Close
2	HID_UART_FLUSH_RX_OPEN	0x04	Flush RX on Open
3	HID_UART_FLUSH_RX_CLOSE	0x08	Flush RX on Close

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED

3.5. HidUart_SetManufacturingString

Description: Allows one-time customization of the USB manufacturing string.

Prototype: `HID_UART_STATUS HidUart_SetManufacturingString (HID_UART_DEVICE device, char* manufacturingString, BYTE strlen)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *manufacturingString*—Variable of type `HID_UART_CP2110/4_MFG_STR`, a 62-byte character buffer containing the ASCII manufacturing string.
3. *strlen*—The length of *manufacturingString* in bytes. The maximum string length is 62 bytes.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED

3.6. HidUart_GetManufacturingString

Description: Retrieves the USB manufacturing string.

Prototype: `HID_UART_STATUS HidUart_GetManufacturingString (HID_UART_DEVICE device, char* manufacturingString, BYTE* strlen)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *manufacturingString*—Variable of type HID_UART_CP2110/4_MFG_STR, a 62-byte character buffer that will contain the ASCII manufacturing string.
3. *strlen*—Returns the length of the string in bytes.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED

3.7. HidUart_SetProductString

Description: Allows one-time customization of the USB product string.

Prototype: `HID_UART_STATUS HidUart_SetProductString (HID_UART_DEVICE device, char* productString, BYTE strlen)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *productString*—Variable of type `HID_UART_CP2110/4_PRODUCT_STR`, a 62-byte character buffer containing the ASCII product string.
3. *strlen*—The length of *productString* in bytes. The maximum string length is 62 bytes.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED

3.8. HidUart_GetProductString

Description: Retrieves the USB product string.

Prototype: `HID_UART_STATUS HidUart_GetProductString (HID_UART_DEVICE device, char* productString, BYTE* strlen)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *productString*—Variable of type HID_UART_CP2110/4_PRODUCT_STR, a 62-byte character buffer that will contain the ASCII product string.
3. *strlen*—Returns the length of the string in bytes.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED

3.9. HidUart_SetSerialString

Description: Allows one-time customization of the USB serial string.

Prototype: `HID_UART_STATUS HidUart_SetSerialString (HID_UART_DEVICE device, char* serialString, BYTE strlen)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *serialString*—Variable of type HID_UART_CP2110/4_SERIAL_STR, a 30-byte character buffer containing the ASCII serial string.
3. *strlen*—The length of *serialString* in bytes. The maximum string length is 30 bytes.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED

3.10. HidUart_GetSerialString

Description: Retrieves the USB product string.

Prototype: `HID_UART_STATUS HidUart_GetSerialString (HID_UART_DEVICE device, char* serialString, BYTE* strlen)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *serialString*—Variable of type HID_UART_CP2110/4_SERIAL_STR, a 30-byte character buffer that will contain the Unicode product string.
3. *strlen*—Returns the length of the string in bytes.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED

4. CP2110 User-Customizable API Functions

The following parameters are programmable on the CP2110. Different functions are provided to program these parameters. Each of these functions can only be called once for each device.

Name	Size	Short Description
Pin Configuration	18	All pins configuration

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

Definition	Description	Page #
HidUart_SetPinConfig()	Configures the pin behavior	30
HidUart_GetPinConfig()	Gets pin configuration	34

4.1. HidUart_SetPinConfig

Description: Allows one-time configuration of the GPIO mode for each pin.

Prototype:
```
HID_UART_STATUS HidUart_SetPinConfig(HID_UART_DEVICE device, BYTE* pinConfig, BOOL useSuspendValues, WORD suspendValue, WORD suspendMode, BYTE rs485Level, BYTE clkDiv);
```

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *pinConfig*—A pointer to a 13-byte array that configures the GPIO mode for each of the 13 pins. The RX pin is not configurable.

