MCP2221 DLL User Guide

Contents

Document Revision History

Version	Release Date	Description
V1.0	4-Jan-2016	Initial release
V1.1	14-Mar-2016	Added information about calling convention
V1.2	18-May-2016	Added SmbusSendByte and SmbusReceiveByte documentation

Which library type to choose?

The MCP2221 DLL package contains two different types of DLL: managed and unmanaged. The managed DLL utilizes the Microsoft .NET framework when the unmanaged does not. To get help on which version to use, follow the guidelines below:

Scenario:	Which version to use:
You are planning to use the DLL with a .NET application	Managed
You are looking for the most simple way to interface with this DLL	Managed
You are using Visual Studio IDE	Managed
You do not want your application to require the .NET framework	Unmanaged
You are using programming tools/languages like Python, Java, C++, LabVIEW, etc.	Unmanaged

The package also contains the MCP2221 control API in the form of a standard C library (.lib) that can be statically linked to the user application.

DLL Requirements

A breakdown of the requirements is shown below:

DLL Version:	Requirements:
Managed (.NET4 version)	1. .NET framework (v4 Client profile or higher) 2. Microsoft Visual C++ 2010 Redistributable Package (x86) (OR msvcp100.dll and msvcr100.dll files in MCP2221 DLL directory)
Managed (.NET2 version)	1. .NET framework (v2 or V3.5) 2. Microsoft Visual C++ 2008 Redistributable Package (x86) (OR msvcp90.dll, msvcr90.dll, and msvcm90.dll files in MCP2221 DLL directory)
Unmanaged	No redistributable package required.

DLL design main characteristics

API similarity

The two types of DLL provide the same functionality, hence they export very similar API with a very few exceptions:

- The **managed** DLL naming convention for functions, constants and error codes adds the prefix "**M_**" to the similar names exported by the **unmanaged** DLL.
- The **unmanaged** function parameters are standard C types, while the **managed** functions parameters are Microsoft C++ compatible, adapted for easy integration with .NET applications
- The **unmanaged** DLL exports no dynamically allocated memory buffer pointer. The user application must ensure proper memory management and buffer provisioning for the data that **unmanaged** API returns to the user.

Please refer to the individual API function descriptions in order to ensure the appropriate API utilization for the version of the DLL chosen.

DLL Structure

The **managed** DLL implements a public class (named "MCP2221") with public **static** methods, implementing the managed API as a wrapper over the unmanaged API. As a matter of fact, the unmanaged library is statically linked to the managed DLL. In addition to the public methods, the MCP2221 class also exports as public, "static const" all the error codes and configuration constants.

DLL Initialization

One very important characteristic for managed and unmanaged DLLs is that both are **stateless.** There is no DLL initialization required. Also, the DLL makes no assumption about the device connection status or about the device configuration. Therefore, the user application design must take into consideration the following aspects:

- The error codes returned by the DLL API must be checked and addressed accordingly. For instance, the communication errors should be treated as catastrophic failures, as if there is a hardware failure or the device is disconnected.
- It is the user application responsibility to manage and keep the track of the device status, using the available status interrogation API and the error codes reported by the API calls

API integration main considerations

The **device handles**:

- The DLL makes no assumption about how many MCP2221 devices are connected (the limitations are actually given by the system memory resources). The library can be used to control any of them simultaneously because they are uniquely identified by "handles". The MCP2221 devices are automatically recognized and enumerated by Windows OS as HID devices. The operating

system offers support to access the devices as binary “files”, using normal “read” and “write” operations and file handles. So, the first step is to “**open”** the device and to save the returned “**handle**”. All the other API function calls must receive as parameter the unique device handle.

- The device handles returned by the DLL “open” APIs allows only **exclusive access**: one device cannot be shared by multiple applications/processes.
- The device handle is not automatically closed if the device gets disconnected while being in the “open” state. Therefore, the application must close the handle once the device abnormal state is determined, based on the error codes returned by subsequent attempts to access the device.
- Because of the device exclusive access mode, it is recommended that the application closes the device handle if the device access is no longer needed.

Device parallel access / multithreading:

- Since the device “open” is made in exclusive mode, only one application can access a certain device and that application can open only a single handle. The attempts to open a second handle will return “already open” error code.
- Although the handles are “exclusive”, the library API is not “thread safe” because the API calls are not atomic. Since most DLL API functions need to do several “file read/write” operations in sequence in order to implement the end-user features, the user application must ensure that:
    - All the “device open” calls are serialized
    - Each “open” call is followed by a “get last error” call in order to check the outcome of the open operation
    - Each API calls for the same MCP2221 device are serialized and consistent with the desired functionality.

