# AN144

# CP210x/CP211x Device Customization Guide

**Relevant Devices**
This application note applies to the following devices:
CP2101, CP2102, CP2103, CP2104, CP2105, CP2110, CP2112

## 1. Introduction

This document is intended for developers creating products based on the CP210x/CP211x USB Bridge Controllers. It provides information about obtaining a Vendor ID (VID) and Product ID (PID) for a CP210x/CP211x product and describes the steps necessary for customizing the device descriptors contained in the CP210x/CP211x programmable area. Refer to www.silabs.com for the latest revisions of this document and other application notes related to the CP210x/CP211x device families.

## 2. Obtaining a VID and PID

Each device on a USB bus must have a unique VID, PID, and Serial Number combination. Vendor IDs are owned by the vendor company and assigned by the USB Implementers Forum (USB-IF) only. Details about obtaining a unique VID can be found at www.usb.org/developers/vendor.

To obtain the right to license the USB-IF logo, you must register your product's VID and PID with USB-IF and submit your product to the USB-IF Compliance Program. USB-IF Compliance Program details are available at www.usb.org/developers/compliance. Once your product is certified, you can add it to the USB-IF Integrators List, and you can also use the “Certified USB” logo on your product.

If you do not wish to license the USB-IF logo for your product, you can use the default Silicon Laboratories VID along with a unique PID. To obtain a unique PID for your CP210x/CP211x-based product, visit http://www.silabs.com/RequestPID. Having a unique PID reduces the chances that another device with the same VID, PID, and Serial Number will appear on the same USB bus.

## 3. Customizing Driver Installations, CP210x Custom Setup Utility (CP2101/2/3/4/5)

The driver installation is customizable by modifying certain sections of the hardware installation files (.inf). The strings contained in the .inf files affect what is displayed in the “Found New Hardware Wizard” dialogs, Device Manager, and the Registry. Refer to “AN220: USB Driver Customization” for more details.

**Note:** Any changes to the Windows® installation .inf files will require new Windows Hardware Quality Labs (WHQL) tests.

# 4. Customizing Device Descriptors and Other Configurable Options

The descriptors and other configurable options of the devices in the CP210x/CP211x families are modifiable using the following Windows programs.

**Table 1. Device Customization Programs**

| Device | Program |
|---|---|
| CP2101/2/3/4/5 | CP210xSetIDs.exe |
| CP2110 | CP2110SetIDs.exe |
| CP2112 | CP2112SetIDs.exe |

These programs are included in the software zip file in this application note. How to use these programs is described in more detail in the following sections.

*CP210xSetIDs* uses the Windows Host API functions implemented by *CP210xManufacturing.DLL*. The Host API functions give read/write access to the descriptors contained in programmable areas of a connected device. Another option is implementing a custom application using the Host API and *CP210xManufacturing.DLL* suited to the individual needs of a particular production environment.

Both *CP2110SetIDs* and *CP2112SetIDs* use the SLABHIDDevice.dll and the customization interface specified in their respective application notes, “AN433: CP2110 HID to UART API Specification” and “AN496: CP2112 HID USB to SMBus API Specification”.

The descriptors can also be set in the factory at production time for large orders. Contact your Silicon Laboratories sales representative for details.

## 4.1. Customizing Device Descriptors Using CP210xSetIDs.exe (CP2101/2/3/4/5)

*CP210xSetIDs.exe* is an example program that uses the CP210x Host API Functions implemented by *CP210xManufacturing.DLL*. The program window is shown in Figure 1. *CP210xSetIDs.exe* demonstrates the method of accessing and changing the descriptors contained in the connected device's programmable area. To use *CP210xSetIDs.exe*, it is necessary to have the devices connected and have the unmodified device drivers that shipped with the original CP210x kit installed. The customized device driver installation files that contain the VID and PID values should also be installed. For information on creating customized device drivers for the CP210x, refer to “AN220: USB Driver Customization.” The default driver must be installed so that the CP210x device with the default factory settings appears in the device list. The customized drivers will be needed after the device IDs have been changed and the CP210x device is reset.

All CP210x devices are programmed in the same manner using the CP210xSetIDs.exe tool, but each parameter on the One-Time-Programmable (OTP) devices can only be programmed one time. The CP2101, CP2102, and CP2103 parameters can be reprogrammed multiple times using the CP210xSetIDs.exe utility, but the CP2104 and CP2105 parameters can only be programmed once using the CP210xSetIDs.exe utility. After the CP2104 and CP2105 have been programmed once, the CP210xSetIDs.exe utility can be run, but the new values will not be programmed into the device.

