

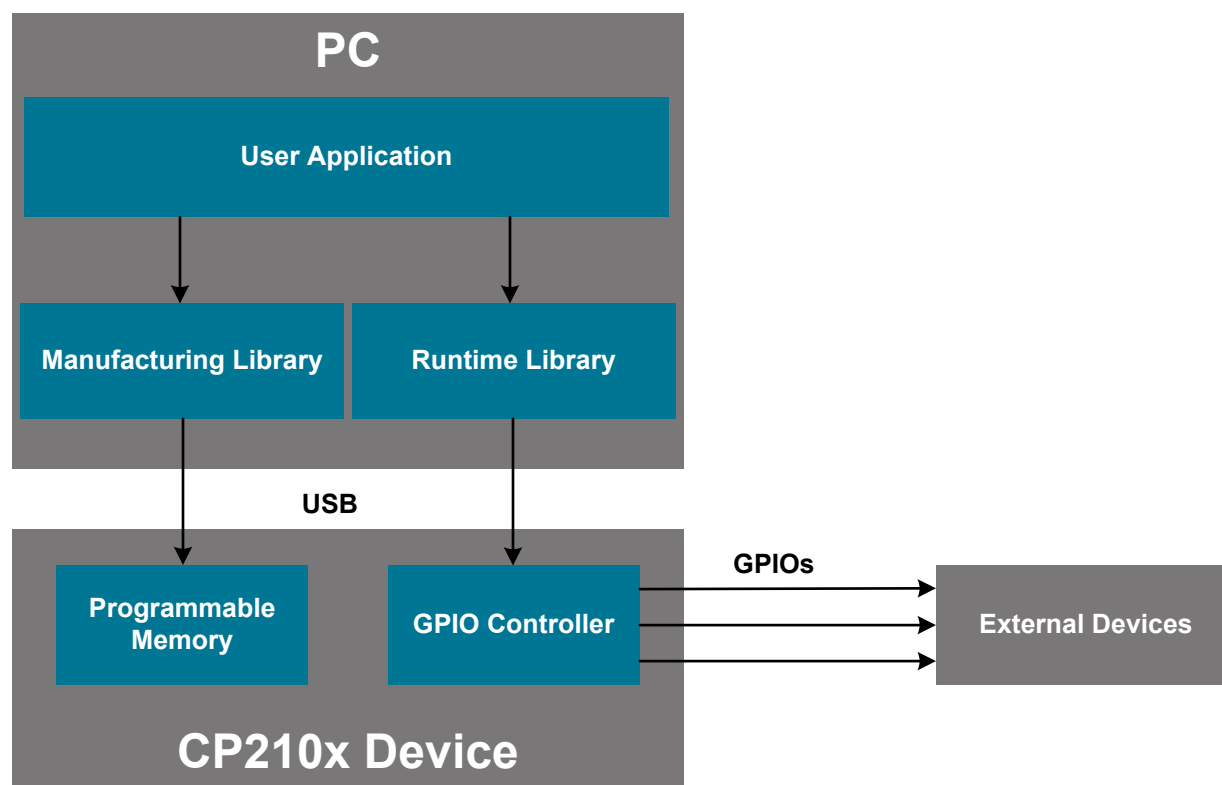
AN978: CP210x USB-to-UART API Specification

This document describes the API for the CP210x Manufacturing Library, used to configure CP210x devices, and the CP210x Runtime Library, used to operate the devices' GPIOs during runtime.

The Silicon Labs CP210x USB-to-UART bridges are devices that communicate over the Universal Serial Bus (USB) to perform Universal Asynchronous Receiver/Transmitter (UART) data transfers. These devices have many programmable options that can be configured via USB. These devices also often include flexible GPIO functions that can be configured and accessed via USB during runtime. Silicon Labs provides libraries that can be used to configure these devices and access their GPIOs.

KEY POINTS

- Silicon Labs provides public libraries to access all available features on CP210x devices.
- Use the CP210x Manufacturing library to configure a device's programmable settings.
- Use the CP210x Runtime library to control a device's GPIOs at runtime.



1. CP210x Host API Functions

Two DLL files can be used to interface with CP210x devices, CP210xManufacturing.DLL, which is responsible for reading and writing the device settings, and CP210xRuntime.DLL, which is responsible for interfacing with the device's GPIOs. The APIs for these DLLs are described in the following sections.

1.1 CP210xManufacturing.DLL

The CP210x Host API is provided as a means to facilitate production of customized CP210x devices. The API allows access to the CP210x device for retrieving and setting the VID, PID, product string, serial number, self- power attribute, maximum power consumption, and device version.

The CP210x Host API is provided in the form of a Windows Dynamic Link Library (DLL), CP210xManufacturing.DLL. The host interface DLL communicates with the bridge controller device via the provided device driver and the operating system's USB stack. The following is a list of the available host API functions:

CP210x_GetNumDevices()	Returns the number of CP210x devices connected.
CP210x_GetProductString()	Returns a descriptor from the registry for a CP210x USB device.
CP210x_GetPartNumber()	Returns the 1-byte Part Number of a CP210x device.
CP210x_Open()	Opens a CP210x device as a USB device and returns a handle.
CP210x_Close()	Closes a CP210x device handle.
CP210x_SetVid()	Sets the 2-byte vendor ID of a CP210x device.
CP210x_SetPid()	Sets the 2-byte product ID of a CP210x device.
CP210x_SetManufacturerString()	Sets the manufacturer description string of a CP210x device.
CP210x_SetProductString()	Sets the product description string of a CP210x device.
CP210x_SetInterfaceString()	Sets the interface string of a CP2105 device.
CP210x_SetSerialNumber()	Sets the serial number string of a CP210x device.
CP210x_SetSelfPower()	Sets the self-power attribute of a CP210x device.
CP210x_SetMaxPower()	Sets the maximum power consumption of a CP210x device.
CP210x_SetFlushBufferConfig()	Sets the flush buffer configuration of CP2104/5 devices.
CP210x_SetDeviceMode()	Sets the operating modes of both interfaces of a CP2105 device.
CP210x_SetDeviceVersion()	Sets version number of the CP210x device.
CP210x_SetBaudRateConfig()	Sets the baud rate configuration data of a CP210x device.
CP210x_SetLockValue()	Sets the 1-byte Lock Value of a CP210x device.
CP210x_SetPortConfig()	Sets the port configuration of a CP2101/2/3/4 device.
CP210x_SetDualPortConfig()	Sets the port configuration of a CP2105 device.
CP210x_SetQuadPortConfig()	Sets the port configuration of a CP2108 device.
CP210x_SetConfig()	Programs the entire configuration of a CP2102N device.

CP210x_GetDeviceManufacturerString()	Gets the manufacturer description string of a CP210x device.
CP210x_GetDeviceProductString()	Gets the product description string of a CP210x device.
CP210x_GetDeviceInterfaceString()	Gets the interface string of a CP2105 device.
CP210x_GetDeviceSerialNumber()	Gets the serial number string of a CP210x device.
CP210x_GetDeviceVid()	Gets the vendor ID of a CP210x device.
CP210x_GetDevicePid()	Gets the product ID of a CP210x device.
CP210x_GetSelfPower()	Gets the self-power attribute of a CP210x device.
CP210x_GetMaxPower()	Gets the maximum power consumption value of a CP210x device.
CP210x_GetFlushBufferConfig()	Gets the flush buffer configuration of CP2104/5 devices.
CP210x_GetDeviceMode()	Gets the operating modes of interfaces of a CP2105 device.
CP210x_GetDeviceVersion()	Gets the version number of a CP210x device.
CP210x_GetBaudRateConfig()	Gets the baud rate configuration data of a CP210x device.
CP210x_GetLockValue()	Gets the 1-byte Lock Value of a CP210x device.
CP210x_GetPortConfig()	Gets the port configuration of a CP210x device.
CP210x_GetDualPortConfig()	Gets the port configuration of a CP2105 device.
CP210x_GetQuadPortConfig()	Gets the port configuration of a CP2108 device.
CP210x_GetFirmwareVersion()	Gets the firmware version of a CP210x device.
CP210x_GetConfig()	Gets the entire configuration of a CP2102N device as a byte array.
CP210x_Reset()	Resets a CP210x device.

In general, the user initiates communication with the target CP210x device by making a call to CP210x_GetNumDevices(). This call returns the number of CP210x target devices. This number is used as a range when calling CP210x_GetProductString() to build a list of devices connected to the host machine.

A handle to the device must first be opened by a call to CP210x_Open() using an index determined from the call to CP210x_GetNumDevices(). The handle will be used for all subsequent accesses. When I/O operations are complete, the device handle is closed by a call to CP210x_Close(). When programming a CP2105 device to configure the mode, the following functions must be called in the following order:

```
CP210x_SetDeviceMode()
CP210x_SetDualPortConfig()
```

The remaining functions are provided to allow access to customizable values contained in the CP210x programmable area.

1.1.1 CP210x_GetNumDevices

Description : This function returns the number of CP210x devices connected to the host.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_GetNumDevices(uint32_t *lpdwNumDevices)

Parameters : 1. lpdwNumDevices—Pointer to a uint32_t that will contain the number of devices.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_DEVICE_NOT_FOUND
- CP210x_INVALID_PARAMETER

1.1.2 CP210x_GetProductString

Description : This function returns a NULL-terminated serial number (S/N) string, product description string, or full path string for the device specified by an index passed in the DeviceNum parameter. The index of the first device is 0, and the index of the last device is the value (NumDevices) returned by CP210x_GetNumDevices() – 1.

Note: This function may return cached data, or data from the device driver. To access the data from the device directly, please use [1.1.24 CP210x_GetDeviceProductString](#).

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_GetProductString(uint32_t dwDeviceIndex, void * lpvDeviceString, uint32_t dwFlags)

Parameters :

1. dwDeviceIndex—Index of the device for which the product description string, serial number, or full path is desired.
2. lpvDeviceString—Variable of type CP210x_DEVICE_STRING returning the NULL-terminated serial number, device description or full path string.
3. dwFlags—Flag that determines if DeviceString contains the product description, serial number, or full-path string. Available options:
 - CP210x_RETURN_SERIAL_NUMBER
 - CP210x_RETURN_DESCRIPTION
 - CP210x_RETURN_FULL_PATH

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_DEVICE_NOT_FOUND
- CP210x_INVALID_PARAMETER

1.1.3 CP210x_GetPartNumber

Description : Returns the 1-byte Part Number contained in a CP210x device.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_GetPartNumber(HANDLE cyHandle, uint8_t *lpbPartNum)

Parameters :

1. cyHandle —Handle to the device as returned by CP210x_Open()
2. lpbPartNum —Pointer to a 1-byte value returning the Part Number of the device. For example, a CP210x_CP2101_DEVICE denotes a CP2101 device, and a CP210x_CP2102_DEVICE denotes a CP2102 device.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.4 CP210x_Open

Description : Opens and returns a handle to a device using a device number determined by the number returned from CP210x_GetNumDevices().

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_Open(uint32_t DeviceIndex, PHANDLE pcyHandle)

Parameters :

1. DeviceIndex —Device index
2. pcyHandle—Pointer to a variable where the handle to the device will be stored. This handle is used for all subsequent accesses to the device.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_DEVICE_NOT_FOUND
- CP210x_INVALID_PARAMETER

1.1.5 CP210x_Close

Description : Closes an open device handle.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_Close(HANDLE cyHandle)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE

1.1.6 CP210x_SetVid

Description : Sets the 2-byte Vendor ID field of the Device Descriptor of a CP210x device.

Supported Devices : CP2101, CP2102, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_SetVid(HANDLE cyHandle, uint16_t Vid)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. Vid—2-byte Vendor ID value.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.7 CP210x_SetPid

Description : Sets the 2-byte Product ID field of the Device Descriptor of a CP210x device.

Supported Devices : CP2101, CP2102, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_SetPid(HANDLE cyHandle, uint16_t Pid)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. Pid—2-byte Product ID value.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.8 CP210x_SetManufacturerString

Description : Sets the Manufacturer Description String of the String Descriptor of a CP210x device. If the string is not already in Unicode format, the function will convert the string to Unicode before committing it to programmable memory. The character size limit (in characters, not bytes), NOT including a NULL terminator, is CP210x_MAX_MANUFACTURER_STRLen, CP2105_MAX_MANUFACTURER_STRLen, or CP2108_MAX_MANUFACTURER_STRLen.