Byte	Name	Value	Mode
0	GPIO.0/CLK	0x00 0x01 0x02 0x03	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull CLK Output–Push Pull
1	GPIO.1/RTS	0x00 0x01 0x02 0x03	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull RTS Output–Push Pull
2	GPIO.2/CTS	0x00 0x01 0x02 0x03	GPIO Input GPIO Output– Open Drain GPIO Output– Push Pull CTS Input
3	GPIO.3/RS485	0x00 0x01 0x02 0x03	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull RS485 Output–Push Pull

Byte	Name	Value	Mode
4	GPIO.4/TX Toggle	0x00 0x01 0x02 0x03	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull TX Toggle Output–Push Pull
5	GPIO.5/RX Toggle	0x00 0x01 0x02 0x03	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull RX Toggle Output–Push Pull
6	GPIO.6	0x00 0x01 0x02	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull
7	GPIO.7	0x00 0x01 0x02	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull
8	GPIO.8	0x00 0x01 0x02	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull
9	GPIO.9	0x00 0x01 0x02	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull
10	TX	0x01 0x02	TX–Open Drain TX–Push Pull
11	Suspend	0x01 0x02	Suspend–Open Drain Suspend–Push Pull
12	/Suspend	0x01 0x02	/Suspend–Open Drain /Suspend–Push Pull

Definition	Value	Description
HID_UART_GPIO_MODE_INPUT	0x00	GPIO Input
HID_UART_GPIO_MODE_OD	0x01	GPIO Output–Open Drain
HID_UART_GPIO_MODE_PP	0x02	GPIO Output–Push Pull
HID_UART_GPIO_MODE_FUNCTION1	0x03	Pin specific function and mode

3. *useSuspendValues*—Specifies if the device is to use suspendValue and suspendMode when device is in USB suspend. If set to 1, the device will use these values. If cleared to 0, the device's GPIO pins will remain in the state they were in before entering USB suspend.
4. *suspendValue*–This is the latch value that will be driven on each GPIO pin when the device is in a suspend state.

Bit	Definition	Bit Mask	Description
0	CP2110_MASK_GPIO_0_CLK	0x0001	GPIO.0/CLK
1	CP2110_ MASK_GPIO_1_RTS	0x0002	GPIO.1/RTS
2	CP2110_ MASK_GPIO_2_CTS	0x0004	GPIO.2/CTS
3	CP2110_ MASK_GPIO_3_RS485	0x0008	GPIO.3/RS485
4	CP2110_ MASK_TX	0x0010	TX
5	CP2110_ MASK_RX	0x0020	RX
6	CP2110_ MASK_GPIO_4_TX_TOGGLE	0x0040	TX Toggle
7	CP2110_ MASK_GPIO_5_RX_TOGGLE	0x0080	RX Toggle
8	CP2110_MASK_SUSPEND_BAR	0x0100	$\overline{\text{Suspend}}$
9	N/A		
10	CP2110_ MASK_GPIO_6	0x0400	GPIO.6
11	CP2110_ MASK_GPIO_7	0x0800	GPIO.7
12	CP2110_ MASK_GPIO_8	0x1000	GPIO.8
13	CP2110_ MASK_GPIO_9	0x2000	GPIO.9
14	CP2110_MASK_SUSPEND	0x4000	Suspend
15	N/A		

Definition	Bit Value	Description
HID_UART_VALUE_SUSPEND_LO	0	Latch = 0 in suspend
HID_UART_VALUE_SUSPEND_HI	1	Latch = 1 in suspend

5. *suspendMode*—Specifies the mode for each GPIO pin when the device is in a suspend state.

Bit	Definition	Bit Mask	Description
0	CP2110_MASK_GPIO_0_CLK	0x0001	GPIO.0/SYSCLK
1	CP2110_ MASK_GPIO_1_RTS	0x0002	GPIO.1/RTS
2	CP2110_ MASK_GPIO_2_CTS	0x0004	GPIO.2/CTS
3	CP2110_ MASK_GPIO_3_RS485	0x0008	GPIO.3/RS485
4	CP2110_ MASK_TX	0x0010	TX
5	CP2110_ MASK_RX	0x0020	RX
6	CP2110_ MASK_GPIO_4_TX_TOGGLE	0x0040	TX Toggle
7	CP2110_ MASK_GPIO_5_RX_TOGGLE	0x0080	RX Toggle
8	CP2110_MASK_SUSPEND_BAR	0x0100	$\overline{\text{Suspend}}$
9	N/A		
10	CP2110_ MASK_GPIO_6	0x0400	GPIO.6
11	CP2110_ MASK_GPIO_7	0x0800	GPIO.7
12	CP2110_ MASK_GPIO_8	0x1000	GPIO.8
13	CP2110_ MASK_GPIO_9	0x2000	GPIO.9
14	CP2110_MASK_SUSPEND	0x4000	Suspend
15	N/A		