Before running *CP210xSetIDs.exe*, copy *CP210xManufacturing.DLL* into the \windows, \winnt, or \system32 directory, the directory containing the executable, or any directory in the “Path” environment variable.

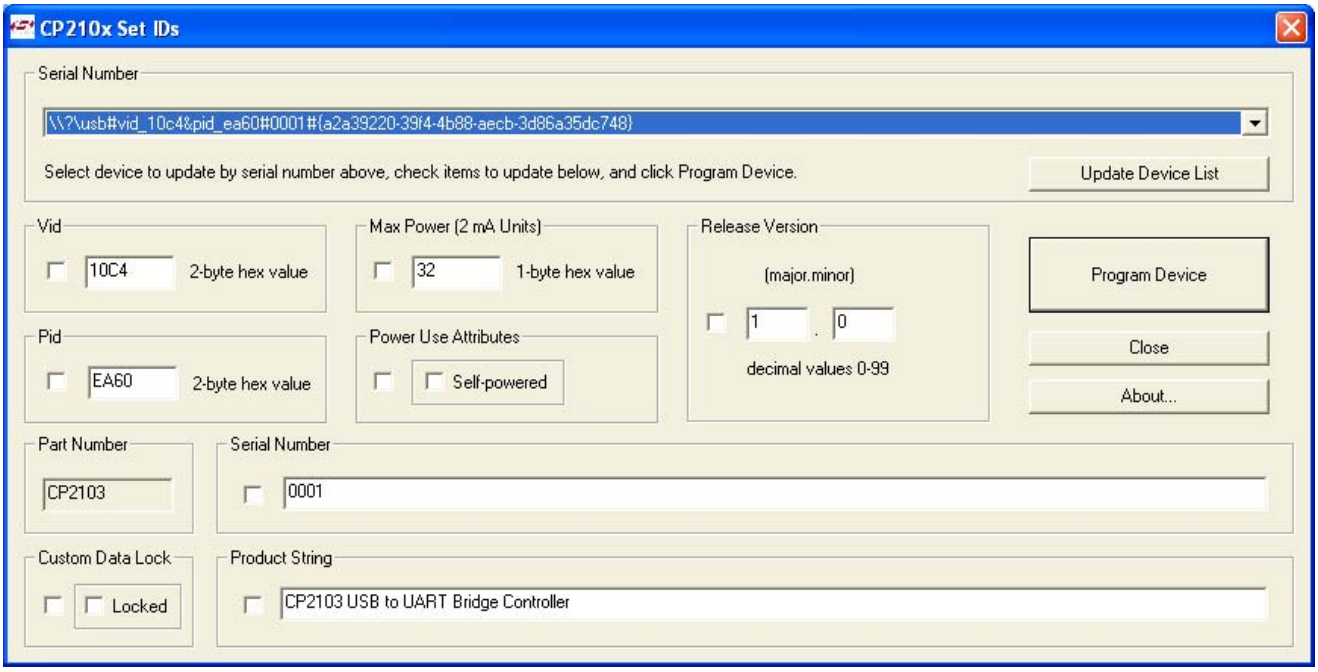


**Figure 1. CP210xSetIDs Example Application**

When *CP210xSetIDs.exe* is launched, the program searches the Windows registry for any CP210x devices attached to the PC. The full path information for all of the devices found is inserted into the “Select Device” drop-down list, and the first device is selected automatically. *CP210xSetIDs.exe* then queries information from the selected device using the CP210x Host API functions and fills in the values for each of the seven selectable fields of the application. When a new device is selected from the list, the fields will be updated with information from the most recently selected device.

To change one or more of the values, click on the checkbox to the left of the field and enter the new value. Once the new values have been entered, click on the “Program Device” button.

The values entered are subject to the following limitations:

1. **VID**—Four hexadecimal digits.
2. **PID**—Four hexadecimal digits.
3. **Max. Power**—Two hexadecimal digits with maximum setting of 250 (the value is in 2 mA units).
4. **Serial Number**—Any sequence of up to 63 characters.
5. **Product String**—Any sequence of up to 126 characters.
6. **Release Version**—Each field is a decimal number value 0–99.

**Notes:**

1. Avoid connecting more than one device containing the same VID, PID, and serial number combination.
2. When the serial number of a CP210x device is changed and the device is reset by calling the Host API function *CP210x_Reset()*, the device will reenumerate, and the device driver will be installed.
3. The serial number and product string are automatically converted to Unicode strings before programming.