Supported Devices : CP2101, CP2102, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_SetManufacturerString(HANDLE cyHandle, void *lpvManufacturerString, uint8_t bManufacturerStringLength, BOOL bConvertToUnicode = TRUE)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpvManufacturerString—Buffer containing the Manufacturer String value.
3. bManufacturerStringLength—Length of the string in characters (not bytes), NOT including a NULL terminator.
4. bConvertToUnicode—Boolean flag that tells the function if the string needs to be converted to Unicode. The flag is set to TRUE by default (i.e., the string is in ASCII format and needs to be converted to Unicode).

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.9 CP210x_SetProductString

Description : Sets the Product Description String of the String Descriptor of a CP210x device. If the string is not already in Unicode format, the function will convert the string to Unicode before committing it to programmable memory. The character size limit (in characters, not bytes), NOT including a NULL terminator, is CP210x_MAX_PRODUCT_STRLen, CP2105_MAX_PRODUCT_STRLen, or CP2108_MAX_PRODUCT_STRLen.

Supported Devices : CP2101, CP2102, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_SetProductString(HANDLE cyHandle, void *lpvProductString, uint8_t bProductStringLength, BOOL bConvertToUnicode = TRUE)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpvProductString—Buffer containing the Product String value.
3. bProductStringLength—Length of the string in characters (not bytes), NOT including a NULL terminator.
4. bConvertToUnicode—Boolean flag that tells the function if the string needs to be converted to Unicode. The flag is set to TRUE by default (i.e., the string is in ASCII format and needs to be converted to Unicode).

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.10 CP210x_SetInterfaceString

Description : Sets the Interface String for the one of the interfaces available on the CP2105 or CP2108. If the string is not already in Unicode format, the function will convert the string to Unicode before committing it to programmable memory. The character size limit (in characters, not bytes), NOT including a NULL terminator, is CP2105_MAX_INTERFACE_STRLEN or CP2108_MAX_INTERFACE_STRLEN.

Supported Devices : CP2105, CP2108

Prototype : CP210x_STATUS CP210x_SetInterfaceString(HANDLE cyHandle, uint8_t bInterfaceNumber, void * lpvInterfaceString, uint8_t bInterfaceStringLength, BOOL bConvertToUnicode)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. bInterfaceNumber—Set to 0 for Enhanced Interface String, or 1 for Standard Interface String on the CP2105. 0-3 for the CP2108, which has 4 interfaces.
3. lpvInterfaceString—Buffer containing the Interface String.
4. bInterfaceStringLength—Length of the string in characters (not bytes), NOT including a NULL terminator.
5. bConvertToUnicode—Boolean flag that tells the function if the string needs to be converted to Unicode. The flag is set to TRUE by default (i.e., the string is in ASCII format and needs to be converted to Unicode).

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.11 CP210x_SetSerialNumber

Description : Sets the Serial Number String of the String Descriptor of a CP210x device. If the string is not already in Unicode format, the function will convert the string to Unicode before committing it to programmable memory. The character size limit (in characters, not bytes), NOT including a NULL terminator, is CP210x_MAX_SERIAL_STRLEN.

Supported Devices : CP2101, CP2102, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_SetSerialNumber(HANDLE cyHandle, void *lpvSerialNumberString, uint8_t bSerialNumberStringLength, BOOL bConvertToUnicode = TRUE)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpvSerialNumberString—Buffer containing the Serial Number String value.
3. bSerialNumberStringLength—Length in characters (not bytes), NOT including a NULL terminator.
4. bConvertToUnicode—Boolean flag that tells the function if the string needs to be converted to Unicode. The flag is set to TRUE by default, i.e. the string is in ASCII format and needs to be converted to Unicode.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.12 CP210x_SetSelfPower

Description : Sets or clears the Self-Powered bit of the Power Attributes field of the Configuration Descriptor of a CP210x device.

Supported Devices : CP2101, CP2102, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_SetSelfPower(HANDLE cyHandle, BOOL bSelfPower)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. bSelfPower—Boolean flag where TRUE means set the Self-Powered bit, and FALSE means clear the Self-Powered bit.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.13 CP210x_SetMaxPower

Description : Sets the Max Power field of the Configuration Descriptor of a CP210x device.

Supported Devices : CP2101, CP2102, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_SetMaxPower(HANDLE cyHandle, uint8_t bMaxPower)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. bMaxPower—1-byte value representing the maximum power consumption of the CP210x USB device, expressed in 2 mA units.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.14 CP210x_SetFlushBufferConfig

Description : Sets the Flush Buffer configuration of a CP210x device.

Supported Devices : CP2104, CP2105, CP2108

Prototype : CP210x_STATUS CP210x_SetFlushBufferConfig(HANDLE cyHandle, uint16_t wFlushBufferConfig)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. wFlushBufferConfig—Set to determine which buffer(s) to flush (TX and/or RX) and upon which event (Open and/or Close). See the CP210xManufacturingDLL.h or [1.1.43 CP210x Manufacturing Library Type Definitions and Constants](#) for the bit definitions for this byte value.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_FUNCTION_NOT_SUPPORTED
- CP210x_DEVICE_NOT_FOUND

1.1.15 CP210x_SetDeviceMode

Description : Sets the operating mode (GPIO or Modem) or each Interface of a CP2105 device.

Supported Devices : CP2105

Prototype : CP210x_STATUS CP210x_SetDeviceMode(HANDLE cyHandle, uint8_t bDeviceModeECI, uint8_t bDeviceModeSCI)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. bDeviceModeECI—Set to 0 for modem mode for Enhanced interface
3. bDeviceModeSCI—Set to 0 for modem mode for Standard interface

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_NOT_FOUND
- CP210x_FUNCTION_NOT_SUPPORTED

1.1.16 CP210x_SetDeviceVersion

Description : Sets the Device Release Version field of the Device Descriptor of a CP210x device.

Supported Devices : CP2101, CP2102, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_SetDeviceVersion(HANDLE cyHandle, uint16_t wVersion)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. wVersion—2-byte Device Release Version number in Binary-Coded Decimal (BCD) format with the upper two nibbles containing the two decimal digits of the major version and the lower two nibbles containing the two decimal digits of the minor version.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.17 CP210x_SetBaudRateConfig

Description : Sets the baud rate configuration data of a CP210x device.

Supported Devices : CP2102, CP2103

Prototype : CP210x_STATUS CP210x_SetBaudRateConfig(HANDLE cyHandle, PBAUD_CONFIG baudConfigData)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. baudConfigData—Pointer to a BAUD_CONFIG structure containing the Baud Config data to be set on the device.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.18 CP210x_SetLockValue

Description : Sets the 1-byte Lock Value of a CP210x device.

Note: Setting the lock value locks ALL customizable data and cannot be reset; only use this function to keep all customizable data on the part permanently.

Supported Devices : CP2102, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_SetLockValue(HANDLE cyHandle)

Parameters : 1. cyHandle—Handle to the device as returned by CP210x_Open()

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.19 CP210x_SetPortConfig

Description : Sets the current port pin configuration from the CP210x device.

Supported Devices : CP2103, CP2104

Prototype : CP210x_STATUS CP210x_SetPortConfig(HANDLE cyHandle, PORT_CONFIG *PortConfig)

Parameters : 1. cyHandle—Handle to the device as returned by CP210x_Open()
2. PortConfig—Pointer to a PORT_CONFIG structure

Return Value : CP210x_STATUS

- CP210X_SUCCESS
- CP210X_INVALID_HANDLE
- CP210X_DEVICE_IO_FAILED
- CP210X_UNSUPPORTED_DEVICE

1.1.20 CP210x_SetDualPortConfig

Description : Sets the current port pin configuration from the CP2105 device. SetDeviceMode() must be called before calling this function.

Supported Devices : CP2105

Prototype : CP210x_STATUS CP210x_SetDualPortConfig(HANDLE cyHandle, DUAL_PORT_CONFIG *DualPortConfig)

Parameters : 1. cyHandle—Handle to the device as returned by CP210x_Open()
2. DualPortConfig—Pointer to a DUAL_PORT_CONFIG structure

Return Value : CP210x_STATUS

- CP210X_SUCCESS
- CP210X_INVALID_HANDLE
- CP210X_DEVICE_IO_FAILED
- CP210X_UNSUPPORTED_DEVICE

1.1.21 CP210x_SetQuadPortConfig

Description : Sets the current port pin configuration from the CP2108 device.

Supported Devices : CP2108

Prototype : CP210x_STATUS CP210x_SetQuadPortConfig(HANDLE cyHandle, QUAD_PORT_CONFIG *QuadPortConfig)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. QuadPortConfig—Pointer to a QUAD_PORT_CONFIG structure.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED
- CP210x_UNSUPPORTED_DEVICE

1.1.22 CP210x_SetConfig

Description : Programs the device's configurable area with the given byte array.

Supported Devices : CP2102N

Prototype : CP210x_STATUS CP210x_SetConfig(HANDLE cyHandle, uint8_t *lpbConfig, uint16_t bLength)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpbConfig—A pointer to a byte array that holds the configuration to be programmed to the device (see [1.1.41 CP2102N Configuration Array lpbConfig](#) for details).
3. bLength—The length of the given byte array.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.23 CP210x_GetDeviceManufacturerString

Description : Returns the Manufacturer Description String of the String Descriptor of a CP210x device. If the Convert-ToASCII parameter is set, the string will be converted to ASCII format before being returned to the caller. The character size limit (in characters, not bytes), NOT including a NULL terminator, is CP210x_MAX_MANUFACTURER_STRLEN, CP2105_MAX_MANUFACTURER_STRLEN, or CP2108_MAX_MANUFACTURER_STRLEN.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_GetDeviceManufacturerString(HANDLE cyHandle, void *lpManufacturer, uint8_t *lpbLength, BOOL bConvertToASCII = TRUE)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpManufacturer—Pointer to a buffer returning the Manufacturer String value.
3. lpbLength—Pointer to a BYTE value returning the length of the string in characters (not bytes), NOT including a NULL terminator.
4. bConvertToASCII—Boolean flag that tells the function whether the string needs to be converted to ASCII before it is returned to the caller. The flag is set to TRUE by default (i.e., the caller is expecting the string in ASCII format).

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.24 CP210x_GetDeviceProductString

Description : Returns the Product Description String of the String Descriptor of a CP210x device. If the Convert-ToASCII parameter is set, the string will be converted to ASCII format before being returned to the caller. The character size limit (in characters, not bytes), NOT including a NULL terminator, is CP210x_MAX_PRODUCT_STRLEN, CP2105_MAX_PRODUCT_STRLEN, CP2108_MAX_PRODUCT_STRLEN.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_GetDeviceProductString(HANDLE cyHandle, void *lpProduct, uint8_t *lpbLength, BOOL bConvertToASCII = TRUE)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpProduct—Pointer to a buffer returning the Product String value.
3. lpbLength—Pointer to a BYTE value returning the length of the string in characters (not bytes), NOT including a NULL terminator.
4. bConvertToASCII—Boolean flag that tells the function whether the string needs to be converted to ASCII before it is returned to the caller. The flag is set to TRUE by default (i.e., the caller is expecting the string in ASCII format).

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.25 CP210x_GetDeviceInterfaceString

Description : Gets the specified interface string from a CP210x device. If the ConvertToASCII parameter is set, the string will be converted to ASCII format before being returned to the caller. The character size limit (in characters, not bytes), NOT including a NULL terminator, is CP2105_MAX_INTERFACE_STRLen or CP2108_MAX_INTERFACE_STRLen.

Supported Devices : CP2105, CP2108

Prototype : CP210x_STATUS CP210x_GetDeviceInterfaceString(HANDLE cyHandle, uint8_t bInterfaceNumber, void * lpInterface, uint8_t *lpbLength, BOOL bConvertToASCII)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. bInterfaceNumber —Set to 0 for Enhanced Interface.
3. lpInterface—Pointer to buffer returning the selected Interface String value.
4. lpbLength—Pointer to a uint8_t value returning the length of the string in characters (not bytes), NOT including a NULL terminator.
5. bConvertToASCII—Boolean flag that tells the function whether the string needs to be converted to ASCII before it is returned to the caller. The flag is set to TRUE by default (i.e., the caller is expecting the string in ASCII format).