Device **index** and **serial number**:

- The device **index** may change dynamically, if new MCP2221 devices are connected or removed. So, the user must not do any association between the device handle and the ordinal index that was used to open it with “open by index” API.
- If there are many devices with the same **serial** number connected (quite unlikely), only one can be successfully open using “open by SN” API.

The library **API C/C++ calling convention**:

- MCP22xy dll v2 libraries are built using the "__stdcall" calling convention (Visual Studio C/C++ compiler option "/Gz"). Please make sure that your C/C++ project default calling convention is set to "__stdcall".
- Alternatively, if the project setting cannot be changed globally, the included mcp22xx library header file (e.g. mcp2221_dll_um.h) can be adapted to indicate for each API the calling convention.

  For example, change:
  " *MCP2221_DLL_UM_API int Mcp2221_GetLibraryVersion(wchar_t *version);*" to:
  "*MCP2221_DLL_UM_API int* ***__stdcall*** *Mcp2221_GetLibraryVersion(wchar_t *version);*"

Function List

Unmanaged DLL	Managed DLL
Mcp2221_GetLibraryVersion	M_Mcp2221_GetLibraryVersion
Mcp2221_GetConnectedDevices	M_Mcp2221_GetConnectedDevices
Device Connection	
Mcp2221_OpenByIndex	M_Mcp2221_OpenByIndex
Mcp2221_OpenBySN	M_Mcp2221_OpenBySN
Mcp2221_Close	M_Mcp2221_Close
Mcp2221_CloseAll	M_Mcp2221_CloseAll
Mcp2221_Reset	M_Mcp2221_Reset
Mcp2221_GetLastError	M_Mcp2221_GetLastError
I2C/SMBus	
Mcp2221_SetSpeed	M_Mcp2221_SetSpeed
Mcp2221_SetAdvancedCommParams	M_Mcp2221_SetAdvancedCommParams
Mcp2221_I2cCancelCurrentTransfer	M_Mcp2221_I2cCancelCurrentTransfer
Mcp2221_I2cRead	M_Mcp2221_I2cRead
Mcp2221_I2cWrite	M_Mcp2221_I2cWrite
Mcp2221_I2cWriteNoStop	M_Mcp2221_I2cWriteNoStop
Mcp2221_I2cReadRestart	M_Mcp2221_I2cReadRestart
Mcp2221_I2cWriteRestart	M_Mcp2221_I2cWriteRestart
Mcp2221_SmbusSendByte	M_Mcp2221_SmbusSendByte
Mcp2221_SmbusReceiveByte	M_Mcp2221_SmbusReceiveByte
Mcp2221_SmbusReadByte	M_Mcp2221_SmbusReadByte
Mcp2221_SmbusWriteWord	M_Mcp2221_SmbusWriteWord
Mcp2221_SmbusReadWord	M_Mcp2221_SmbusReadWord
Mcp2221_SmbusBlockWrite	M_Mcp2221_SmbusBlockWrite
Mcp2221_SmbusBlockRead	M_Mcp2221_SmbusBlockRead
Mcp2221_ SmbusBlockWriteBlockReadProcessCall	M_Mcp2221_ SmbusBlockWriteBlockReadProcessCall
USB settings and Device Information	
Mcp2221_GetManufacturerDescriptor	M_Mcp2221_GetManufacturerDescriptor
Mcp2221_SetManufacturerDescriptor	M_Mcp2221_SetManufacturerDescriptor
Mcp2221_GetProductDescriptor	M_Mcp2221_GetProductDescriptor
Mcp2221_SetProductDescriptor	M_Mcp2221_SetProductDescriptor
Mcp2221_GetSerialNumberDescriptor	M_Mcp2221_GetSerialNumberDescriptor
Mcp2221_SetSerialNumberDescriptor	M_Mcp2221_SetSerialNumberDescriptor