## 4.2. Customizing Device Descriptors using *CP2110SetIDs.exe* (CP2110)

*CP2110SetIDs.exe* is an example program that uses the CP2110 API Functions implemented by *SLABHIDtoUART.dll*. The program window is shown in Figure 2. *CP2110SetIDs.exe* demonstrates the method of accessing and changing the descriptors contained in the connected device's programmable area.

The various customizable fields of the CP2110 devices are only programmable one time using the program. After the fields are programmed once, *CP2110SetIDs* can access the fields, but it is not able to reprogram them.

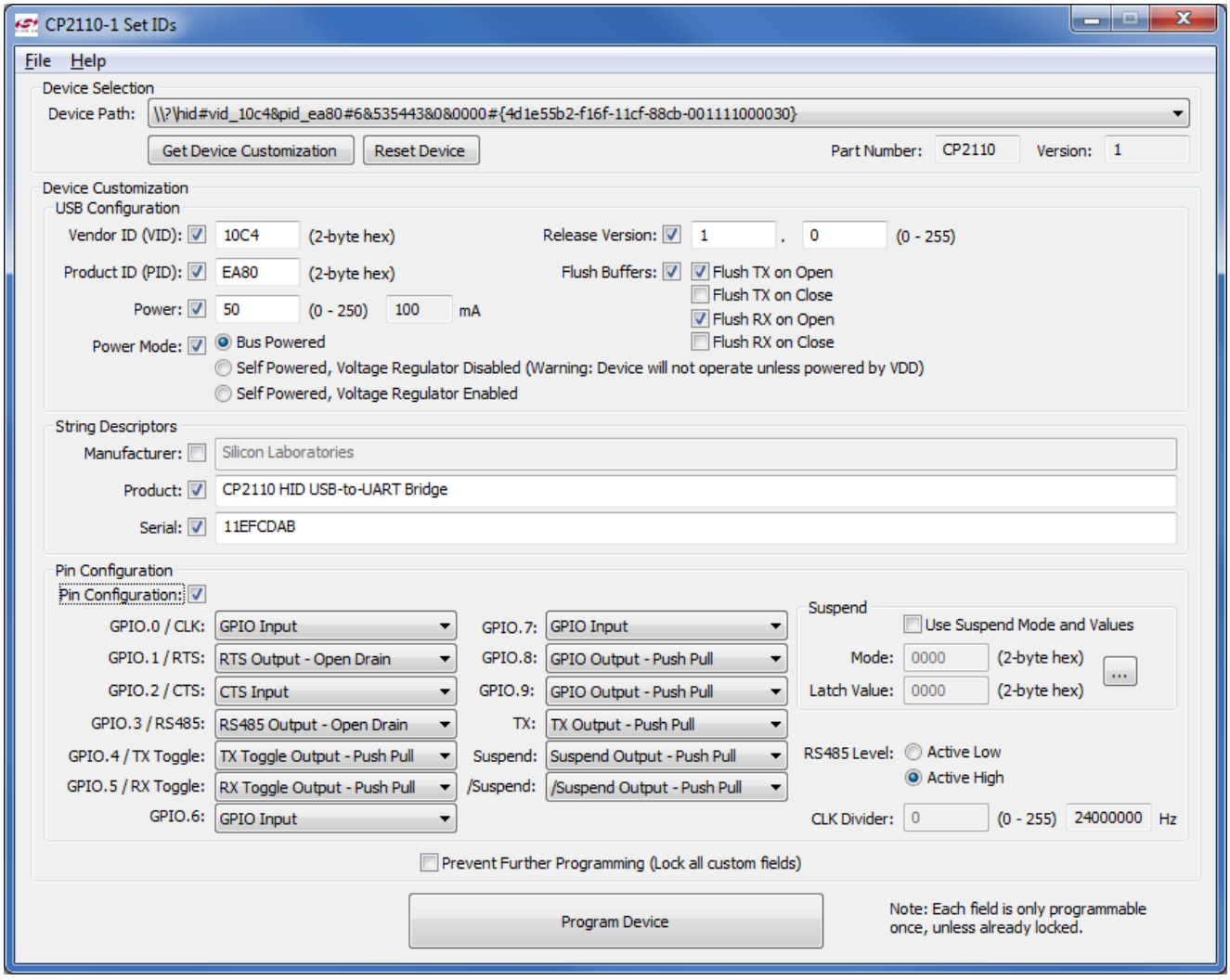


**Figure 2. CP2110SetIDs Application**

When *CP2110SetIDs.exe* is launched, the program searches for any CP2110 devices attached to the PC. The full path information for all of the devices found is inserted into the "Device Path" drop-down list, and the first device is selected automatically. The application then queries information from the selected device using the API functions and fills in the values for each of the selectable fields of the application. When a new device is selected from the list, the fields are updated with information from the most recently selected device.