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.26 CP210x_GetDeviceSerialNumber

Description : Gets the Serial Number String of the String Descriptor of a CP210x device. If the ConvertToASCII parameter is set, the string will be converted to ASCII format before being returned to the caller. The character size limit (in characters, not bytes), NOT including a NULL terminator, is CP210x_MAX_SERIAL_STRLen.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_GetDeviceSerialNumber(HANDLE cyHandle, void *lpSerialNumber, uint8_t *lpbLength, BOOL bConvertToASCII = TRUE)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpSerialNumber —Pointer to a buffer returning the Serial Number String value.
3. lpbLength—Pointer to a uint8_t value returning the length of the string in characters (not bytes), NOT including a NULL terminator.
4. bConvertToASCII—Boolean flag that tells the function whether the string needs to be converted to ASCII before it is returned to the caller. The flag is set to TRUE by default (i.e., the caller is expecting the string in ASCII format).

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.27 CP210x_GetDeviceVid

Description : Returns the 2-byte Vendor ID field of the Device Descriptor of a CP210x device.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_GetDeviceVid(HANDLE cyHandle, uint16_t *lpwVid)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpwVid—Pointer to a 2-byte value that returns the Vendor ID of the CP210x device.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.28 CP210x_GetDevicePid

Description : Returns the 2-byte Product ID field of the Device Descriptor of a CP210x device.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_GetDevicePid(HANDLE cyHandle, uint16_t *lpwPid)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpwPid—Pointer to a 2-byte value that returns the Product ID of the CP210x device.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.29 CP210x_GetSelfPower

Description : Returns the state of the Self-Powered bit of the Power Attributes field of the Configuration Descriptor of a CP210x device.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_GetSelfPower(HANDLE cyHandle, LPBOOL lpbSelfPower)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpbSelfPower—Pointer to a boolean flag where TRUE means the Self-Powered bit is set, and FALSE means the Self-Powered bit is cleared.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.30 CP210x_GetMaxPower

Description : Returns the 1-byte Max Power field of the Configuration Descriptor of a CP210x device.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_GetMaxPower(HANDLE cyHandle, uint8_t *lpbPower)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpbPower—Pointer to a 1-byte value returning the Maximum power consumption of the CP210x USB device expressed in 2 mA units.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.31 CP210x_GetFlushBufferConfig

Description : Returns the flush buffer configuration of a CP210x device.

Supported Devices : CP2104, CP2105, CP2108

Prototype : CP210x_STATUS CP210x_GetFlushBufferConfig(HANDLE cyHandle, uint16_t *lpwFlushBufferConfig)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpwFlushBufferConfig—Pointer to the values which indicates which buffer(s) are flushed (TX and/ or RX) and upon which event (Open and/or Close). See the CP210xManufacturingDLL.h or [1.1.43 CP210x Manufacturing Library Type Definitions and Constants](#) for the bit definitions for this byte value.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_DEVICE_NOT_FOUND
- CP210x_INVALID_HANDLE
- CP210x_FUNCTION_NOT_SUPPORTED

1.1.32 CP210x_GetDeviceMode

Description : Gets the operating mode (GPIO or Modem) or each Interface of a CP2105 device.

Supported Devices : CP2105

Prototype : CP210x_STATUS CP210x_GetDeviceMode(HANDLE cyHandle, uint8_t *lpbDeviceModeECI, uint8_t *lpbDeviceModesCI)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpbDeviceModeECI—Pointer to a 1-byte value returning the 0 if interface is in Modem mode, or 1 if GPIO mode.
3. lpbDeviceModesCI—Pointer to a 1-byte value returning the 0 if interface is in Modem mode, or 1 if GPIO mode.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_NOT_FOUND
- CP210x_FUNCTION_NOT_SUPPORTED

1.1.33 CP210x_GetBaudRateConfig

Description : Returns the baud rate configuration data of a CP210x device.

Supported Devices : CP2102, CP2103, CP2109

Prototype : CP210x_STATUS CP210x_GetBaudRateConfig(HANDLE cyHandle, PBAUD_CONFIG pBaudConfig)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. pBaudConfig—Pointer to a BAUD_CONFIG array containing structures returning the Baud Config data of the device.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.34 CP210x_GetLockValue

Description : Returns the 1-byte Lock Value of a CP210x device.

Supported Devices : CP2102, CP2103, CP2104, CP2105, CP2108

Prototype : CP210x_STATUS CP210x_GetLockValue(HANDLE cyHandle, uint8_t *lpbLockValue)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpbLockValue—Pointer to a 1-byte value returning the Lock Value of the device that the device is locked, and a 0x00 denotes that the device is unlocked.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_PARAMETER
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.35 CP210x_GetPortConfig

Description : Gets the current port pin configuration from the CP210x device.

Supported Devices : CP2103, CP2104

Prototype : CP210x_STATUS CP210x_GetPortConfig(HANDLE cyHandle, PPORT_CONFIG pPortConfig)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. pPortConfig—Pointer to a PORT_CONFIG structure.

Return Value : CP210x_STATUS

- CP210X_SUCCESS
- CP210X_INVALID_HANDLE
- CP210X_DEVICE_IO_FAILED
- CP210X_UNSUPPORTED_DEVICE

1.1.36 CP210x_GetDualPortConfig

Description : Gets the current port pin configuration from the CP2105 device.

Supported Devices : CP2105

Prototype : CP210x_STATUS CP210x_GetDualPortConfig(HANDLE cyHandle, DUAL_PORT_CONFIG* pDualPortConfig)

Parameters : 1. cyHandle—Handle to the device as returned by CP210x_Open()
2. pDualPortConfig—Pointer to a DUAL_PORT_CONFIG structure.

Return Value : CP210x_STATUS

- CP210X_SUCCESS
- CP210X_INVALID_HANDLE
- CP210X_DEVICE_IO_FAILED
- CP210X_UNSUPPORTED_DEVICE

1.1.37 CP210x_GetQuadPortConfig

Description : Gets the current port pin configuration from the CP2108 device.

Supported Devices : CP2108

Prototype : CP210x_STATUS CP210x_GetQuadPortConfig(HANDLE cyHandle, QUAD_PORT_CONFIG* pQuadPortConfig)

Parameters : 1. cyHandle—Handle to the device as returned by CP210x_Open()
2. pQuadPortConfig—Pointer to a QUAD_PORT_CONFIG structure.

Return Value : CP210x_STATUS

- CP210X_SUCCESS
- CP210X_INVALID_HANDLE
- CP210X_DEVICE_IO_FAILED
- CP210X_UNSUPPORTED_DEVICE

1.1.38 CP210x_GetFirmwareVersion

Description : Retrieves the firmware version from the device.

Supported Devices : CP2102N, CP2108

Prototype : CP210x_STATUS CP210x_GetFirmwareVersion(HANDLE cyHandle, pFirmware_t lpVersion)

Parameters : 1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpVersion—3-byte structure that indicates major, minor, and build version numbers.

Return Value : CP210x_STATUS

- CP210X_SUCCESS
- CP210X_INVALID_HANDLE
- CP210X_INVALID_PARAMETER
- CP210X_FUNCTION_NOT_SUPPORTED
- CP210X_DEVICE_IO_FAILED

1.1.39 CP210x_GetConfig

Description : Retrieves the current configuration from the device as a byte array.

Supported Devices : CP2102N

Prototype : CP210x_STATUS CP210x_GetConfig(HANDLE cyHandle, uint8_t *lpbConfig, uint16_t bLength)

Parameters :

1. cyHandle—Handle to the device as returned by CP210x_Open()
2. lpbConfig—A pointer to a byte array to hold the configuration (see [1.1.41 CP2102N Configuration Array lpbConfig](#) for details).
3. bLength—The length of the given byte array.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.40 CP210x_Reset

Description : Initiates a reset of the USB interface.

Note: There is a delay of ~1 second before the reset is initiated by the device firmware to give the application time to call CP210x_Close() to close the device handle. No further operations should be performed with the device until it resets, re-enumerates in Windows, and a new handle is opened.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210x_Reset(HANDLE cyHandle)

Parameters :

1. cyHandle—Handle to the device to close as returned by CP210x_Open().

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED

1.1.41 CP2102N Configuration Array lpbConfig

Table 1.1 CP2102N Configuration Array Map on page 19 describes all bytefields within the CP2102N's configuration array. This array is used to program the device using the CP210x_SetConfig command and is retrieved by the CP210x_GetConfig command in the CP210x_Manufacturing.dll.

The table lists the following attributes for each entry in the array:

- **Property ID** : the ID used internally by Xpress Configurator for this field
- **Label** : the "User Friendly" label that is usually displayed in the Xpress Configurator UI
- **Length** : the length of the field in bytes or bits
- **Offset** : the offset of the field from the beginning of the configuration structure
- **Default Value** : the default value of the field
- **Type** : "User" if the property can be modified by the user, "-" otherwise

All rows that describe user-settable properties are identified by the term "User" in the **Type** column. Other properties (denoted by "-" and gray text) should be left at their default values.

Note: The last entry in the table is the Fletcher Checksum of the rest of the array. Details on how to calculate this value are provided in section 1.1.42 Fletcher Checksum. This checksum property must be updated if any other properties in the configuration array are modified from their default values.

Table 1.1. CP2102N Configuration Array Map

Property ID	Label	Length	Offset	Default Value	Type
cp2102n.configSize	Config Size	2 B	0	0x02A6	-
cp2102n.configVersion	Config Version	1 B	2	0x01	-
cp2102n.enableBootloader	Enable Bootloader	1 B	3	0xFF	-
cp2102n.enableConfigUpdate	Enable Config Update	1 B	4	0xFF	User
cp2102n.deviceDesc.bLength	B Length	1 B	5	0x12	-
cp2102n.deviceDesc.bDescriptorType	B Descriptor Type	1 B	6	0x01	-
cp2102n.deviceDesc.bcdUSB	Bcd USB	2 B	7	0x0200	-
cp2102n.deviceDesc.bDeviceClass	B Device Class	1 B	9	0x00	-
cp2102n.deviceDesc.bDeviceSubClass	B Device Sub Class	1 B	10	0x00	-
cp2102n.deviceDesc.bDeviceProtocol	B Device Protocol	1 B	11	0x00	-
cp2102n.deviceDesc.bMaxPacketSize0	B Max Packet Size0	1 B	12	0x40	-
cp210x_base.set_ids.VID	Vid	2 B	13	0x10C4	User
cp210x_base.set_ids.PID	Pid	2 B	15	0xEA60	User
cp210x_base.set_ids.releaseVersion	Release Version	2 B	17	0x0100	User
cp2102n.deviceDesc.iManufacturer	I Manufacturer	1 B	19	0x01	-
cp2102n.deviceDesc.iProduct	I Product	1 B	20	0x02	-
cp2102n.deviceDesc.iSerialNumber	I Serial Number	1 B	21	0x03	-
cp2102n.deviceDesc.bNumConfigurations	B Num Configurations	1 B	22	0x01	-
cp2102n.configDesc.bLength	B Length	1 B	23	0x09	-
cp2102n.configDesc.bDescriptorType	B Descriptor Type	1 B	24	0x02	-
cp2102n.configDesc.wTotalLength	W Total Length	2 B	25	0x0020	-
cp2102n.configDesc.bNumInterfaces	B Num Interfaces	1 B	27	0x01	-