Mcp2221_GetFactorySerialNumber	M_Mcp2221_GetFactorySerialNumber
Mcp2221_GetVidPid	M_Mcp2221_GetVidPid
Mcp2221_SetVidPid	M_Mcp2221_SetVidPid
Mcp2221_GetUsbPowerAttributes	M_Mcp2221_GetUsbPowerAttributes
Mcp2221_SetUsbPowerAttributes	M_Mcp2221_SetUsbPowerAttributes
Mcp2221_GetSerialNumberEnumerationEnable	M_Mcp2221_GetSerialNumberEnumerationEnable
Mcp2221_SetSerialNumberEnumerationEnable	M_Mcp2221_SetSerialNumberEnumerationEnable
Mcp2221_GetHwFwRevisions	M_Mcp2221_GetHardwareRevision M_Mcp2221_GetFirmwareRevision
Pin Functions	
Mcp2221_GetInitialPinValues	M_Mcp2221_GetInitialPinValues
Mcp2221_SetInitialPinValues	M_Mcp2221_SetInitialPinValues
Mcp2221_GetInterruptEdgeSetting	M_Mcp2221_GetInterruptEdgeSetting
Mcp2221_SetInterruptEdgeSetting	M_Mcp2221_SetInterruptEdgeSetting
Mcp2221_ClearInterruptPinFlag	M_Mcp2221_ClearInterruptPinFlag
Mcp2221_GetInterruptPinFlag	M_Mcp2221_GetInterruptPinFlag
Mcp2221_GetClockSettings	M_Mcp2221_GetClockSettings
Mcp2221_SetClockSettings	M_Mcp2221_SetClockSettings
Mcp2221_GetDacVref	M_Mcp2221_GetDacVref
Mcp2221_SetDacVref	M_Mcp2221_SetDacVref
Mcp2221_GetAdcData	M_Mcp2221_GetAdcData
Mcp2221_GetAdcVref	M_Mcp2221_GetAdcVref
Mcp2221_SetAdcVref	M_Mcp2221_SetAdcVref
Mcp2221_GetDacValue	M_Mcp2221_GetDacValue
Mcp2221_SetDacValue	M_Mcp2221_SetDacValue
Mcp2221_GetGpioSettings	M_Mcp2221_GetGpioSettings
Mcp2221_SetGpioSettings	M_Mcp2221_SetGpioSettings
Mcp2221_GetGpioValues	M_Mcp2221_GetGpioValues
Mcp2221_SetGpioValues	M_Mcp2221_SetGpioValues
Mcp2221_GetGpioDirection	M_Mcp2221_GetGpioDirection
Mcp2221_SetGpioDirection	M_Mcp2221_SetGpioDirection
Security	
Mcp2221_GetSecuritySetting	M_Mcp2221_GetSecuritySetting
Mcp2221_SetSecuritySettings	M_Mcp2221_SetSecuritySetting
Mcp2221_SetPermanentLock	M_Mcp2221_SetPermanentLock
Mcp2221_SendPassword	M_Mcp2221_SendPassword

DLL Error Codes

Value	Unmanaged	Managed	Description	Suggestion
0	E_NO_ERR	M_E_NO_ERR	Operation was successful	
-1	E_ERR_UNKOWN_ERROR	M_E_ERR_UNKOWN_ERROR	Unknown error. This can happen in the getconnecteddevices, openbyindex or openbysn if searching through the connected hid devices fails.	Try again
-2	E_ERR_CMD_FAILED	M_E_ERR_CMD_FAILED	The library indicates an unexpected device reply after being given a command: neither successful operation nor specific error code.	This is a command failure indication. Depending on the application strategy, the next step can be a device status check followed by command retry.
-3	E_ERR_INVALID_HANDLE	M_E_ERR_INVALID_HANDLE	Invalid device handle usage attempt. The device is already closed or there is an issue with the device handles management in the application	Re-open the device, or exit the application.
-4	E_ERR_INVALID_PARAMETER	M_E_ERR_INVALID_PARAMETER	At least one api parameter is not valid.	Check the parameter validity and try again.
-5	E_ERR_INVALID_PASS	M_E_ERR_INVALID_PASS	Invalid password string (length < 8)	Check the password string and try again.
-6	E_ERR_PASSWORD_LIMIT_REACHED	M_E_ERR_PASSWORD_LIMIT_REACHED	An incorrect password was sent 3 times.	Reset the device, check the password and try again
-7	E_ERR_FLASH_WRITE_PROTECTED	M_E_ERR_FLASH_WRITE_PROTECTED	The command cannot be executed because the device is password protected or locked.	Check the security settings (GetSecuritySetting) and if the device is not permanently locked, send the current password before retrying the operation
-10	E_ERR_NULL	M_E_ERR_NULL	Null pointer received	Validate the input parameters
-11	E_ERR_DESTINATION_TOO_SMALL	M_E_ERR_DESTINATION_TOO_SMALL	Destination string too small	
-12	E_ERR_INPUT_TOO_LARGE	M_E_ERR_INPUT_TOO_LARGE	The input string exceeds the maximum allowed size	Check that the string length is within the range provided in the function documentation.