To change one or more of the values, click on the checkbox to the left of the field and enter the new value. Once the new values are entered, click on the "Program Device" button. Once a field is programmed, the checkbox is no longer selectable, indicating that the field is locked from further programming.

The values entered are subject to the following limitations:

1. **VID**—2-byte value in hexadecimal.
2. **PID**—2-byte value hexadecimal.
3. **Max. Power**—1-byte value in hexadecimal digits with maximum setting of 0xFA (the value is in 2 mA units).
4. **Release Version**—Each field is a decimal number value 0-255.
5. **Manufacturer String**—Any sequence of up to 62 ASCII characters
6. **Product String**—Any sequence of up to 62 ASCII characters.
7. **Serial Number**—Any sequence of up to 30 ASCII characters.

If the “Prevent Further Programming” checkbox is selected, all customizable fields of the CP2110 are permanently locked and are no longer customizable.

See “AN434: CP2110 Interface Specification” for a full description of each of the customizable parameters.

If you are using the CP2110SetIDs to program multiple devices, you can save the customized values to a text file using the File → Save and File → Save As commands. When connecting a new device, use the File → Open command to retrieve the saved settings, which are then directly programmable to the new device.

**Notes:**

1. Avoid connecting more than one device containing the same VID, PID, and serial number combination.
2. GPIO.0 / CLK must be configured as a “CLK Output - Push Pull” before configuring the “CLK Output Divider” setting.
3. The Manufacturer String, Product String, and Serial Number are automatically converted to Unicode strings before programming.

## 4.3. Customizing Device Descriptors Using *CP2112SetIDs.exe* (CP2112)

*CP2112SetIDs.exe* is an example program that uses the CP2112 API Functions implemented by *SLABHIDtoUART.dll*. The program window is shown in Figure 3. *CP2112SetIDs.exe* demonstrates the method of accessing and changing the descriptors contained in the connected device's programmable area.

The various customizable fields of the CP2112 devices are only programmable one time using the program. After the fields are programmed once, *CP2112SetIDs* can access the fields, but it is not able to reprogram them.