Property ID	Label	Length	Offset	Default Value	Type
cp2102n.configDesc.bConfigurationValue	B Configuration Value	1 B	28	0x01	-
cp2102n.configDesc.iConfiguration	I Configuration	1 B	29	0x00	-
[BITFIELD]	main.configDesc.bmAttributes	1 B	30		-
BIT_PADDING_0	Bit Padding 0	1 bit	30[0]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	30[1]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	30[2]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	30[3]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	30[4]	0	-
cp2102n.configDesc.bmAttributes.remoteWakeup	Remote Wakeup	1 bit	30[5]	0	User
cp210x_base.set_ids.powerMode	Power Mode	1 bit	30[6]	0	User
BIT_PADDING_1	Bit Padding 1	1 bit	30[7]	1	-
cp210x_base.set_ids.maxPower_real	Max Power Real	1 B	31	0x32	-
cp2102n.interfaceDescriptor.bLength	B Length	1 B	32	0x09	-
cp2102n.interfaceDescriptor.bDescriptorType	B Descriptor Type	1 B	33	0x04	-
cp2102n.interfaceDescriptor.bInterfaceNumber	B Interface Number	1 B	34	0x00	-
cp2102n.interfaceDescriptor.bAlternateSetting	B Alternate Setting	1 B	35	0x00	-
cp2102n.interfaceDescriptor.bNumEndpoints	B Num Endpoints	1 B	36	0x02	-
cp2102n.interfaceDescriptor.bInterfaceClass	B Interface Class	1 B	37	0xFF	-
cp2102n.interfaceDescriptor.bInterfaceSubClass	B Interface Sub Class	1 B	38	0x00	-
cp2102n.interfaceDescriptor.bInterfaceProtocol	B Interface Protocol	1 B	39	0x00	-
cp2102n.interfaceDescriptor.iInterface	I Interface	1 B	40	0x00	-
cp2102n.bulkOut.bLength	B Length	1 B	41	0x07	-
cp2102n.bulkOut.bDescriptorType	B Descriptor Type	1 B	42	0x05	-
cp2102n.bulkOut.bEndpointAddress	B Endpoint Address	1 B	43	0x02	-
cp2102n.bulkOut.bmAttributes	Bm Attributes	1 B	44	0x02	-
cp2102n.bulkOut.wMaxPacketSize	W Max Packet Size	2 B	45	0x0040	-
cp2102n.bulkOut.bInterval	B Interval	1 B	47	0x00	-
cp2102n.bulkIn.bLength	B Length	1 B	48	0x07	-
cp2102n.bulkIn.bDescriptorType	B Descriptor Type	1 B	49	0x05	-
cp2102n.bulkIn.bEndpointAddress	B Endpoint Address	1 B	50	0x82	-
cp2102n.bulkIn.bmAttributes	Bm Attributes	1 B	51	0x02	-
cp2102n.bulkIn.wMaxPacketSize	W Max Packet Size	2 B	52	0x0040	-
cp2102n.bulkIn.bInterval	B Interval	1 B	54	0x00	-
cp2102n.langDesc.langDesc[0]	Lang Desc[0]	2 B	55	0x0304	-
cp2102n.langDesc.langDesc[1]	Lang Desc[1]	2 B	57	0x0409	-
BIT_PADDING_0	Bit Padding 0	1 B	59	0x00	-

Property ID	Label	Length	Offset	Default Value	Type
cp2102n.mfrDesc.mfrDescLen	Mfr Desc Len	1 B	60	0x1A	User
USB_STRING_DESCRIPTOR	USB String Descriptor	1 B	61	0x03	-
cp210x_base.set_ids.manufacturerString	Manufacturer String	128 B	62	Silicon Labs	User
BIT_PADDING_0	Bit Padding 0	1 B	190	0x00	-
cp2102n.prodDesc.prodDescLen	Prod Desc Len	1 B	191	0x4C	User
USB_STRING_DESCRIPTOR	USB String Descriptor	1 B	192	0x03	-
cp210x_base.set_ids.productString	Product String	256 B	193	CP2102N USB to UART Bridge Controller	User
cp2102n.useInternalSerial	Use Internal Serial	1 B	449	0x00	User
cp2102n.serDesc.serDescLen	Ser Desc Len	1 B	450	0x0A	User
USB_STRING_DESCRIPTOR	USB String Descriptor	1 B	451	0x03	-
cp210x_base.set_ids.serialString	Serial String	128 B	452	1	User
[BITFIELD]	main.Reset Mode P0	1 B	580		-
cp2102n.portSettings.gpio.RI.reset.Mode	Mode	1 bit	580[0]	0	User
cp2102n.portSettings.gpio.DCD.reset.Mode	Mode	1 bit	580[1]	0	User
cp2102n.portSettings.gpio.DTR.reset.Mode	Mode	1 bit	580[2]	1	User
cp2102n.portSettings.gpio.DSR.reset.Mode	Mode	1 bit	580[3]	0	User
cp2102n.portSettings.gpio.TXD.reset.Mode	Mode	1 bit	580[4]	1	User
cp2102n.portSettings.gpio.RXD.reset.Mode	Mode	1 bit	580[5]	0	User
cp2102n.portSettings.gpio.RTS.reset.Mode	Mode	1 bit	580[6]	1	User
cp2102n.portSettings.gpio.CTS.reset.Mode	Mode	1 bit	580[7]	0	User
[BITFIELD]	main.Reset Mode P1	1 B	581		-
cp2102n.portSettings.gpio.SUSPEND.reset.Mode	Mode	1 bit	581[0]	1	User
cp2102n.portSettings.gpio.UNUSED.reset.Mode	Mode	1 bit	581[1]	0	-
cp2102n.portSettings.gpio.SUSPENDb.reset.Mode	Mode	1 bit	581[2]	1	User
cp2102n.portSettings.gpio.GPIO0.reset.Mode	Mode	1 bit	581[3]	0	User
cp2102n.portSettings.gpio.GPIO1.reset.Mode	Mode	1 bit	581[4]	0	User
cp2102n.portSettings.gpio.GPIO2.reset.Mode	Mode	1 bit	581[5]	0	User
cp2102n.portSettings.gpio.GPIO3.reset.Mode	Mode	1 bit	581[6]	0	User
cp2102n.portSettings.gpio.UNUSED.reset.Mode	Mode	1 bit	581[7]	0	-
[BITFIELD]	main.Reset Mode P2	1 B	582		-
cp2102n.portSettings.gpio.UNUSED.reset.Mode	Mode	1 bit	582[0]	0	-
cp2102n.portSettings.gpio.UNUSED.reset.Mode	Mode	1 bit	582[1]	0	-
cp2102n.portSettings.gpio.UNUSED.reset.Mode	Mode	1 bit	582[2]	0	-
cp2102n.portSettings.gpio.UNUSED.reset.Mode	Mode	1 bit	582[3]	0	-

Property ID	Label	Length	Offset	Default Value	Type
BIT_PADDING_0	Bit Padding 0	1 bit	582[4]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	582[5]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	582[6]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	582[7]	0	-
[BITFIELD]	main.Reset Low Power P0	1 B	583		-
cp2102n.portSettings.gpio.RI.reset.LowPower	Low Power	1 bit	583[0]	1	User
cp2102n.portSettings.gpio.DCD.reset.LowPower	Low Power	1 bit	583[1]	1	User
cp2102n.portSettings.gpio.DTR.reset.LowPower	Low Power	1 bit	583[2]	1	User
cp2102n.portSettings.gpio.DSR.reset.LowPower	Low Power	1 bit	583[3]	1	User
cp2102n.portSettings.gpio.TXD.reset.LowPower	Low Power	1 bit	583[4]	1	User
cp2102n.portSettings.gpio.RXD.reset.LowPower	Low Power	1 bit	583[5]	1	User
cp2102n.portSettings.gpio.RTS.reset.LowPower	Low Power	1 bit	583[6]	1	User
cp2102n.portSettings.gpio.CTS.reset.LowPower	Low Power	1 bit	583[7]	1	User
[BITFIELD]	main.Reset LowPower P1	1 B	584		-
cp2102n.portSettings.gpio.SUSPEND.reset.LowPower	Low Power	1 bit	584[0]	1	User
cp2102n.portSettings.gpio.UNUSED.reset.LowPower	Low Power	1 bit	584[1]	1	-
cp2102n.portSettings.gpio.SUSPENDb.reset.LowPower	Low Power	1 bit	584[2]	1	User
cp2102n.portSettings.gpio.GPIO0.reset.LowPower	Low Power	1 bit	584[3]	1	User
cp2102n.portSettings.gpio.GPIO1.reset.LowPower	Low Power	1 bit	584[4]	1	User
cp2102n.portSettings.gpio.GPIO2.reset.LowPower	Low Power	1 bit	584[5]	1	User
cp2102n.portSettings.gpio.GPIO3.reset.LowPower	Low Power	1 bit	584[6]	1	User
cp2102n.portSettings.gpio.UNUSED.reset.LowPower	Low Power	1 bit	584[7]	1	-
[BITFIELD]	main.Reset LowPower P2	1 B	585		-
cp2102n.portSettings.gpio.UNUSED.reset.LowPower	Low Power	1 bit	585[0]	1	-
cp2102n.portSettings.gpio.UNUSED.reset.LowPower	Low Power	1 bit	585[1]	1	-
cp2102n.portSettings.gpio.UNUSED.reset.LowPower	Low Power	1 bit	585[2]	1	-
cp2102n.portSettings.gpio.UNUSED.reset.LowPower	Low Power	1 bit	585[3]	1	-
BIT_PADDING_0	Bit Padding 0	1 bit	585[4]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	585[5]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	585[6]	0	-

Property ID	Label	Length	Offset	Default Value	Type
BIT_PADDING_0	Bit Padding 0	1 bit	585[7]	0	-
[BITFIELD]	main.Reset Latch P0	1 B	586		-
cp2102n.portSettings.gpio.RI.reset.Latch	Latch	1 bit	586[0]	1	User
cp2102n.portSettings.gpio.DCD.reset.Latch	Latch	1 bit	586[1]	1	User
cp2102n.portSettings.gpio.DTR.reset.Latch	Latch	1 bit	586[2]	1	User
cp2102n.portSettings.gpio.DSR.reset.Latch	Latch	1 bit	586[3]	1	User
cp2102n.portSettings.gpio.TXD.reset.Latch	Latch	1 bit	586[4]	1	User
cp2102n.portSettings.gpio.RXD.reset.Latch	Latch	1 bit	586[5]	1	User
cp2102n.portSettings.gpio.RTS.reset.Latch	Latch	1 bit	586[6]	1	User
cp2102n.portSettings.gpio.CTS.reset.Latch	Latch	1 bit	586[7]	1	User
[BITFIELD]	main.Reset Latch P1	1 B	587		-
cp2102n.portSettings.gpio.SUSPEND.reset.Latch	Latch	1 bit	587[0]	0	User
cp2102n.portSettings.gpio.UNUSED.reset.Latch	Latch	1 bit	587[1]	1	-
cp2102n.portSettings.gpio.SUSPENDb.reset.Latch	Latch	1 bit	587[2]	1	User
cp2102n.portSettings.gpio.GPIO0.reset.Latch	Latch	1 bit	587[3]	1	User
cp2102n.portSettings.gpio.GPIO1.reset.Latch	Latch	1 bit	587[4]	1	User
cp2102n.portSettings.gpio.GPIO2.reset.Latch	Latch	1 bit	587[5]	1	User
cp2102n.portSettings.gpio.GPIO3.reset.Latch	Latch	1 bit	587[6]	1	User
cp2102n.portSettings.gpio.UNUSED.reset.Latch	Latch	1 bit	587[7]	1	-
[BITFIELD]	main.Reset Latch P2	1 B	588		-
cp2102n.portSettings.gpio.UNUSED.reset.Latch	Latch	1 bit	588[0]	1	-
cp2102n.portSettings.gpio.UNUSED.reset.Latch	Latch	1 bit	588[1]	1	-
cp2102n.portSettings.gpio.UNUSED.reset.Latch	Latch	1 bit	588[2]	1	-
cp2102n.portSettings.gpio.UNUSED.reset.Latch	Latch	1 bit	588[3]	1	-
BIT_PADDING_0	Bit Padding 0	1 bit	588[4]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	588[5]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	588[6]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	588[7]	0	-
[BITFIELD]	main.Suspend Mode P0	1 B	589		-
cp2102n.portSettings.gpio.RI.suspend.Mode	Mode	1 bit	589[0]	0	User
cp2102n.portSettings.gpio.DCD.suspend.Mode	Mode	1 bit	589[1]	0	User
cp2102n.portSettings.gpio.DTR.suspend.Mode	Mode	1 bit	589[2]	1	User
cp2102n.portSettings.gpio.DSR.suspend.Mode	Mode	1 bit	589[3]	0	User
cp2102n.portSettings.gpio.TXD.suspend.Mode	Mode	1 bit	589[4]	1	User