-13	E_ERR_FLASH_WRITE_FAILED	M_E_ERR_FLASH_WRITE_FAILED	Flash write failed due to an unknown cause	
-14	E_ERR_MALLOC	M_ E_ERR_MALLOC	Memory allocation error	
-101	E_ERR_NO_SUCH_INDEX	M_E_ERR_NO_SUCH_INDEX	An attempt was made to open a connection to a non existing index (usually >= the number of connected devices)	Check the number of connected devices (with *getconnecteddevices)*; the index must be smaller
-103	E_ERR_DEVICE_NOT_FOUND	M_E_ERR_DEVICE_NOT_FOUND	No device with the provided vid/pid or SN has been found. This error can also occur during i2c/smbus operations if the device is disconnected from the usb before the operation is complete. The OpenBySn method will also return this code if a connection to a matching device is already open.	
-104	E_ERR_INTERNAL_BUFFER_TOO_SMALL	M_E_ERR_INTERNAL_BUFFER_TOO_SMALL	One of the internal buffers of the function was too small	No action
-105	E_ERR_OPEN_DEVICE_ERROR	M_E_ERR_OPEN_DEVICE_ERROR	An error occurred when trying to g et the device handle	Retry operation
-106	E_ERR_CONNECTION_ALREADY_OPENED	M_E_ERR_CONNECTION_ALREADY_OPENED	Connection already opened	Sharing mode is not allowed. Please read the paragraph “device parallel access / multithreading
-107	E_ERR_CLOSE_FAILED	M_E_ERR_CLOSE_FAILED	File close operation failed due to unknown reasons.	Try again or exit the application
-301	E_ERR_RAW_TX_TOO_LARGE	M_E_ERR_RAW_TX_TOO_LARGE	Low level communication error, shouldn't appear during normal operation	Restart application
-302	E_ERR_RAW_TX_COPYFAILED	M_E_ERR_RAW_TX_COPYFAILED	Low level communication error, shouldn't appear during normal operation	Restart application
-303	E_ERR_RAW_RX_COPYFAILED	E_ERR_RAW_RX_COPYFAILED	Low level communication error shouldn't appear during normal	Restart application

			operation	
-401	E_ERR_INVALID_SPEED	M_E_ERR_INVALID_SPEED	I2c/smbus speed is not within accepted range of 46875 - 500000	
-402	E_ERR_SPEED_NOT_SET	M_E_ERR_SPEED_NOT_SET	The speed may fail to be set if an i2c/smbus operation is already in progress or in a timeout situation. The "mcp2221_i2ccancelcurrenttransfer" function can be used to free the bus before retrying to set the speed.	
-403	E_ERR_INVALID_BYTE_NUMBER	M_E_ERR_INVALID_BYTE_NUMBER	The byte count is outside the accepted range for the attempted operation	Check the valid range for the desired operation and retry
-404	E_ERR_INVALID_ADDRESS	M_E_ERR_INVALID_ADDRESS	Invalid slave address. If 7 bit addressing is used the maximum address value is 127	
-405	E_ERR_I2C_BUSY	M_E_ERR_I2C_BUSY	The mcp2221 i2c/smbus engine is currently busy.	Retry operation or call cancelcurrenti2ctransfer before another retry.
-406	E_ERR_I2C_READ_ERROR	M_E_ERR_I2C_READ_ERROR	Mcp2221 signaled an error during the i2c read operation	Retry or reset device before retrying
-407	E_ERR_ADDRESS_NACK	M_E_ERR_ADDRESS_NACK	Nack received for the slave address used	Check that the slave address is correct.
-408	E_ERR_TIMEOUT	M_E_ERR_TIMEOUT	Either the "timeout" or "retries" value has been exceeded and no reply was received from the slave.	I2c/smbus transfer is not working properly. Check the communication settings and try again. The retries and timeout values can also be updated in the SetAdvancedCommParams
-409	E_ERR_TOO_MANY_RX_BYTES	M_E_ERR_TOO_MANY_RX_BYTES	The number of received data bytes is greater than requested	
-410	E_ERR_COPY_RX_DATA_FAILED	M_E_ERR_COPY_RX_DATA_FAILED	Could not copy the data received from the slave into the provided buffer;	Check buffer size and retry operation
-411	E_ERR_NO_EFFECT	M_E_ERR_NO_EFFECT	The i2c engine (inside mcp2221) was already idle. The cancellation command had no effect.	