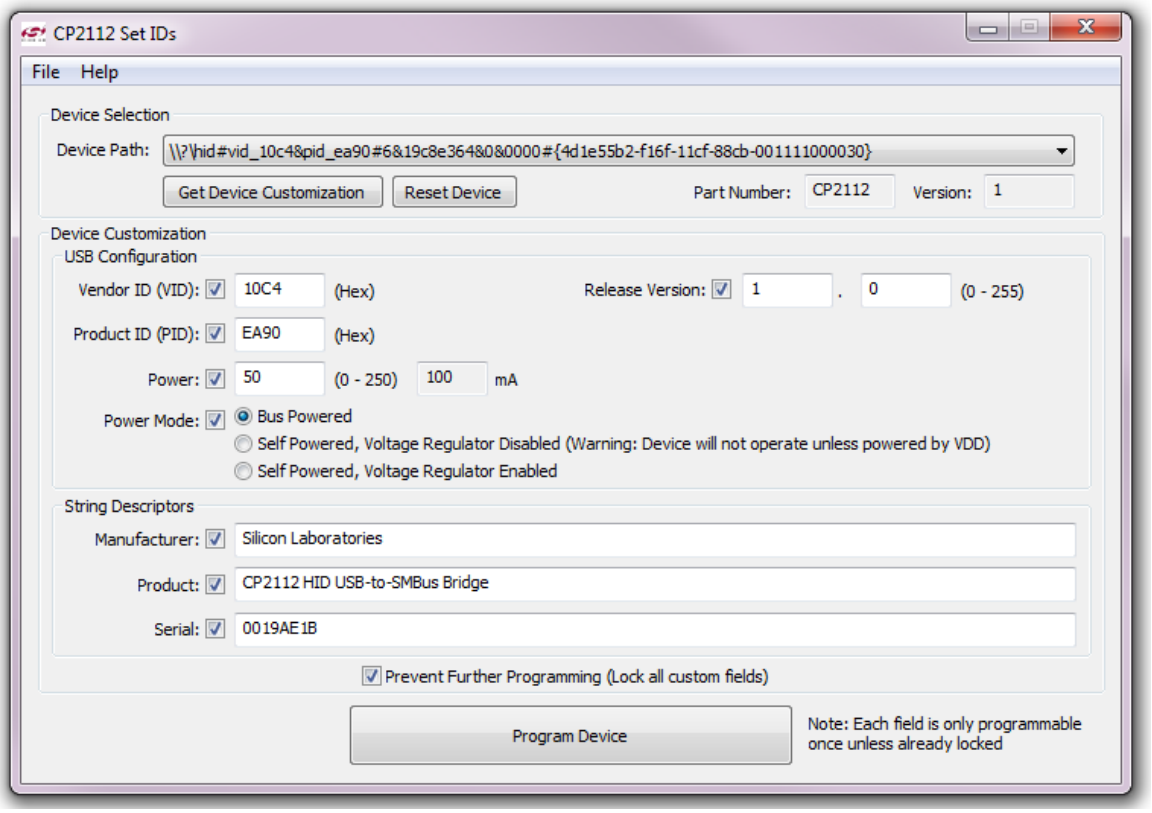


**Figure 3. CP2112SetIDs Application**

When *CP2112SetIDs.exe* is launched, the program searches for any CP2112 devices attached to the PC. The full path information for all of the devices found is inserted into the “Device Path” drop-down list, and the first device is selected automatically. The application then queries information from the selected device using the API functions and fills in the values for each of the selectable fields of the application. When a new device is selected from the list, the fields will be updated with information from the most recently selected device.

To change one or more of the values, click on the checkbox to the left of the field, and enter the new value. Once the new values are entered, click on the “Program Device” button.

The values entered are subject to the following limitations:

1. **VID**—2-byte value in hexadecimal.
2. **PID**—2-byte value hexadecimal.
3. **Max. Power**—1 byte value in hexadecimal digits with maximum setting of 0xFA (the value is in 2 mA units).
4. **Release Version**—Each field is a decimal number value 0–255.
5. **Manufacturer String**—Any sequence of up to 62 ASCII characters
6. **Product String**—Any sequence of up to 62 ASCII characters.
7. **Serial Number**—Any sequence of up to 30 ASCII characters.

If the “Prevent Further Programming” checkbox is selected, all customizable fields of the CP2112 are permanently locked and are no long customizable.

See “AN496: CP2112 HID USB-to-SMBus API Specification” for a full description of each of the customizable parameters.

If you are using the *CP2112SetIDs* to program multiple devices, the customized values can be saved to a text file using the File → Save and File → Save As commands. When connecting a new device, use the File → Open command to retrieve the saved settings, which can be directly programmed to the new device.

**Notes:**

1. Avoid connecting more than one device containing the same VID, PID, and serial number combination.
2. The Manufacturer String, Product String, and Serial Number are automatically converted to Unicode strings before programming.

## 4.4. Building *CP210xSetIDs.exe, CP2110SetIDs.exe, or CP2112SetIDs.exe*

Open the corresponding project file for the program with the *.dsw extension in Visual Studio 6.0. Select “Release” or “Debug”, and build the project.

## 4.5. Creating Custom Applications

For CP2101/2/3/4/5 devices, custom applications can use the CP210x Host API implemented in *CP210xManufacturing.DLL*. To use functions implemented in *CP210xManufacturing.DLL*, link CP210xManufacturing.LIB with your Visual C++ 6.0 application. Include *CP210xManufacturingDLL.h* in any file that calls functions implemented in *CP210xManufacturing.DLL*. See “5. CP210x Host API Functions (CP2101/2/3/4/5)” for more information regarding the DLL.

For CP2110 devices, custom applications can use the CP2110 API implemented in *SLABHIDtoUART.dll*. To use functions implemented in *SLABHIDtoUART.dll*, link SLABHIDtoUART.lib with your Visual C++ 6.0 application. Include *SLABCP2110.h* in any file that calls functions implemented in *SLABHIDtoUART.dll*. See “AN433: CP2110 HID to UART API Specification” for more details.

For CP2112 devices, custom applications can use the CP2112 API implemented in *SLABHIDtoSMBus.dll*. To use functions implemented in *SLABHIDtoSMBus.dll*, link SLABHIDtoSMBus.lib with your Visual C++ 6.0 application. Include *SLABCP2112.h* in any file that calls functions implemented in *SLABHIDtoSMBus.dll*. See “AN496: CP2112 HID USB to SMBus API Specification” for more details.