Property ID	Label	Length	Offset	Default Value	Type
cp2102n.portSettings.gpio.RXD.suspend.Mode	Mode	1 bit	589[5]	0	User
cp2102n.portSettings.gpio.RTS.suspend.Mode	Mode	1 bit	589[6]	1	User
cp2102n.portSettings.gpio.CTS.suspend.Mode	Mode	1 bit	589[7]	0	User
[BITFIELD]	main.Suspend Mode P1	1 B	590		-
cp2102n.portSettings.gpio.SUSPEND.suspend.Mode	Mode	1 bit	590[0]	1	User
cp2102n.portSettings.gpio.UNUSED.suspend.Mode	Mode	1 bit	590[1]	0	-
cp2102n.portSettings.gpio.SUSPENDb.suspend.Mode	Mode	1 bit	590[2]	1	User
cp2102n.portSettings.gpio.GPIO0.suspend.Mode	Mode	1 bit	590[3]	0	User
cp2102n.portSettings.gpio.GPIO1.suspend.Mode	Mode	1 bit	590[4]	0	User
cp2102n.portSettings.gpio.GPIO2.suspend.Mode	Mode	1 bit	590[5]	0	User
cp2102n.portSettings.gpio.GPIO3.suspend.Mode	Mode	1 bit	590[6]	0	User
cp2102n.portSettings.gpio.UNUSED.suspend.Mode	Mode	1 bit	590[7]	0	-
[BITFIELD]	main.Suspend Mode P2	1 B	591		-
cp2102n.portSettings.gpio.UNUSED.suspend.Mode	Mode	1 bit	591[0]	0	-
cp2102n.portSettings.gpio.UNUSED.suspend.Mode	Mode	1 bit	591[1]	0	-
cp2102n.portSettings.gpio.UNUSED.suspend.Mode	Mode	1 bit	591[2]	0	-
cp2102n.portSettings.gpio.UNUSED.suspend.Mode	Mode	1 bit	591[3]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	591[4]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	591[5]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	591[6]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	591[7]	0	-
[BITFIELD]	main.Suspend Low Power P0	1 B	592		-
cp2102n.portSettings.gpio.RI.suspend.LowPower	Low Power	1 bit	592[0]	1	User
cp2102n.portSettings.gpio.DCD.suspend.LowPower	Low Power	1 bit	592[1]	1	User
cp2102n.portSettings.gpio.DTR.suspend.LowPower	Low Power	1 bit	592[2]	1	User
cp2102n.portSettings.gpio.DSR.suspend.LowPower	Low Power	1 bit	592[3]	1	User
cp2102n.portSettings.gpio.TXD.suspend.LowPower	Low Power	1 bit	592[4]	1	User
cp2102n.portSettings.gpio.RXD.suspend.LowPower	Low Power	1 bit	592[5]	1	User
cp2102n.portSettings.gpio.RTS.suspend.LowPower	Low Power	1 bit	592[6]	1	User
cp2102n.portSettings.gpio.CTS.suspend.LowPower	Low Power	1 bit	592[7]	1	User
[BITFIELD]	main.Suspend LowPower P1	1 B	593		-
cp2102n.portSettings.gpio.SUSPEND.suspend.LowPower	Low Power	1 bit	593[0]	1	User

Property ID	Label	Length	Offset	Default Value	Type
cp2102n.portSettings.gpio.UNUSED.suspend.Low-Power	Low Power	1 bit	593[1]	1	-
cp2102n.portSettings.gpio.SUSPENDb.suspend.LowPower	Low Power	1 bit	593[2]	1	User
cp2102n.portSettings.gpio.GPIO0.suspend.Low-Power	Low Power	1 bit	593[3]	1	User
cp2102n.portSettings.gpio.GPIO1.suspend.Low-Power	Low Power	1 bit	593[4]	1	User
cp2102n.portSettings.gpio.GPIO2.suspend.Low-Power	Low Power	1 bit	593[5]	1	User
cp2102n.portSettings.gpio.GPIO3.suspend.Low-Power	Low Power	1 bit	593[6]	1	User
cp2102n.portSettings.gpio.UNUSED.suspend.Low-Power	Low Power	1 bit	593[7]	1	-
[BITFIELD]	main.Suspend LowPower P2	1 B	594		-
cp2102n.portSettings.gpio.UNUSED.suspend.Low-Power	Low Power	1 bit	594[0]	1	-
cp2102n.portSettings.gpio.UNUSED.suspend.Low-Power	Low Power	1 bit	594[1]	1	-
cp2102n.portSettings.gpio.UNUSED.suspend.Low-Power	Low Power	1 bit	594[2]	1	-
cp2102n.portSettings.gpio.UNUSED.suspend.Low-Power	Low Power	1 bit	594[3]	1	-
BIT_PADDING_0	Bit Padding 0	1 bit	594[4]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	594[5]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	594[6]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	594[7]	0	-
[BITFIELD]	main.Suspend Latch P0	1 B	595		-
cp2102n.portSettings.gpio.RI.suspend.Latch	Latch	1 bit	595[0]	1	User
cp2102n.portSettings.gpio.DCD.suspend.Latch	Latch	1 bit	595[1]	1	User
cp2102n.portSettings.gpio.DTR.suspend.Latch	Latch	1 bit	595[2]	1	User
cp2102n.portSettings.gpio.DSR.suspend.Latch	Latch	1 bit	595[3]	1	User
cp2102n.portSettings.gpio.TXD.suspend.Latch	Latch	1 bit	595[4]	1	User
cp2102n.portSettings.gpio.RXD.suspend.Latch	Latch	1 bit	595[5]	1	User
cp2102n.portSettings.gpio.RTS.suspend.Latch	Latch	1 bit	595[6]	1	User
cp2102n.portSettings.gpio.CTS.suspend.Latch	Latch	1 bit	595[7]	1	User
[BITFIELD]	main.Suspend Latch P1	1 B	596		-
cp2102n.portSettings.gpio.SUSPEND.suspend.Latch	Latch	1 bit	596[0]	1	User
cp2102n.portSettings.gpio.UNUSED.suspend.Latch	Latch	1 bit	596[1]	1	-

Property ID	Label	Length	Offset	Default Value	Type
cp2102n.portSettings.gpio.SUSPENDb.suspend.Latch	Latch	1 bit	596[2]	0	User
cp2102n.portSettings.gpio.GPIO0.suspend.Latch	Latch	1 bit	596[3]	1	User
cp2102n.portSettings.gpio.GPIO1.suspend.Latch	Latch	1 bit	596[4]	1	User
cp2102n.portSettings.gpio.GPIO2.suspend.Latch	Latch	1 bit	596[5]	1	User
cp2102n.portSettings.gpio.GPIO3.suspend.Latch	Latch	1 bit	596[6]	1	User
cp2102n.portSettings.gpio.UNUSED.suspend.Latch	Latch	1 bit	596[7]	1	-
[BITFIELD]	main.Suspend Latch P2	1 B	597		-
cp2102n.portSettings.gpio.UNUSED.suspend.Latch	Latch	1 bit	597[0]	1	-
cp2102n.portSettings.gpio.UNUSED.suspend.Latch	Latch	1 bit	597[1]	1	-
cp2102n.portSettings.gpio.UNUSED.suspend.Latch	Latch	1 bit	597[2]	1	-
cp2102n.portSettings.gpio.UNUSED.suspend.Latch	Latch	1 bit	597[3]	1	-
BIT_PADDING_0	Bit Padding 0	1 bit	597[4]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	597[5]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	597[6]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	597[7]	0	-
cp2102n.portSettings.control.driveStrength	Drive Strength	1 B	598	0x0F	User
[BITFIELD]	main.GPIO Control 0	1 B	599		-
BIT_PADDING_1	Bit Padding 1	1 bit	599[0]	1	-
BIT_PADDING_1	Bit Padding 1	1 bit	599[1]	1	-
BIT_PADDING_1	Bit Padding 1	1 bit	599[2]	1	-
BIT_PADDING_1	Bit Padding 1	1 bit	599[3]	1	-
BIT_PADDING_1	Bit Padding 1	1 bit	599[4]	1	-
BIT_PADDING_1	Bit Padding 1	1 bit	599[5]	1	-
BIT_PADDING_1	Bit Padding 1	1 bit	599[6]	1	-
BIT_PADDING_1	Bit Padding 1	1 bit	599[7]	1	-
[BITFIELD]	main.GPIO Control 1	1 B	600		-
BIT_PADDING_1	Bit Padding 1	1 bit	600[0]	1	-
BIT_PADDING_1	Bit Padding 1	1 bit	600[1]	1	-
cp2102n.portSettings.control.TXLED	Txled	1 bit	600[2]	0	User
cp2102n.portSettings.control.RXLED	Rxled	1 bit	600[3]	0	User
cp2102n.portSettings.control.RS485	Rs485	1 bit	600[4]	0	User
cp2102n.portSettings.control.WAKEUP	Wakeup	1 bit	600[5]	0	User
cp2102n.portSettings.control.CLK	Clk	1 bit	600[6]	0	User
BIT_PADDING_1	Bit Padding 1	1 bit	600[7]	1	-

Property ID	Label	Length	Offset	Default Value	Type
[BITFIELD]	main.GPIO Control 2	1 B	601		-
cp2102n.portSettings.control.RS485LOGIC	Rs485 Logic	1 bit	601[0]	0	User
cp2102n.portSettings.control.WEAKRESET	Weakreset	1 bit	601[1]	1	User
cp2102n.portSettings.control.WEAKSUSPEND	Weaksuspend	1 bit	601[2]	1	User
cp2102n.portSettings.serial.dynamicSuspend	Dynamic Suspend	1 bit	601[3]	0	User
cp2102n.portSettings.gpio.dynamicSuspend	Dynamic Suspend	1 bit	601[4]	0	User
cp2102n.portSettings.control.CHARGING	Charging	3 bits	601[5]	0	User
[BITFIELD]	main.Flush Buffers Config	1 B	602		-
cp2102n.flushBuffers.UART0.onOpen.TX	Tx	1 bit	602[0]	1	User
cp2102n.flushBuffers.UART0.onOpen.RX	Rx	1 bit	602[1]	1	User
cp2102n.flushBuffers.UART0.onClose.TX	Tx	1 bit	602[2]	0	User
cp2102n.flushBuffers.UART0.onClose.RX	Rx	1 bit	602[3]	0	User
BIT_PADDING_1	Bit Padding 1	1 bit	602[4]	1	-
BIT_PADDING_1	Bit Padding 1	1 bit	602[5]	1	-
BIT_PADDING_0	Bit Padding 0	1 bit	602[6]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	602[7]	0	-
cp2102n.commProp.PacketLength	Packet Length	2 B	603	0x0042	-
cp2102n.commProp.PacketVersion	Packet Version	2 B	605	0x0100	-
cp2102n.commProp.ServiceMask	Service Mask	4 B	607	0x00000001	-
cp2102n.commProp.Reserved1	Reserved1	4 B	611	0x00000000	-
cp2102n.commProp.MaxTxQueue	Max Tx Queue	4 B	615	0x00000280	-
cp2102n.commProp.MaxRxQueue	Max Rx Queue	4 B	619	0x00000280	-
cp2102n.commProp.MaxBaud	Max Baud	4 B	623	0x102DC6C0	User
cp2102n.commProp.ProvSubType	Prov Sub Type	4 B	627	0x00000001	-
cp2102n.commProp.ProvCapabilities	Prov Capabilities	4 B	631	0x0000013F	-
cp2102n.commProp.SettableParams	Settable Params	4 B	635	0x0000007F	-
[BITFIELD]	main.SettableBaud	4 B	639		-
cp2102n.commProp.SettableBaud.SERIAL_BAUD_075	Serial Baud 075	1 bit	639[0]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_110	Serial Baud 110	1 bit	639[1]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_134_5	Serial Baud 134 5	1 bit	639[2]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_150	Serial Baud 150	1 bit	639[3]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_300	Serial Baud 300	1 bit	639[4]	1	User

Property ID	Label	Length	Offset	Default Value	Type
cp2102n.commProp.SettableBaud.SERIAL_BAUD_600	Serial Baud 600	1 bit	639[5]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_1200	Serial Baud 1200	1 bit	639[6]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_1800	Serial Baud 1800	1 bit	639[7]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_2400	Serial Baud 2400	1 bit	639[8]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_4800	Serial Baud 4800	1 bit	639[9]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_7200	Serial Baud 7200	1 bit	639[10]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_9600	Serial Baud 9600	1 bit	639[11]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_14400	Serial Baud 14400	1 bit	639[12]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_19200	Serial Baud 19200	1 bit	639[13]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_38400	Serial Baud 38400	1 bit	639[14]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_56K	Serial Baud 56 K	1 bit	639[15]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_128K	Serial Baud 128 K	1 bit	639[16]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_115200	Serial Baud 115200	1 bit	639[17]	1	User
cp2102n.commProp.SettableBaud.SERIAL_BAUD_57600	Serial Baud 57600	1 bit	639[18]	1	User
BIT_PADDING_0	Bit Padding 0	1 bit	639[19]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	639[20]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	639[21]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	639[22]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	639[23]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	639[24]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	639[25]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	639[26]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	639[27]	0	-
cp2102n.commProp.SettableBaud.SERIAL_BAUD_USER	Serial Baud User	1 bit	639[28]	1	User
BIT_PADDING_0	Bit Padding 0	1 bit	639[29]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	639[30]	0	-