Definition	Bit Value	Description
HID_UART_MODE_SUSPEND_OD	0	Open Drain in suspend
HID_UART_MODE_SUSPEND_PP	1	Push Pull in suspend

6. *rs485Level*—Specifies the RS-485 pin level of GPIO.2 when configured in RS-485 mode.

Definition	Value	Description
HID_UART_MODE_RS485_ACTIVE_LO	0x00	GPIO.2/RS485 pin is active low
HID_UART_MODE_RS485_ACTIVE_HI	0x01	GPIO.2/RS485 pin is active high

7. *clkDiv*—Divider applied to GPIO0_CLK clock output. When 0, the output frequency is 24 MHz. For 1–255, the output frequency is 24 MHz/(2 x *clkDiv*).

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

4.2. HidUart_GetPinConfig

Description: Retrieves the GPIO mode configuration for each pin.

Prototype:
```
HID_UART_STATUS HidUart_GetPinConfig(HID_UART_DEVICE device, BYTE* pinConfig, BOOL* useSuspendValues, WORD* suspendValue, WORD* suspendMode, BYTE* rs485Level, BYTE* clkDiv);
```

Parameters:

1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *pinConfig*—A pointer to a 13-byte array that will contain the GPIO mode configuration for each of the 13 pins.
3. *useSuspendValues*—Returns the configuration for using the values in suspendValue and suspendMode when in suspend mode. This bit is the same as bit 15 of suspendMode.
4. *suspendValue*—Returns the latch value that will be driven on each GPIO pin when the device is in a suspend state.
5. *suspendMode*—Returns the mode for each GPIO pin when the device is in a suspend state.
6. *rs485Level*—Returns the RS-485 pin level of GPIO.2 when configured in RS–485 mode.
7. *clkDiv*—Divider applied to GPIO0_CLK clock output. When 0, the output frequency is 24 MHz. For 1–255, the output frequency is 24 MHz/(2 x *clkDiv*).

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5. CP2114 User-Customizable API Functions

The following API functions access customizable features of CP2114 devices.

Definition	Description	Page
CP2114_GetVersions()	Gets the API and firmware versions	36
CP2114_SetPinConfig()	Configures the pin behavior	36
CP2114_GetPinConfig()	Gets pin configuration	39
CP2114_GetDeviceStatus()	Gets the CP2114 device status	40
CP2114_GetDeviceCaps()	Gets the CP2114 device capabilities	41
CP2114_SetRamConfig()	Sets the CP2114 configuration in RAM	42
CP2114_GetRamConfig()	Gets the CP2114 device configuration from RAM	42
CP2114_SetDacRegisters()	Sets the DAC configuration registers	43
CP2114_GetDacRegisters()	Gets the DAC configuration registers	43
CP2114_GetOtpConfig()	Gets the OTP configuration based on the current index	44
CP2114_CreateOtpConfig()	Creates a new configuration block for the CP2114	44
CP2114_SetBootConfig()	Sets the CP2114 boot configuration index	45
CP2114_ReadOTP()	Reads OTP customization block	45
CP2114_WriteOTP()	Writes OTP customization block	45

5.1. CP2114_GetVersions

Description: Returns the CP2114 API and firmware versions.

Prototype: `HID_UART_STATUS CP2114_GetVersions(HID_UART_DEVICE device, BYTE* api_version, BYTE* fw_version);`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *api_version*—Returns the API version of the device.
3. *fw_version*—Returns the firmware version of the device.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5.2. CP2114_SetPinConfig

Description: Allows one-time configuration of the GPIO mode for each pin.

Prototype: `HID_UART_STATUS CP2114_SetPinConfig(HID_UART_DEVICE device, BYTE* pinConfig, BOOL useSuspendValues, WORD suspendValue, WORD suspendMode, BYTE clkDiv)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *pinConfig*—A pointer to a 14-byte array that configures the GPIO mode or dedicated function for each of the 14 pins.