## 5. CP210x Host API Functions (CP2101/2/3/4/5)

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_SetProductString()` | Sets the product description string of a CP210x 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_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_GetDeviceProductString()` | Returns the product description string of a CP210x device. |
| `CP210x_GetDeviceSerialNumber()` | Returns the serial number string of a CP210x device. |
| `CP210x_GetDeviceVid()` | Returns the vendor ID of a CP210x device. |
| `CP210x_GetDevicePid()` | Returns the product ID of a CP210x device. |
| `CP210x_GetSelfPower()` | Returns the self-power attribute of a CP210x device. |
| `CP210x_GetMaxPower()` | Returns max. power consumption value of a CP210x device. |
| `CP210x_GetDeviceVersion()` | Returns the version number of a CP210x device. |
| `CP210x_GetBaudRateConfig()` | Returns the baud rate configuration data of a CP210x device. |
| `CP210x_GetLockValue()` | Returns the 1-byte Lock Value of a CP210x device. |
| `CP210x_SetPortConfig()` | Sets the port configuration of a CP210x device. |
| `CP210x_GetPortConfig()` | Returns the port configuration of a CP210x device. |
| `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()*.

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

Each of these functions is described in the following sections. Type definitions and constants are defined in "Appendix—Type Definitions and Constants" on page 21.

## 5.1. CP210x_GetNumDevices

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

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_GetNumDevices( LPDWORD NumDevices )`

**Parameters:** **1.** NumDevices—Address of a DWORD that will contain the number of devices.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_DEVICE_NOT_FOUND,
CP210x_INVALID_PARAMETER

## 5.2. 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.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_GetProductString( DWORD DeviceNum,
LPVOID DeviceString, DWORD Options )
```

**Parameters:**
**1.** DeviceNum—Index of the device for which the product description string, serial number, or full path is desired.
**2.** DeviceString—Variable of type *CP210x_DEVICE_STRING* returning the NULL-terminated serial number, device description or full path string.
**3.** Options—Flag that determines if *DeviceString* contains the product description, serial number, or full-path string.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_DEVICE_NOT_FOUND,
CP210x_INVALID_PARAMETER

## 5.3. CP210x_GetPartNumber

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

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS WINAPI CP210x_GetPartNumber(HANDLE cyHandle,
LPBYTE lpbPartNum);
```

**Parameters:**
1. Handle—Handle to the device returning a Part Number.
2. PartNum—Pointer to a 1-byte value returning the Part Number of the device. 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

## 5.4. 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, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_Open( DWORD DeviceNum, HANDLE* Handle )
```

**Parameters:**
1. DeviceNum—Device index. 0 for the first device, 1 for the second, etc.
2. Handle—Pointer to a variable where the handle to the device will be stored. This handle will be used for all subsequent accesses to the device.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_DEVICE_NOT_FOUND,
CP210x_INVALID_PARAMETER

## 5.5. CP210x_Close

**Description:** Closes an open device handle.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_Close( HANDLE Handle )
```

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

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE

## 5.6. 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

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_SetVid( HANDLE Handle, WORD Vid )`

**Parameters:**
1. Handle—Handle to the device to close 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

## 5.7. 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

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_SetPid( HANDLE Handle, WORD Pid )`

**Parameters:**
1. Handle—Handle to the device to close 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

## 5.8. 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*.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_SetProductString( HANDLE Handle, LPVOID Product,
BYTE Length, BOOL ConvertToUnicode=TRUE )
```

**Parameters:**

1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. Product—Buffer containing the Product String value.
3. Length—Length of the string in characters (not bytes), NOT including a NULL terminator.
4. ConvertToUnicode—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

## 5.9. 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

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_SetSerialNumber( HANDLE Handle,
LPVOID SerialNumber, BYTE Length, BOOL ConvertToUnicode=TRUE )
```

**Parameters:**

1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. SerialNumber—Buffer containing the Serial Number String value.
3. Length—Length in characters (not bytes), NOT including a NULL terminator.
4. ConvertToUnicode—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

## 5.10. 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

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_SetSelfPower( HANDLE Handle, BOOL SelfPower )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. SelfPower—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

## 5.11. CP210x_SetMaxPower

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

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_SetMaxPower( HANDLE Handle, BYTE MaxPower )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. MaxPower—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

## 5.12. CP210x_SetDeviceVersion

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

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_SetDeviceVersion( HANDLE Handle, WORD Version )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. Version—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