Property ID	Label	Length	Offset	Default Value	Type
BIT_PADDING_0	Bit Padding 0	1 bit	639[31]	0	-
[BITFIELD]	main.settableData	2 B	643		-
cp2102n.commProp.SettableData.SERIAL_DATA-BITS_5	Serial Databits 5	1 bit	643[0]	1	User
cp2102n.commProp.SettableData.SERIAL_DATA-BITS_6	Serial Databits 6	1 bit	643[1]	1	User
cp2102n.commProp.SettableData.SERIAL_DATA-BITS_7	Serial Databits 7	1 bit	643[2]	1	User
cp2102n.commProp.SettableData.SERIAL_DATA-BITS_8	Serial Databits 8	1 bit	643[3]	1	User
BIT_PADDING_0	Bit Padding 0	1 bit	643[4]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	643[5]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	643[6]	0	-
BIT_PADDING_0	Bit Padding 0	1 bit	643[7]	0	-
BIT_PADDING_0	Bit Padding 0	8 bits	643[8]	0	-
[BITFIELD]	main.settableData	2 B	645		-
cp2102n.commProp.SettableStopParity.SERIAL_STOPBITS_1_0	Serial Stopbits 1 0	1 bit	645[0]	1	User
cp2102n.commProp.SettableStopParity.SERIAL_STOPBITS_1_5	Serial Stopbits 1 5	1 bit	645[1]	1	User
cp2102n.commProp.SettableStopParity.SERIAL_STOPBITS_2_0	Serial Stopbits 2 0	1 bit	645[2]	1	User
BIT_PADDING_0	Bit Padding 0	1 bit	645[3]	0	-
BIT_PADDING_0	Bit Padding 0	4 bits	645[4]	0	-
cp2102n.commProp.SettableStopParity.SERIAL_PARITY_NONE	Serial Parity None	1 bit	645[8]	1	User
cp2102n.commProp.SettableStopParity.SERIAL_PARITY_ODD	Serial Parity Odd	1 bit	645[9]	1	User
cp2102n.commProp.SettableStopParity.SERIAL_PARITY_EVEN	Serial Parity Even	1 bit	645[10]	1	User
cp2102n.commProp.SettableStopParity.SERIAL_PARITY_MARK	Serial Parity Mark	1 bit	645[11]	1	User
cp2102n.commProp.SettableStopParity.SERIAL_PARITY_SPACE	Serial Parity Space	1 bit	645[12]	1	User
BIT_PADDING_0	Bit Padding 0	3 bits	645[13]	0	-
cp2102n.commProp.CurrentTxQueue	Current Tx Queue	4 B	647	0x00000280	-
cp2102n.commProp.CurrentRxQueue	Current Rx Queue	4 B	651	0x00000280	-
cp2102n.commProp.ProvSpec1	Prov Spec1	4 B	655	0x00000000	-
cp2102n.commProp.ProvSpec2	Prov Spec2	4 B	659	0x00000000	-
cp2102n.commProp.ProvChar.ProvChar[0]	Prov Char[0]	1 B	663	0x33	-

Property ID	Label	Length	Offset	Default Value	Type
cp2102n.commProp.ProvChar.ProvChar[1]	Prov Char[1]	1 B	664	0x00	-
cp2102n.commProp.ProvChar.ProvChar[2]	Prov Char[2]	1 B	665	0x2E	-
cp2102n.commProp.ProvChar.ProvChar[3]	Prov Char[3]	1 B	666	0x00	-
cp2102n.commProp.ProvChar.ProvChar[4]	Prov Char[4]	1 B	667	0x30	-
cp2102n.commProp.ProvChar.ProvChar[5]	Prov Char[5]	1 B	668	0x00	-
cp2102n.rs485Setup	Rs485 Setup	2 B	669	0x0000	User
cp2102n.rs485Hold	Rs485 Hold	2 B	671	0x0000	User
cp2102n.flowOn	Flow On	1 B	673	0x03	-
cp2102n.flowOff	Flow Off	1 B	674	0x01	-
cp2102n.clockDivider	Clock Divider	1 B	675	0x00	-
FLETCHER CHECKSUM	Fletcher Checksum	2 B	676	0x8335	User

1.1.42 Fletcher Checksum

The last property in [1.1.41 CP2102N Configuration Array 1pbConfig](#) holds a 16-bit "Fletcher" checksum of the other content in the array. This checksum must be recalculated to reflect any changes made elsewhere in the configuration array.

For reference, Wikipedia provides a [C implementation for the Fletcher checksum](#) as follows:

```
// Taken from Wikipedia https://en.wikipedia.org/wiki/Fletcher%27s_checksum
// Must use generic pointer since function is called on XDATA and CODE spaces.
uint16_t fletcher16(uint8_t *dataIn, uint16_t bytes)
{
    uint16_t sum1 = 0xff, sum2 = 0xff;
    uint16_t tlen;

    while (bytes) {
        tlen = bytes >= 20 ? 20 : bytes;
        bytes -= tlen;
        do {
            sum2 += sum1 += *dataIn++;
        } while (--tlen);
        sum1 = (sum1 & 0xff) + (sum1 >> 8);
        sum2 = (sum2 & 0xff) + (sum2 >> 8);
    }
    /* Second reduction step to reduce sums to 8 bits */
    sum1 = (sum1 & 0xff) + (sum1 >> 8);
    sum2 = (sum2 & 0xff) + (sum2 >> 8);
    return sum2 << 8 | sum1;
}
```

1.1.43 CP210x Manufacturing Library Type Definitions and Constants

This section lists type definitions from C/C++ Header File CP210xManufacturingDLL.h. For a complete listing, function prototypes, and other API information for the CP210x Manufacturing DLL, please refer to CP210xManufacturingDLL.h.

```

/// @ref CP210x_GetProductString() function flags
typedef enum _CP210X_RETURN_GETPRODUCTSTRING {
    CP210x_RETURN_SERIAL_NUMBER = 0x00 ///< Serial Number string
    , CP210x_RETURN_DESCRIPTION = 0x01 ///< Description string a la "Friendly Name"
    , CP210x_RETURN_FULL_PATH = 0x02 ///< Full Path string a la "Device Path"
    ///< "A NULL-terminated string that contains the device
    ///< interface path. This path can be passed to Win32
    ///< functions such as CreateFile()."
} CP210X_RETURN_GETPRODUCTSTRING, *PCP210X_RETURN_GETPRODUCTSTRING;

// CP210x_GetPartNumber() returned PartNums (@deprecated {instead see @ref CP210X_PARTNUM})
#define CP210x_CP2101_VERSION CP210x_PARTNUM_CP2101
#define CP210x_CP2102_VERSION CP210x_PARTNUM_CP2102
#define CP210x_CP2103_VERSION CP210x_PARTNUM_CP2103
#define CP210x_CP2104_VERSION CP210x_PARTNUM_CP2104
#define CP210x_CP2105_VERSION CP210x_PARTNUM_CP2105
#define CP210x_CP2108_VERSION CP210x_PARTNUM_CP2108
#define CP210x_CP2109_VERSION CP210x_PARTNUM_CP2109
#define CP210x_CP2102N_QFN28_VERSION CP210x_PARTNUM_CP2102N_QFN28
#define CP210x_CP2102N_QFN24_VERSION CP210x_PARTNUM_CP2102N_QFN24
#define CP210x_CP2102N_QFN20_VERSION CP210x_PARTNUM_CP2102N_QFN20

/// API Status return codes
#define CP210x_SUCCESS SILABS_STATUS_SUCCESS ///< Success
#define CP210x_INVALID_HANDLE 0x01 ///< an handle parameter was not valid
#define CP210x_INVALID_PARAMETER 0x02 ///< a parameter was not valid
#define CP210x_DEVICE_IO_FAILED 0x03 ///< device I/O failed
#define CP210x_FUNCTION_NOT_SUPPORTED 0x04 ///< the specified function is not supported
#define CP210x_GLOBAL_DATA_ERROR 0x05 ///< global data error
#define CP210x_FILE_ERROR 0x06 ///< file error
#define CP210x_COMMAND_FAILED 0x08 ///< command failed
#define CP210x_INVALID_ACCESS_TYPE 0x09 ///< invalid access type
#define CP210x_DEVICE_NOT_FOUND 0xFF ///< the specified device was not found
typedef int CP210x_STATUS;

// Buffer size limits
//
// CP2101/2/3/4/9
#define CP210x_MAX_DEVICE_STRLen 256 ///< CP210x Maximum Device String length
#define CP210x_MAX_MANUFACTURER_STRLen 45

#define CP210x_MAX_PRODUCT_STRLen 126
#define CP210x_MAX_SERIAL_STRLen 63
//
// CP2105
#define CP2105_MAX_MANUFACTURER_STRLen 12
#define CP2105_MAX_PRODUCT_STRLen 47
#define CP2105_MAX_SERIAL_STRLen 16
#define CP2105_MAX_INTERFACE_STRLen 32
//
// CP2108
#define CP2108_MAX_MANUFACTURER_STRLen 126
#define CP2108_MAX_PRODUCT_STRLen 126
#define CP2108_MAX_SERIAL_STRLen 126
#define CP2108_MAX_INTERFACE_STRLen 126
//
// CP2102N
#define CP2102N_MAX_
// Type definitions
typedef char CP210x_DEVICE_STRING[CP210x_MAX_DEVICE_STRLen];
typedef char CP210x_MANUFACTURER_STRING[CP210x_MAX_MANUFACTURER_STRLen];
typedef char CP210x_PRODUCT_STRING[CP210x_MAX_PRODUCT_STRLen];
typedef char CP210x_SERIAL_STRING[CP210x_MAX_SERIAL_STRLen];
typedef char CP2105_INTERFACE_STRING[CP2105_MAX_INTERFACE_STRLen];
typedef char CP2108_INTERFACE_STRING[CP2108_MAX_INTERFACE_STRLen];

```

```

#define      CP210x_MAX_MAXPOWER      250

/// Baud Rate Aliasing definitions
#define      NUM_BAUD_CONFIGS      32
///
/// Baud Config Structure
typedef struct _BAUD_CONFIG
{
    uint16_t    BaudGen;
    uint16_t    Timer0Reload;
    uint8_t     Prescaler;
    uint32_t    BaudRate;
} BAUD_CONFIG, *PBAUD_CONFIG;
///
/// Size of the Baud Config Structure
#define      BAUD_CONFIG_SIZE      10
///
/// Array of all Baud Rate Configurations
typedef      BAUD_CONFIG      BAUD_CONFIG_DATA[NUM_BAUD_CONFIGS];

/// Flush Buffer definitions see @ref CP210x_SetFlushBufferConfig()
/// When these bits are set, dictate when the device will flush that
/// buffer (TX and/or RX) and upon which event (Open and/or Close)

/// CP2104
#define FC_OPEN_TX      0x01      ///< on Open, Tx
#define FC_OPEN_RX      0x02      ///< on Open, Rx
#define FC_CLOSE_TX      0x04      ///< on Close, Tx
#define FC_CLOSE_RX      0x08      ///< on Close, Tx
///
/// CP2105 - Standard Port
#define FC_OPEN_TX_SCI    FC_OPEN_TX    ///< Standard port, on Open, Tx
#define FC_OPEN_RX_SCI    FC_OPEN_RX    ///< Standard port, on Open, Rx
#define FC_CLOSE_TX_SCI    FC_CLOSE_TX    ///< Standard port, on Close, Tx
#define FC_CLOSE_RX_SCI    FC_CLOSE_RX    ///< Standard port, on Close, Rx
///
/// CP2105 - Enhanced Port
#define FC_OPEN_TX_ECI    0x10      ///< Enhanced port, on Open, Tx
#define FC_OPEN_RX_ECI    0x20      ///< Enhanced port, on Open, Tx
#define FC_CLOSE_TX_ECI    0x40      ///< Enhanced port, on Close, Tx
#define FC_CLOSE_RX_ECI    0x80      ///< Enhanced port, on Close, Rx
///
/// CP2108
#define FC_OPEN_TX_IFC0    0x0001    ///< CP2108 port 0, on Open, Tx
#define FC_OPEN_RX_IFC0    0x0002    ///< CP2108 port 0, on Open, Rx
#define FC_CLOSE_TX_IFC0    0x0004    ///< CP2108 port 0, on Close, Tx
#define FC_CLOSE_RX_IFC0    0x0008    ///< CP2108 port 0, on Close, Rx
#define FC_OPEN_TX_IFC1    0x0010    ///< CP2108 port 1, on Open, Tx
#define FC_OPEN_RX_IFC1    0x0020    ///< CP2108 port 1, on Open, Rx
#define FC_CLOSE_TX_IFC1    0x0040    ///< CP2108 port 1, on Close, Tx
#define FC_CLOSE_RX_IFC1    0x0080    ///< CP2108 port 1, on Close, Rx
#define FC_OPEN_TX_IFC2    0x0100    ///< CP2108 port 2, on Open, Tx
#define FC_OPEN_RX_IFC2    0x0200    ///< CP2108 port 2, on Open, Rx
#define FC_CLOSE_TX_IFC2    0x0400    ///< CP2108 port 2, on Close, Tx
#define FC_CLOSE_RX_IFC2    0x0800    ///< CP2108 port 2, on Close, Rx
#define FC_OPEN_TX_IFC3    0x1000    ///< CP2108 port 3, on Open, Tx
#define FC_OPEN_RX_IFC3    0x2000    ///< CP2108 port 3, on Open, Rx
#define FC_CLOSE_TX_IFC3    0x4000    ///< CP2108 port 3, on Close, Tx
#define FC_CLOSE_RX_IFC3    0x8000    ///< CP2108 port 3, on Close, Rx

/// Port Config definitions
/// CP2103/4 Port Config Structure
typedef struct _PORT_CONFIG
{
    uint16_t    Mode;                // Push-Pull = 1, Open-Drain = 0
    uint16_t    Reset_Latch;         // Logic High = 1, Logic Low = 0
    uint16_t    Suspend_Latch;       // Logic High = 1, Logic Low = 0
    unsigned char EnhancedFxn;
} PORT_CONFIG, *PPORT_CONFIG;
///
/// Define bit locations for Mode/Latch for Reset and Suspend structures