Byte	Name	Value	Mode
0	GPIO.0_RMUTE	0x00 0x01 0x02 0x03	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull Record Mute Input*
1	GPIO.1_PMUTE	0x00 0x01 0x02 0x03	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull Play Back Mute Input*
2	GPIO.2_VOL–	0x00 0x01 0x02 0x03	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull Volume Down Input*
3	GPIO.3_VOL+	0x00 0x01 0x02 0x03	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull Volume Up Input*

4	GPIO.4_RMUTELED	0x00 0x01 0x02 0x03	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull LED1 RMute Output*
5	GPIO.5_TXT_DACSEL0	0x00 0x01 0x02 0x03 0x04	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull TX Toggle Output–Push Pull DAC Select 0 Input*
6	GPIO.6_RXT_DACSEL1	0x00 0x01 0x02 0x03 0x04	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull RX Toggle Output–Push Pull DAC Select 1 Input*
7	GPIO.7_RTS_DACSEL2	0x00 0x01 0x02 0x03 0x04	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull RTS Output–Push Pull DAC Select 2 Input*
8	GPIO.8_CTS_DACSEL3	0x00 0x01 0x02 0x03 0x04	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull CTS Input DAC Select 3 Input*
9	GPIO.9_CLKOUT	0x00 0x01 0x02 0x03	GPIO Input GPIO Output–Open Drain GPIO Output–Push Pull CLK Output–Push Pull*
10	GPIO.10_TX	0x00 0x01 0x02 0x03 0x04	GPIO Input GPIO Output–Open Drain GPIO Output– Open Drain TX Output–Open Drain TX Output–Push Pull*
11	GPIO.11_RX	0x00 0x01 0x02 0x03	GPIO Input GPIO Output–Open Drain GPIO Output– Open Drain RX Input*
12	Suspend	0x01 0x02	Suspend–Open Drain Suspend–Push Pull*
13	/Suspend	0x01 0x02	/Suspend–Open Drain /Suspend–Push Pull*
***Note:** Default setting.			

useSuspendValues—Specifies if the device is to use suspendValue and suspendMode when device is in USB suspend. If set to 1, the device will use these values. If cleared to 0, the device's GPIO pins will remain in the state they were in before entering USB suspend.

suspendValue—This is the latch value that will be driven on each GPIO pin except Suspend and /Suspend when the device is in a suspend state.

suspendMode—Specifies the mode for each GPIO pin when the device is in a suspend state.

Bit	Definition	Bit Mask	Description
0	CP2114_MASK_GPIO_0	0x0001	GPIO.0_RMUTE
1	CP2114_MASK_GPIO_1	0x0002	GPIO.1_PMUTE
2	CP2114_MASK_GPIO_2	0x0004	GPIO.2_VOL-
3	CP2114_MASK_GPIO_3	0x0008	GPIO.3_VOL+
4	CP2114_MASK_GPIO_4	0x0010	GPIO.4_RMUTELED
5	CP2114_MASK_GPIO_5	0x0020	GPIO.5_TXT_DACSEL0
6	CP2114_MASK_GPIO_6	0x0040	GPIO.6_RXT_DACSEL1
7	CP2114_MASK_GPIO_7	0x0080	GPIO.7_RTS_DACSEL2
8	CP2114_MASK_GPIO_8	0x0100	GPIO.8_CTS_DACSEL3
9	CP2114_MASK_GPIO_9	0x0200	GPIO.9_CLKOUT
10	CP2114_MASK_TX	0x0400	GPIO.10_TX
11	CP2114_MASK_RX	0x0800	GPIO.11_RX
12	CP2114_MASK_SUSPEND	0x1000	Suspend
13	CP2114_MASK_SUSPEND_BAR	0x2000	/Suspend

clkDiv—Divider applied to GPIO9./CLK clock output when the pin is configured to CLK Output-Push Pull. When 0, the output frequency is SYSCLK/(2x256). For 1-255, the output frequency is SYSCLK/(2 x clkDiv). SYSCLK can be either 48MHz or 49.152MHz depending on the configuration.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5.3. CP2114_GetPinConfig

Description: Retrieves the GPIO mode configuration for each pin.