## 5.13. CP210x_SetBaudRateConfig

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

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

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS WINAPI CP210x_SetBaudRateConfig(HANDLE cyHandle,
BAUD_CONFIG* baudConfigData);
```

**Parameters:**
1. Handle—Handle to the device from which to get the Part Number.
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

## 5.14. CP210x_SetLockValue

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

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

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS WINAPI CP210x_SetLockValue(HANDLE cyHandle);`

**Parameters:** **1.** Handle—Handle of the device to lock. This will permanently set the lock value to 0x01.

**WARNING:** 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.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

## 5.15. 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*.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_GetDeviceProductString( HANDLE Handle,
LPVOID Product, LPBYTE Length, BOOL ConvertToASCII=TRUE )
```

**Parameters:**
**1.** Handle—Handle to the device to close as returned by CP210x_Open().
**2.** Product—Pointer to a buffer returning the Product String value.
**3.** Length—Pointer to a BYTE value returning the length of the string in characters (not bytes), NOT including a NULL terminator.
**4.** ConvertToASCII—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

## 5.16. 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, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_GetDeviceSerialNumber( HANDLE Handle,
LPVOID SerialNumber, LPBYTE Length, BOOL ConvertToASCII=TRUE )
```

**Parameters:**

1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. SerialNumber —Pointer to a buffer returning the Serial Number String value.
3. Length—Pointer to a BYTE value returning the length of the string in characters (not bytes), NOT including a NULL terminator.
4. ConvertToASCII—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

## 5.17. CP210x_GetDeviceVid

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

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS CP210x_GetDeviceVid( HANDLE Handle, LPWORD Vid )
```

**Parameters:**

1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. VID—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

## 5.18. CP210x_GetDevicePid

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

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_GetDevicePid( HANDLE Handle, LPWORD Pid )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. PID—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

## 5.19. 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, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_GetSelfPower( HANDLE Handle, LPBOOL SelfPower )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. SelfPower—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

## 5.20. CP210x_GetMaxPower

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

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_GetMaxPower( HANDLE Handle, LPBYTE MaxPower )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. MaxPower—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

## 5.21. CP210x_GetDeviceVersion

**Description:** Returns the 2-byte Device Release Version field of the Device Descriptor of a CP210x device.

**Supported Devices:** CP2101, CP2102, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_GetDeviceVersion( HANDLE Handle, LPWORD Version )`

**Parameters:**
1. Handle—Handle to the device to close as returned by *CP210x_Open()*.
2. Version—Pointer to a 2-byte value returning the 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_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

## 5.22. CP210x_GetBaudRateConfig

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

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

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS WINAPI CP210x_GetBaudRateConfig(HANDLE cyHandle,
BAUD_CONFIG* baudConfigData);
```

**Parameters:**
1. Handle—Handle to the device on which to determine the lock value.
2. BaudConfigData—Pointer to a *BAUD_CONFIG* structure returning the Baud Config data of the device.

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_PARAMETER,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

## 5.23. CP210x_GetLockValue

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

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

**Location:** CP210x Manufacturing DLL

**Prototype:**
```
CP210x_STATUS WINAPI CP210x_GetLockValue(HANDLE cyHandle,
LPBYTE lpbLockValue);
```

**Parameters:**
1. Handle—Handle to the device on which to determine the lock value.
2. LockValue—Pointer to a 1-byte value returning the Lock Value of the device. A 0x01 denotes 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

## 5.24. CP210x_GetPortConfig

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

**Supported Devices:** CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210X_STATUS CP210x_GetPortConfig(HANDLE Handle, LPVOID PortConfig)`

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

**Return Value:** CP210X_STATUS = CP210X_SUCCESS,
CP210X_INVALID_HANDLE,
CP210X_DEVICE_IO_FAILED,
CP210X_UNSUPPORTED_DEVICE

## 5.25. CP210xSetPortConfig

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

**Supported Devices:** CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210X_STATUS CP210x_SetPortConfig(HANDLE Handle, LPVOID PortConfig)`

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

**Return Value:** CP210X_STATUS = CP210X_SUCCESS,
CP210X_INVALID_HANDLE,
CP210X_DEVICE_IO_FAILED,
CP210X_UNSUPPORTED_DEVICE

## 5.26. 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, CP2103, CP2104, CP2105

**Location:** CP210x Manufacturing DLL

**Prototype:** `CP210x_STATUS CP210x_Reset( HANDLE Handle )`

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

**Return Value:** CP210x_STATUS = CP210x_SUCCESS,
CP210x_INVALID_HANDLE,
CP210x_DEVICE_IO_FAILED

## APPENDIX—TYPE DEFINITIONS AND CONSTANTS

Type Definitions from C++ Header File CP210xManufacturingDLL.h