```

```
#define PORT_RI_ON          0x0001
#define PORT_DCD_ON        0x0002
#define PORT_DTR_ON        0x0004
#define PORT_DSR_ON        0x0008
#define PORT_TXD_ON        0x0010
#define PORT_RXD_ON        0x0020
#define PORT_RTS_ON        0x0040
#define PORT_CTS_ON        0x0080
//
#define PORT_GPIO_0_ON      0x0100
#define PORT_GPIO_1_ON      0x0200
#define PORT_GPIO_2_ON      0x0400
#define PORT_GPIO_3_ON      0x0800
//
#define PORT_SUSPEND_ON     0x4000 // Can't configure latch value
#define PORT_SUSPEND_BAR_ON 0x8000 // Can't configure latch value
//
// Define bit locations for EnhancedFxn
#define EF_GPIO_0_TXLED      0x01 // Under device control
#define EF_GPIO_1_RXLED      0x02 // Under device control
#define EF_GPIO_2_RS485      0x04 // Under device control
#define EF_RS485_INVERT      0x08 // RS485 Invert bit
#define EF_WEAKPULLUP        0x10 // Weak Pull-up on
#define EF_RESERVED_1        0x20 // Reserved, leave bit 5 cleared
#define EF_SERIAL_DYNAMIC_SUSPEND 0x40 // For 8 UART/Modem signals
#define EF_GPIO_DYNAMIC_SUSPEND 0x80 // For 4 GPIO signals

/// Dual Port Config definitions
/// CP2105 Dual Port Config Structure
typedef struct _DUAL_PORT_CONFIG
{
    uint16_t Mode; // Push-Pull = 1, Open-Drain = 0
    uint16_t Reset_Latch; // Logic High = 1, Logic Low = 0
    uint16_t Suspend_Latch; // Logic High = 1, Logic Low = 0
    unsigned char EnhancedFxn_ECI;
    unsigned char EnhancedFxn_SCI;
    unsigned char EnhancedFxn_Device;
} DUAL_PORT_CONFIG, *PDUAL_PORT_CONFIG;
//
// CP2105 Define bit locations for Mode/Latch for Reset and Suspend structures
#define PORT_RI_SCI_ON      0x0001
#define PORT_DCD_SCI_ON     0x0002
#define PORT_DTR_SCI_ON     0x0004
#define PORT_DSR_SCI_ON     0x0008
#define PORT_TXD_SCI_ON     0x0010
#define PORT_RXD_SCI_ON     0x0020
#define PORT_RTS_SCI_ON     0x0040
#define PORT_CTS_SCI_ON     0x0080
#define PORT_GPIO_0_SCI_ON  0x0002
#define PORT_GPIO_1_SCI_ON  0x0004
#define PORT_GPIO_2_SCI_ON  0x0008
#define PORT_SUSPEND_SCI_ON 0x0001 // Can't configure latch value
//
#define PORT_RI_ECI_ON      0x0100
#define PORT_DCD_ECI_ON     0x0200
#define PORT_DTR_ECI_ON     0x0400
#define PORT_DSR_ECI_ON     0x0800
#define PORT_TXD_ECI_ON     0x1000
#define PORT_RXD_ECI_ON     0x2000
#define PORT_RTS_ECI_ON     0x4000
#define PORT_CTS_ECI_ON     0x8000
#define PORT_GPIO_0_ECI_ON  0x0400
#define PORT_GPIO_1_ECI_ON  0x0800
#define PORT_SUSPEND_ECI_ON 0x0100 // Can't configure latch value
//
// CP2105 Define bit locations for EnhancedFxn_ECI
#define EF_GPIO_0_TXLED_ECI 0x01 // Under device control
#define EF_GPIO_1_RXLED_ECI 0x02 // Under device control
#define EF_GPIO_1_RS485_ECI 0x04 // Under device control
#define EF_RS485_INVERT     0x08 // Under device control
#define EF_INVERT_SUSPEND_ECI 0x10 // RS485 Invert bit
#define EF_DYNAMIC_SUSPEND_ECI 0x40 // For GPIO signals
```

```
//
// CP2105 Define bit locations for EnhancedFxn_SCI
#define EF_GPIO_0_TXLED_SCI      0x01    // Under device control
#define EF_GPIO_1_RXLED_SCI      0x02    // Under device control
#define EF_INVERT_SUSPEND_SCI     0x10    // RS485 Invert bit
#define EF_DYNAMIC_SUSPEND_SCI    0x40    // For GPIO signals
//
// CP2105 Define bit locations for EnhancedFxn_Device
#define EF_WEAKPULLUP             0x10    // Weak Pull-up on

/// Quad Port Config definitions
/// CP2108 Quad Port State structure (used in Quad Port Config structure)
typedef struct _QUAD_PORT_STATE
{
    uint16_t Mode_PB0;
    uint16_t Mode_PB1;
    uint16_t Mode_PB2;
    uint16_t Mode_PB3;
    uint16_t Mode_PB4;

    uint16_t LowPower_PB0;
    uint16_t LowPower_PB1;
    uint16_t LowPower_PB2;
    uint16_t LowPower_PB3;
    uint16_t LowPower_PB4;

    uint16_t Latch_PB0;
    uint16_t Latch_PB1;
    uint16_t Latch_PB2;
    uint16_t Latch_PB3;
    uint16_t Latch_PB4;
} QUAD_PORT_STATE, *PQUAD_PORT_STATE;

/// CP2108 Quad Port Config structure
typedef struct _QUAD_PORT_CONFIG
{
    QUAD_PORT_STATE Reset_Latch;
    QUAD_PORT_STATE Suspend_Latch;
    uint8_t IPDelay_IFC0;
    uint8_t IPDelay_IFC1;
    uint8_t IPDelay_IFC2;
    uint8_t IPDelay_IFC3;
    uint8_t EnhancedFxn_IFC0;
    uint8_t EnhancedFxn_IFC1;
    uint8_t EnhancedFxn_IFC2;
    uint8_t EnhancedFxn_IFC3;
    uint8_t EnhancedFxn_Device;
    uint8_t ExtClk0Freq;
    uint8_t ExtClk1Freq;
    uint8_t ExtClk2Freq;
    uint8_t ExtClk3Freq;
} QUAD_PORT_CONFIG, *PQUAD_PORT_CONFIG;

// CP2108 Define bit locations for Mode/Latch for Reset and Suspend structures
// PB0
#define PORT_TX0      0x0001
#define PORT_RX0      0x0002
#define PORT_RTS0     0x0004
#define PORT_CTS0     0x0008
#define PORT_DTR0     0x0010
#define PORT_DSR0     0x0020
#define PORT_DCD0     0x0040
#define PORT_RI0      0x0080
#define PORT_TX1      0x0100
#define PORT_RX1      0x0200
#define PORT_RTS1     0x0400
#define PORT_CTS1     0x0800
#define PORT_DTR1     0x1000
#define PORT_DSR1     0x2000
#define PORT_DCD1     0x4000
#define PORT_RI1      0x8000
```

```
// PB1
#define PORT_GPIO_0    0x0001 // (1<=0)
#define PORT_GPIO_1    0x0002 // (1<=1)
#define PORT_GPIO_2    0x0004 // (1<=2)
#define PORT_GPIO_3    0x0008 // etc.
#define PORT_GPIO_4    0x0010
#define PORT_GPIO_5    0x0020
#define PORT_GPIO_6    0x0040
#define PORT_GPIO_7    0x0080
#define PORT_GPIO_8    0x0100
#define PORT_GPIO_9    0x0200
#define PORT_GPIO_10   0x0400
#define PORT_GPIO_11   0x0800
#define PORT_GPIO_12   0x1000
#define PORT_GPIO_13   0x2000
#define PORT_GPIO_14   0x4000
#define PORT_GPIO_15   0x8000

// PB2
#define PORT_SUSPEND    0x0001
#define PORT_SUSPEND_BAR 0x0002
#define PORT_DTR2       0x0004
#define PORT_DSR2       0x0008

// PB3
#define PORT_TX2        0x0001
#define PORT_RX2        0x0002
#define PORT_RTS2       0x0004
#define PORT_CTS2       0x0008
#define PORT_DCD2       0x0010
#define PORT_RI2        0x0020
#define PORT_DTR3       0x0040
#define PORT_DSR3       0x0080
#define PORT_DCD3       0x0100
#define PORT_RI3        0x0200

// PB4
#define PORT_RTS3       0x0001
#define PORT_CTS3       0x0002
#define PORT_TX3        0x0004
#define PORT_RX3        0x0008

//
// CP2108 Define bit locations for EnhancedFxn_IFCx
#define EF_IFC_GPIO_TXLED 0x01
#define EF_IFC_GPIO_RXLED 0x02
#define EF_IFC_GPIO_RS485 0x04
// If the next bit is clear, GPIO1 is low while sending UART data.
// If it is set, GPIO1 is high while sending UART data, and low otherwise
#define EF_IFC_GPIO_RS485_LOGIC 0x08
#define EF_IFC_GPIO_CLOCK      0x10
#define EF_IFC_DYNAMIC_SUSPEND 0x40

//
// CP2108 Define bit locations for EnhancedFxn_Device
#define EF_DEVICE_WEAKPULLUP_RESET 0x10
#define EF_DEVICE_WEAKPULLUP_SUSPEND 0x20
#define EF_DEVICE_DYNAMIC_SUSPEND 0x40
```

1.2 CP210xRuntime.DLL

The CP210x Runtime API provides access to the GPIO port latch, and is meant for distribution with the product containing a CP210x device.

CP210xRT_GetNumDevices()	Determines the number of CP210x devices connected to the system.
CP210xRT_Open()	Opens and returns a handle to a device using a device index determined by the number returned from CP210xRT_GetNumDevices().
CP210xRT_Close()	Closes an open handle to the device.
CP210xRT_ReadLatch()	Returns the GPIO port latch of a CP210x device.
CP210xRT_WriteLatch()	Sets the GPIO port latch of a CP210x device.
CP210xRT_GetPartNumber()	Returns the 1-byte Part Number of a CP210x device.
CP210xRT_GetProductString()	Returns the product string programmed to the device.
CP210xRT_GetDeviceSerialNumber()	Returns the serial number programmed to the device.
CP210xRT_GetDeviceInterfaceString()	Returns the interface string programmed to the device.
CP210xRT_GetReceiverMaxTimeout()	Reads and returns the Receiver Max Timeout directly from the device.
CP210xRT_SetReceiverMaxTimeout()	Sets the Receiver Max Timeout directly to the device.

Typically, the user initiates communication with the target CP210x device by opening a handle to a COM port using `CreateFile()` (See *AN197: Serial Communication Guide for CP210x*). The handle returned allows the user to call the API functions listed above. Each of these functions are described in the following sections. Type definitions and constants are defined in the file `CP210xRuntimeDLL.h`.