Prototype: `HID_UART_STATUS CP2114_GetPinConfig(HID_UART_DEVICE device, BYTE* pinConfig, BOOL* useSuspendValues, WORD* suspendValue, WORD* suspendMode, BYTE* clkDiv)`

Parameters:

1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *pinConfig*—A pointer to a 14-byte array to store GPIO mode configuration or dedicated function for each of the 14 pins.
3. *useSuspendValues*—Returns the configuration for using the values in suspendValue and suspendMode when in suspend mode. This bit is the same as bit 15 of suspendMode.
4. *suspendValue*—Returns the latch value that will be driven on each GPIO pin when the device is in a suspend state.
5. *suspendMode*—Returns the mode for each GPIO pin when the device is in a suspend state.
6. *clkDiv*—Divider applied to GPIO.9_CLKOUT clock output. When 0, the output frequency is SYSCLK/(2 x 256). For 1–255, the output frequency is SYSCLK/(2 x clkDiv). SYSCLK can be either 48 MHz or 49.152 MHz depending on the configuration.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5.4. CP2114_GetDeviceStatus

Description: Returns the status of the device (the device status is cleared on a read).

Prototype: `HID_UART_STATUS CP2114_GetDeviceStatus(HID_UART_DEVICE device, BYTE *pCP2114Status)`

Parameters:
1. *device*—Device object pointer as returned by HidUart_Open().
2. *pCP2114Status*—Pointer to store status byte.

Definition	Value	Description
HID_UART_SUCCESS	0x00	Last command produced no error
HID_UART_INVALID_CONFIG_NUMBER	0x20	Requested configuration number exceeded max configurations of 32
HID_UART_BOOT_INDEXES_DEPLETED	0x21	All boot indices have been used
HID_UART_REQUESTED_CONFIG_NOT_PRESENT	0x22	Pointer to requested configuration is 0xFFFF
HID_UART_CONFIG_INVALID	0x23	Specified configuration consists of invalid parameters
HID_UART_CONFIG_POINTERS_DEPLETED	0x24	All configuration pointer slots have been used
HID_UART_CONFIG_SPACE_DEPLETED	0x25	Not enough space to save the new Config
HID_UART_BOOT_INDEX_UNCHANGED	0x26	The user-specified boot index is already the current boot index stored in OTP
HID_UART_CONFIG_UNCHANGED	0x27	Current configuration is already the same as the user requested
HID_UART_INVALID_NUMER_OF_CACHED_PARAMS	0x40	Specified tSetParamsForNextGet.params exceeds MAX_CACHED_PARAMS of 4
HID_UART_UNEXPECTED_CACHE_DATA	0x41	Unexpected data in tSetParamsForNext-Get

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5.5. CP2114_GetDeviceCaps

Description: Returns the CP2114 device capabilities.

Prototype: `HID_UART_STATUS CP2114_GetDeviceCaps(HID_UART_DEVICE device, PCP2114_CAPS_STRUCT pCP2114CapsStruct)`

Parameters:

1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *pCP2114CapsStruct*—pointer to store CP2114_CAPS_STRUCT.

```
typedef struct
{
  U8  availableBootIndices;
  U8  availableOtpConfigs;
  U8  currentBootConfig;
  U8  availableOtpConfigSpace_LSB;
  U8  availableOtpConfigSpace_MSB;
}tDeviceCaps;
```

3. *availableBootIndices*—Indicates how many CP2114 OTP Boot indices are left for programming.
4. *availableOtpConfigs*—indicates how many entries are left for programming in the CP2114 configuration table. Three OTP configurations are pre-programmed at factory default.
5. *currentBootConfig*—indicates the current active boot config. This active boot config may be dictated by DAC Select Pins, thus this might not be the boot index in OTP. If currentBootConfig is 0xFF, the device will boot up with no DAC.
6. *availableOtpConfigSpace_LSB*—low byte of OTP space left to support new configurations.
7. *availableOtpConfigSpace_MSB*—high byte of OTP space left to support new configurations.