```
// GetProductString() function flags
#define     CP210x_RETURN_SERIAL_NUMBER             0x00
#define     CP210x_RETURN_DESCRIPTION               0x01
#define     CP210x_RETURN_FULL_PATH                 0x02

// GetDeviceVersion() return codes
#define     CP210x_CP2101_VERSION                   0x01
#define     CP210x_CP2102_VERSION                   0x02
#define     CP210x_CP2103_VERSION                   0x03
#define     CP210x_CP2104_VERSION                   0x04
#define     CP210x_CP2105_VERSION                   0x05

// Return codes
#define     CP210x_SUCCESS                          0x00
#define     CP210x_DEVICE_NOT_FOUND                 0xFF
#define     CP210x_INVALID_HANDLE                   0x01
#define     CP210x_INVALID_PARAMETER                0x02
#define     CP210x_DEVICE_IO_FAILED                 0x03
#define     CP210x_FUNCTION_NOT_SUPPORTED           0x04
#define     CP210x_GLOBAL_DATA_ERROR                0x05
#define     CP210x_FILE_ERROR                       0x06
#define     CP210x_COMMAND_FAILED                   0x08
#define     CP210x_INVALID_ACCESS_TYPE              0x09

// Type definitions
typedef     int         CP210x_STATUS;

// Buffer size limits
#define     CP210x_MAX_DEVICE_STRLEN                256
#define     CP210x_MAX_PRODUCT_STRLEN               126
#define     CP210x_MAX_SERIAL_STRLEN                63
#define     CP210x_MAX_MAXPOWER                     250


// Type definitions
typedef     char  CP210x_DEVICE_STRING[CP210x_MAX_DEVICE_STRLEN];
typedef     char  CP210x_PRODUCT_STRING[CP210x_MAX_PRODUCT_STRLEN];
typedef     char  CP210x_SERIAL_STRING[CP210x_MAX_SERIAL_STRLEN];


//Baud Rate Aliasing definitions
#define     NUM_BAUD_CONFIGS                        32
```

```
typedef     struct
{
      WORD  BaudGen;
      WORD  Timer0Reload;
      BYTE  Prescaler;
      DWORD       BaudRate;
} BAUD_CONFIG;

#define     BAUD_CONFIG_SIZE10

typedef     BAUD_CONFIG          BAUD_CONFIG_DATA[NUM_BAUD_CONFIGS];


//Port Config definitions
typedef     struct
{
      WORD Mode;        // Push-Pull = 1, Open-Drain = 0
      WORD Reset_Latch; // Logic High = 1, Logic Low = 0
      WORD Suspend_Latch;// Logic High = 1, Logic Low = 0
      unsigned char EnhancedFxn;
} PORT_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_RESERVED_0                     0x08   //  Reserved, leave bit 3 cleared
#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
```

# DOCUMENT CHANGE LIST

## Revision 1.4 to Revision 1.5

- Updated text in "1. Introduction" on page 1.
- Updated text in "3. Customizing Driver Installations, CP210x Custom Setup Utility (CP2101/2/3/4/5)" on page 1.
- Sections 3.1 through 3.7 removed.
- “Customizing Driver Installations, Macintosh OS9 and OSX” removed.
- CP210x.DLL changed to CP210xManufacturing.DLL
- CP210x.LIB changed to CP210xManufacturing.LIB
- CP210x.h changed to CP210xManufacturingDLL.h

## Revision 1.5 to Revision 1.6

- Added CP2103 to Relevant Devices on page 1.
- "4.4. Building CP210xSetIDs.exe, CP2110SetIDs.exe, or CP2112SetIDs.exe" on page 7.
  - Updated title
- Added "5.24. CP210x_GetPortConfig" on page 20.
- Added "5.25. CP210xSetPortConfig" on page 20.
- " Appendix—Type Definitions and Constants" on page 21.
  - Updated code.

## Revision 1.6 to Revision 1.7

- Corrected typo in the warning in section "5.14. CP210x_SetLockValue" on page 15.
- Correct PDF bookmarks.

## Revision 1.7 to Revision 1.8

- Added support for CP2104, CP2105.

## Revision 1.8 to Revision 1.9

- Added support for CP2110 and CP2112.

## 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.