Note: Functions calls into this API are blocked until completed. This can take several milliseconds depending on USB traffic.

1.2.1 CP210xRT_GetNumDevices

Description : Determines the number of CP210x devices connected to the system.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : `CP210x_STATUS CP210xRT_GetNumDevices(uint32_t *lpdwNumDevices)`

Parameters : 1. `lpdwNumDevices`—a pointer to a `uint32_t` location to hold the returned device count

Return Value : `CP210x_STATUS`

- `CP210x_SUCCESS`
- `CP210x_INVALID_PARAMETER`

1.2.2 CP210xRT_Open

Description : Opens and returns a handle to a device using a device index determined by the number returned from `CP210xRT_GetNumDevices()`.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : `CP210x_STATUS CP210xRT_Open(uint32_t DeviceIndex, PHANDLE pcyHandle)`

Parameters :

1. `DeviceIndex`—the desired device's index into the device list determined by `CP210xRT_GetNumDevices()`
2. `pcyHandle`—a pointer to a `HANDLE` (4-byte location to hold the returned open handle to the device)

Return Value : `CP210x_STATUS`

- `CP210x_SUCCESS`
- `CP210x_INVALID_HANDLE`
- `CP210x_DEVICE_NOT_FOUND`

1.2.3 CP210xRT_Close

Description : Closes an open handle to the device.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210xRT_Close(HANDLE cyHandle)

Parameters : 1. cyHandle—Handle to the Com port returned by CreateFile().

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE

1.2.4 CP210xRT_ReadLatch

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

Supported Devices : CP2102N, CP2103, CP2104, CP2105, CP2108

Prototype : CP210x_STATUS CP210xRT_ReadLatch(HANDLE cyHandle, LPWORD lpwLatch)

Parameters : 1. cyHandle—Handle to the Com port returned by CreateFile().
2. lpwLatch—Pointer for 2-byte return GPIO latch values [Logic High = 1, Logic Low = 0].

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_INVALID_PARAMETER
- CP210x_DEVICE_NOT_FOUND
- CP210x_DEVICE_IO_FAILED
- CP210x_FUNCTION_NOT_SUPPORTED

1.2.5 CP210xRT_WriteLatch

Description : Sets the current port latch values for the device.

Supported Devices : CP2102N, CP2103, CP2104, CP2105, CP2108

Prototype : CP210x_STATUS CP210xRT_WriteLatch(HANDLE cyHandle, WORD mask, WORD latch)

Parameters : 1. cyHandle—Handle to the Com port returned by CreateFile().
2. mask—Determines which pins to change [Change = 1, Leave = 0].
3. latch— values to write to the masked GPIO latches [Logic High = 1, Logic Low = 0].

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_NOT_FOUND
- CP210x_DEVICE_IO_FAILED
- CP210x_FUNCTION_NOT_SUPPORTED

1.2.6 CP210xRT_GetPartNumber

Description : Gets the part number of the current device.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210xRT_GetPartNumber(HANDLE cyHandle, LPBYTE lpbPartNum)

Parameters :

1. cyHandle—Handle to the Com port returned by CreateFile().
2. lpbPartNum—Pointer to a byte containing the return code for the part number.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_INVALID_PARAMETER
- CP210x_DEVICE_IO_FAILED

1.2.7 CP210xRT_GetDeviceProductString

Description : Gets the product string in the current device.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210xRT_GetDeviceProductString(HANDLE cyHandle, LPVOID lpProductString, LPBYTE lpbProductStringLengthInBytes, BOOL bConvertToASCII = TRUE)

Parameters :

1. cyHandle—Handle to the Com port returned by CreateFile().
2. lpProductString—Variable of type CP210x_PRODUCT_STRING returning the NULL terminated product string.
3. lpbProductStringLengthInBytes—Length in characters (not bytes) not including a NULL terminator.
4. bConvertToASCII—Boolean that determines whether the string should be left in Unicode, or converted to ASCII. This parameter is true by default, and will convert to ASCII.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED
- CP210x_INVALID_PARAMETER

1.2.8 CP210xRT_GetDeviceSerialNumber

Description : Gets the product string in the current device.

Supported Devices : CP2101, CP2102, CP2102N, CP2103, CP2104, CP2105, CP2108, CP2109

Prototype : CP210x_STATUS CP210xRT_GetDeviceSerialNumber(HANDLE cyHandle, LPVOID lpSerialNumberString, LPBYTE lpbSerialNumberStringLengthInBytes, BOOL bConvertToASCII = TRUE)

Parameters :

1. cyHandle—Handle to the Com port returned by CreateFile().
2. lpSerialNumberString—Variable of type CP210x_SERIAL_STRING returning the NULL terminated serial string.
3. lpbSerialNumberStringLengthInBytes—Length in characters (not bytes) not including a NULL terminator.
4. bConvertToASCII—Boolean that determines whether the string should be left in Unicode, or converted to ASCII. This parameter is true by default, and will convert to ASCII.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED
- CP210x_INVALID_PARAMETER

1.2.9 CP210xRT_GetDeviceInterfaceString

Description : Gets the interface string of the current device.

Supported Devices : CP2105, CP2108

Prototype : CP210x_STATUS CP210xRT_GetDeviceInterfaceString(HANDLE cyHandle, LPVOID lpInterfaceString, LPBYTE lpbInterfaceStringLengthInBytes, BOOL bConvertToASCII)

Parameters :

1. cyHandle—Handle to the Com port returned by CreateFile().
2. lpInterfaceString—Variable of type CP210x_SERIAL_STRING returning the NULL terminated serial string.
3. lpbInterfaceStringLengthInBytes—Length in characters (not bytes) not including a NULL terminator.
4. bConvertToASCII—Boolean that determines whether the string should be left in Unicode, or converted to ASCII. This parameter is true by default, and will convert to ASCII.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_DEVICE_IO_FAILED
- CP210x_INVALID_PARAMETER

1.2.10 CP210xRT_GetReceiverMaxTimeout

Description : Reads and returns the Receiver Max Timeout directly from the device.

Supported Devices : CP2102N

Prototype : CP210x_STATUS CP210xRT_GetReceiverMaxTimeout(HANDLE cyHandle, uint16_t *pReceiverMaxTimeoutInMicroseconds)

Parameters :

1. cyHandle—Handle to the Com port returned by CreateFile().
2. pReceiverMaxTimeoutInMicroseconds - Pointer to a buffer into which the current Receiver Max Timeout will be written.

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_INVALID_PARAMETER
- CP210x_DEVICE_IO_FAILED

1.2.11 CP210xRT_SetReceiverMaxTimeout

Description : Sets the Receiver Max Timeout directly to the device.

Supported Devices : CP2102N

Prototype : CP210x_STATUS CP210xRT_SetReceiverMaxTimeout(HANDLE cyHandle, uint16_t ReceiverMaxTimeoutInMicroseconds)

Parameters :

1. cyHandle—Handle to the Com port returned by CreateFile().
2. ReceiverMaxTimeoutInMicroseconds - New Receiver Timeout for the device to use

Return Value : CP210x_STATUS

- CP210x_SUCCESS
- CP210x_INVALID_HANDLE
- CP210x_INVALID_PARAMETER
- CP210x_DEVICE_IO_FAILED

1.2.12 CP210x Runtime Library Type Definitions and Constants

This section lists type definitions from C/C++ Header File CP210xRuntimeDLL.h. For a complete listing, function prototypes, and other API information for the CP210x Runtime DLL, please refer to CP210xRuntimeDLL.h.

```
// CP210xRT_GetPartNumber() returned PartNums (@deprecated {instead see silabs_defs.h::CP210X_PARTNUM})
#define CP210x_CP2101_VERSION CP210x_PARTNUM_CP2101
#define CP210x_CP2102_VERSION CP210x_PARTNUM_CP2102
#define CP210x_CP2103_VERSION CP210x_PARTNUM_CP2103
#define CP210x_CP2104_VERSION CP210x_PARTNUM_CP2104
#define CP210x_CP2105_VERSION CP210x_PARTNUM_CP2105
#define CP210x_CP2108_VERSION CP210x_PARTNUM_CP2108
#define CP210x_CP2109_VERSION CP210x_PARTNUM_CP2109
#define CP210x_CP2102N_QFN28_VERSION CP210x_PARTNUM_CP2102N_QFN28
#define CP210x_CP2102N_QFN24_VERSION CP210x_PARTNUM_CP2102N_QFN24
#define CP210x_CP2102N_QFN20_VERSION CP210x_PARTNUM_CP2102N_QFN20

// API Status return codes
#define CP210x_SUCCESS SILABS_STATUS_SUCCESS ///< Success
#define CP210x_INVALID_HANDLE 0x01 ///< an handle parameter was not valid
#define CP210x_INVALID_PARAMETER 0x02 ///< a parameter was not valid
#define CP210x_DEVICE_IO_FAILED 0x03 ///< device I/O failed
#define CP210x_FUNCTION_NOT_SUPPORTED 0x04 ///< the specified function is not supported
#define CP210x_GLOBAL_DATA_ERROR 0x05 ///< global data error
#define CP210x_FILE_ERROR 0x06 ///< file error
#define CP210x_COMMAND_FAILED 0x08 ///< command failed
#define CP210x_INVALID_ACCESS_TYPE 0x09 ///< invalid access type
#define CP210x_DEVICE_NOT_FOUND 0xFF ///< the specified device was not found
typedef int CP210x_STATUS;

// Buffer size limits
#define CP210x_MAX_PRODUCT_STRLEN 126
#define CP210x_MAX_SERIAL_STRLEN 63

// Type definitions
typedef char CP210x_PRODUCT_STRING[CP210x_MAX_PRODUCT_STRLEN];
typedef char CP210x_SERIAL_STRING[CP210x_MAX_SERIAL_STRLEN];

// Mask and Latch value bit definitions
#define CP210x_GPIO_0 0x0001
#define CP210x_GPIO_1 0x0002
#define CP210x_GPIO_2 0x0004
#define CP210x_GPIO_3 0x0008
#define CP210x_GPIO_4 0x0010
#define CP210x_GPIO_5 0x0020
#define CP210x_GPIO_6 0x0040
#define CP210x_GPIO_7 0x0080
#define CP210x_GPIO_8 0x0100
#define CP210x_GPIO_9 0x0200
#define CP210x_GPIO_10 0x0400
#define CP210x_GPIO_11 0x0800
#define CP210x_GPIO_12 0x1000
#define CP210x_GPIO_13 0x2000
#define CP210x_GPIO_14 0x4000
#define CP210x_GPIO_15 0x8000
```

2. Revision History

Revision 1.3

August, 2020

- Updated function parameter data types throughout.
- Updated API function descriptions to match latest CP210xManufacturingDLL.h and CP210xRuntimeDLL.h files.
- Added section [1.1.43 CP210x Manufacturing Library Type Definitions and Constants](#).
- Updated and corrected function prototypes for [1.1.22 CP210x_SetConfig](#) and [1.1.39 CP210x_GetConfig](#).
- Added section [1.2.1 CP210xRT_GetNumDevices](#)
- Added section [1.2.2 CP210xRT_Open](#)
- Added section [1.2.3 CP210xRT_Close](#)
- Added section [1.2.10 CP210xRT_GetReceiverMaxTimeout](#)
- Added section [1.2.11 CP210xRT_SetReceiverMaxTimeout](#)
- Added section [1.2.12 CP210x Runtime Library Type Definitions and Constants](#)
- Updated table [1.2 CP210xRuntime.DLL](#) with functions CP210xRT_GetNumDevices(), CP210xRT_Open(), CP210xRT_Close(), CP210xRT_GetReceiverMaxTimeout, and CP210xRT_SetReceiverMaxTimeout

Revision 1.2

July, 2019

- Corrected CP210xRT_ReadLatch and CP210xRT_WriteLatch parameter descriptions (latch and lpLatch are 2-byte values).

Revision 1.1

July, 2018

- Added section [1.1.41 CP2102N Configuration Array lpbConfig](#).
- Added section [1.1.42 Fletcher Checksum](#).

Revision 1.0

April, 2016

- Initial revision.

Silicon Labs

Simplicity Studio™4



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!



IoT Portfolio
www.silabs.com/IoT



SW/HW
www.silabs.com/simplicity



Quality
www.silabs.com/quality



Support and Community
community.silabs.com

Disclaimer

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

Trademark Information

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



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

<http://www.silabs.com>