Return Value: HID_UART_STATUS =
- HID_UART_SUCCESS
- HID_UART_INVALID_DEVICE_OBJECT
- HID_UART_INVALID_PARAMETER
- HID_UART_DEVICE_IO_FAILED
- HID_UART_DEVICE_NOT_SUPPORTED

5.6. CP2114_SetRamConfig

Description: Configures the CP2114 RAM configuration parameters with the given values. These settings are written to the internal structures of the CP2114, they're not retained on power cycle. These values might not take immediate effect, device re-enumeration may be required.

Dynamic switch on different clock sources is not supported using CP2114_SetRamConfig. If clock settings differ from those at boot up, the new configuration should be written to OTP and configured as the new boot index. Then power cycle the device to verify the new settings.

The CP2114 datasheet has more information on the audio configuration string format (CP2114_RAM_CONFIG_STRUCT).

Prototype: `HID_UART_STATUS CP2114_SetRamConfig(HID_UART_DEVICE device, PCP2114_RAM_CONFIG_STRUCT pCP2114RamConfigStruct)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *pCP2114RamConfigStruct*—pointer to CP2114_RAM_CONFIG_STRUCT.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5.7. CP2114_GetRamConfig

Description: Gets the current CP2114 RAM configuration parameters.

Prototype: `HID_UART_STATUS CP2114_GetRamConfig(HID_UART_DEVICE device, PCP2114_RAM_CONFIG_STRUCT pCP2114RamConfigStruct)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *pCP2114RamConfigStruct*—pointer to store the 32-byte RAM config.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5.8. CP2114_SetDacRegisters

Description: Configures the device or attached DAC using multiples of 2-byte or 3-byte sequences.

The first byte is DAC register address or special in-band command.

The following byte(s) is the data to write in the specified DAC register if preceded by DAC register address, or parameter(s) of the in-band command if preceded by reserved in-band command IDs.

Some DACs have 8-bit registers, some have 16-bit registers. For 8-bit registers, 2-byte pairs shall be used. For 16-bit registers, 3-byte triplets shall be used.

See User's Guide for details on in-band commands.

Prototype: `HID_UART_STATUS CP2114_SetDacRegisters(HID_UART_DEVICE device, BYTE* pDacConfigBuffer, BYTE dacConfigBufferLength)`

Parameters:

1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *pDacConfigBuffer*—Pointer to the sequence buffer.
3. *dacConfigBufferLength*—Length in bytes of the sequences.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5.9. CP2114_GetDacRegisters

Description: Reads from the specified DAC registers via the I^2C interface. Unlike CP2114_SetDacRegisters, this API retrieves DAC register settings only without intercepting any in-band commands. The host should ensure valid DAC register addresses are used.

Prototype: `HID_UART_STATUS CP2114_GetDacRegisters(HID_UART_DEVICE device, BYTE dacStartAddress, BYTE dacRegistersToRead, BYTE* pDacConfigBuffer)`

Parameters:

1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *dacStartAddress*—Register address from which to start.
3. *dacRegistersToRead*—Number of registers to read.
4. *pDacConfigBuffer*—Pointer to a buffer to store the data returned from the device. Sufficient space must be allocated to store the data. The size of the DacConfigBuffer is:
   2 (or 3) * dacRegistersToRead.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5.10. CP2114_GetOtpConfig

Description: Retrieves a CP2114 configurationfrom OTP.

Prototype: `HID_UART_STATUS CP2114_GetOtpConfig(HID_UART_DEVICE device, BYTE cp2114ConfigNumber, PCP2114_CONFIG_STRUCT pCP2114ConfigStruct)`

Parameters:

1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *cp2114ConfigNumber* —configuration number to retrieve CP2114 OTP.
3. *pCP2114ConfigStruct* —Pointer to store configuration data returned from the device.

```
typedef struct_CONFIG_STRUCT
{
  CP2114_RAM_CONFIG_STRUCT   ramConfig;
  BYTE                       DacConfiguration[MAX_DAC_CONFIG_SIZE];
}CP2114_CONFIG_STRUCT, *PCP2114_CONFIG_STRUCT;
```

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5.11. CP2114_CreateOtpConfig

Description: Creates a new CP2114 configuration in the available OTP space.

Prototype: `HID_UART_STATUS CP2114_CreateOtpConfig(HID_UART_DEVICE device, WORD configBufferLength, BYTE* pConfigBuffer)`

Parameters:

1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *configBufferLength*—Length in bytes of the configuration to be written to OTP.
3. *pConfigBuffer*—Pointer to the buffer containing configuration structured per CP2114_CONFIG_STRUCT excluding the "U16 Length" in CP2114_RAM_CONFIG_STRUCT. The DLL will automatically insert the Length field.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5.12. CP2114_SetBootConfig

Description: Specifies the CP2114 configuration to be loaded from OTP on boot.

Prototype: `HID_UART_STATUS CP2114_SetBootConfig(HID_UART_DEVICE device, BYTE cp2114ConfigNumber)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *cp2114ConfigNumber*—Configuration Index that will be set as the boot configuration upon reset.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5.13. CP2114_ReadOTP

Description: Returns partial or full OTP customization block. The OTP space must be within the range of 6 kilobytes from address 0x6800.

Prototype: `HID_UART_STATUSCP2114_ReadOTP(HID_UART_DEVICE device, UINT cp2114Address ,BYTE* pReadBuffer, UINT ReadLength)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *cp2114Address* —The OTP address to read from. This address must be in between 0x6800 and 0x7FFFF (inclusive).
3. *pReadBuffer*—Pointer to a byte array buffer to store data read from OTP space.
4. *ReadLength*—Length of OTP data to read in bytes.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

5.14. CP2114_WriteOTP

Description: Writes partial or full OTP customization block. The OTP space must be within the range of 6 kilobytes from address 0x6800.

Prototype: `HID_UART_STATUSCP2114_WriteOTP(HID_UART_DEVICE device, UINT cp2114Address ,BYTE* pWriteBuffer, UINT writeLength)`

Parameters:
1. *device*—Device object pointer as returned by *HidUart_Open()*.
2. *cp2114Address* —The OTP address to start writing to. This address must be in between 0x6800 and 0x7FFFF (inclusive).
3. *pWriteBuffer*—Pointer to a byte array buffer that will contain values to write to the OTP space.
4. *writeLength*—The length of write buffer in bytes. The maximum length is 6K bytes.

Return Value: HID_UART_STATUS = HID_UART_SUCCESS
HID_UART_INVALID_DEVICE_OBJECT
HID_UART_INVALID_PARAMETER
HID_UART_DEVICE_IO_FAILED
HID_UART_DEVICE_NOT_SUPPORTED

6. Port Latch Pin Definition

Table 2 and Table 3 describe the GPIO bit definitions for *latchValue* in *HidUart_ReadLatch()* and *HidUart_WriteLatch()*. The library will remap the bit definitions used by the device to match this structure.

Table 2. CP2110 Port Latch Pin Definition

Bit	Pin Name	Pin Number
0	GPIO.0/CLK	1
1	GPIO.1/RTS	24
2	GPIO.2/CTS	23
3	GPIO.3/RS485	22
4	TX	21
5	RX	20
6	GPIO.4/TX Toggle	19
7	GPIO.5/RX Toggle	18
8	$\overline{\text{SUSPEND}}$	17
9	N/A	
10	GPIO.6	15
11	GPIO.7	14
12	GPIO.8	13
13	GPIO.9	12
14	SUSPEND	11
15	N/A	

Table 3. CP2114 Port Latch Pin Definition

Bit	CP2114 Pin Name	CP2114 Pin Number
0	GPIO.0_RMUTE	30
1	GPIO.1_PMUTE	29
2	GPIO.2_VOL-	14
3	GPIO.3_VOL+	13
4	GPIO.4_RMUTELED	12
5	GPIO.5_TXT_DACSEL0	28
6	GPIO.6_RXT_DACSEL1	11

Table 3. CP2114 Port Latch Pin Definition

7	GPIO.7_RTS_DACSEL2	19
8	GPIO.8_CTS_DACSEL3	20
9	GPIO.9_CLKOUT	22
10	GPIO.10_TX	16
11	GPIO.11_RX	15
12	SUSPEND	18
13	$\overline{\text{SUSPEND}}$	17
14	Not Used	Not Used
15	Not Used	Not Used

7. HID_UART_STATUS Return Codes

Each library function returns an `HID_UART_STATUS` return code to indicate that the function returned successfully or to describe an error. The table below describes each error code.

Definition	Value	Description
HID_UART_SUCCESS	0x00	Function returned successfully.*
HID_UART_DEVICE_NOT_FOUND	0x01	Indicates that no devices are connected or that the specified device does not exist.
HID_UART_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_UART_INVALID_DEVICE_OBJECT	0x03	Indicates that the device object pointer does not match the address of a valid HID-to-UART device.
HID_UART_INVALID_PARAMETER	0x04	Indicates that a pointer value is NULL or that an invalid setting was specified.
HID_UART_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_UART_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_UART_WRITE_ERROR	0x11	Indicates that the write was not successful. This means that the output interrupt report failed or timed out.
HID_UART_READ_TIMED_OUT	0x12	Indicates that a read failed to return the number of bytes requested before the read timeout elapsed. Try increasing the read timeout.
HID_UART_WRITE_TIMED_OUT	0x13	Indicates that a write failed to complete sending the number of bytes requested before the write timeout elapsed. Try increasing the write timeout (should be greater than 0 ms).
HID_UART_DEVICE_IO_FAILED	0x14	Indicates that host was unable to get or set a feature report. The device could have been disconnected.
HID_UART_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 when trying to get HID information.
HID_UART_DEVICE_NOT_SUPPORTED	0x16	Indicates that the current device does not support the corresponding action. Functions listed in this document are for the CP2110/4 only.
HID_UART_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.		

8. Thread Safety

The HID-to-UART 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. *HidUART_Read()* issues a pending read request that cannot be canceled from another thread. If the user calls *HidUART_Close()* in a different thread than the thread in which the read request was created, then the device will not be accessible after calling *HidUart_Close()*. The thread that issued the pending read request must return/terminate successfully before the device can be accessed again. See Section "9. Thread Read Access Models (For Windows)" on page 50 for more information.

9. Thread Read Access Models (For Windows)

There are several common read access models when using the HID-to-UART 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-to-UART device until the thread releases access to the device handle.

The following tables describe five common access models and the expected behavior:

Note: *HidUart_Close()* calls *CancelIo()* prior to calling *CloseHandle()*.

Note: *HidUart_Read()* 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.

Note: *HidUart_CancelIo()* forces any pending requests issued by the same thread to complete (cancelled).

***** Indicates that a read is still pending and was issued in the specified thread.
? 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
HidUart_Open()	—	—
*HidUart_Read()**	—	—
HidUart_Close()	—	OK

Table 5. Split Thread Access Model (Unsafe)

Thread A	Thread B	Result
HidUart_Open()	—	—
—	*HidUart_Read()**	—
HidUart_Close()	—	Error: Device inaccessible
—	Terminate Thread	OK: Thread relinquishes device access

Table 6. Split Thread Access Mode (Safe)

Thread A	Thread B	Result
HidUart_Open()	—	—
—	*HidUart_Read()**	—
—	*HidUart_CancelIo()*	—
HidUart_Close()	—	OK

Table 7. Multi-Thread Access Model (Unsafe)

Thread A	Thread B	Result
HidUart_Open()	—	—
HidUart_Read()?	*HidUart_Read()*?	—
HidUart_Close()	—	*Read()** Thread A: OK *Read()** Thread B: Error: Device inaccessible
—	Terminate Thread	OK: Thread relinquishes device access

Table 8. Multi-Thread Access Model (Safe)

Thread A	Thread B	Result
HidUart_Open()	—	—
HidUart_Read()?	*HidUart_Read()?*	—
—	*HidUart_CancelIo()*	—
HidUart_Close()	—	OK

10. Surprise Removal (For Windows)

HidUart_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/PD. Similarly, if a *DBT_DEVICEREMOVECOMPLETE* message is received, then 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, then 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#. The USB Bridge Knowledge Base can be found at the following URL:

http://www.silabs.com/support/knowledgebase/Pages/USB-bridge-knowledge-base.aspx

DOCUMENT CHANGE LIST

Revision 0.4 to Revision 0.5

- Added support for the CP2114.

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.

Patent Notice
Silicon Labs invests in research and development to help our customers differentiate in the market with innovative low-power, small size, analog-intensive mixed-signal solutions. Silicon Labs' extensive patent portfolio is a testament to our unique approach and world-class engineering team.