-412	E_ERR_COPY_TX_DATA_FAILED	M_E_ERR_COPY_TX_DATA_FAILED	Failed to copy the data into the hid buffer	Retry operation
-413	E_ERR_INVALID_PEC	M_E_ERR_INVALID_PEC	The slave replied with a pec value different than the expected one.	Check that the smbus parameters used are supported by the slave.
-414	E_ERR_BLOCK_SIZE_MISMATCH	M_E_ERR_BLOCK_SIZE_MISMATCH	The slave sent a different value for the block size(byte count) than we expected	Check that the SMBus parameters used are supported by the slave.

DLL Constants

Unmanaged	Managed	Value	Description
FLASH_SETTINGS	M_FLASH_SETTINGS	0	read/write chip flash settings
RUNTIME_SETTINGS	M_RUNTIME_SETTINGS	1	read/write chip runtime settings
NO_CHANGE	M_NO_CHANGE	0xff	Do not change the existing value. For example you can alter a pin's function and mark the rest as "no_change" to maintain their existing configuration.
MCP2221_GPFUNC_IO	MCP2221_GPFUNC_IO	0	Pin configured as input/output
MCP2221_GP_SSPND	M_MCP2221_GP_SSPND	1	Pin configured as SSPND
MCP2221_GP_CLOCK_OUT	M_MCP2221_GP_CLOCK_OUT	1	pin configured as ClockOut
MCP2221_GP_USBCFG	M_MCP2221_GP_USBCFG	1	pin configured for USBCFG
MCP2221_GP_LED_I2C	M_MCP2221_GP_LED_I2C	1	pin configured for I2C LED
MCP2221_GP_LED_UART_RX	M_MCP2221_GP_LED_UART_RX	2	pin configured for UART RX LED
MCP2221_GP_ADC	M_MCP2221_GP_ADC	2	pin configured for ADC
MCP2221_GP_LED_UART_TX	M_MCP2221_GP_LED_UART_TX	3	pin configured for UART TX LED
MCP2221_GP_DAC	M_MCP2221_GP_DAC	3	Pin configured for DAC function
MCP2221_GP_IOC	M_MCP2221_GP_IOC	4	Pin configured for Interrupt On Change
MCP2221_GPDIR_INPUT	M_MCP2221_GPDIR_INPUT	1	GPIO pin configured as input
MCP2221_GPDIR_OUTPUT	M_MCP2221_GPDIR_OUTPUT	0	GPIO pin configured as output

MCP2221_GPVAL_HIGH	M_MCP2221_GPVAL_HIGH	1	Logic high value for I/O pins
MCP2221_GPVAL_LOW	M_MCP2221_GPVAL_LOW	0	Logic low value for I/O pins
INTERRUPT_NONE	M_INTERRUPT_NONE	0	Interrupt on change trigger = none
INTERRUPT_POSITIVE_EDGE	M_INTERRUPT_POSITIVE_EDGE	1	interrupt on change trigger = positive edge
INTERRUPT_NEGATIVE_EDGE	M_INTERRUPT_NEGATIVE_EDGE	2	interrupt on change trigger = negative edge
INTERRUPT_BOTH_EDGES	M_INTERRUPT_BOTH_EDGES	3	interrupt on change trigger = both edges
VREF_VDD	M_VREF_VDD	0	ADC/DAC voltage reference = Vdd
VREF_1024V	M_VREF_1024V	1	ADC/DAC voltage reference = 1.024V
VREF_2048V	M_VREF_2048V	2	ADC/DAC voltage reference = 2.048V
VREF_4096V	M_VREF_4096V	3	ADC/DAC voltage reference = 4.096V
MCP2221_USB_BUS	M_MCP2221_USB_BUS	0x80	USB bus powered
MCP2221_USB_SELF	M_MCP2221_USB_SELF	0x40	USB self powered
MCP2221_USB_REMOTE	M_MCP2221_USB_REMOTE	0x20	USB remote wakeup enable
MCP2221_PASS_ENABLE	M_MCP2221_PASS_ENABLE	1	Enable password protection
MCP2221_PASS_DISABLE	M_MCP2221_PASS_DISABLE	0	Disable password protection
MCP2221_PASS_CHANGE	M_MCP2221_PASS_CHANGE	0xff	Change current password

Unmanaged Function List

Mcp2221_GetLibraryVersion

```
int Mcp2221_GetLibraryVersion(wchar_t* version)
================================================================================
Description: Returns the version number of the DLL

Parameters:
 Inputs: None
 Outputs:
  (wchar_t*) version - variable that will store the library version. The library version is a 10
                       character wchar string.

Returns: 0 for success or error code.
```

Mcp2221_GetConnectedDevices

```
int Mcp2221_GetConnectedDevices(unsigned int vid, unsigned int pid, unsigned int *noOfDevs)
================================================================================
Description: Gets the number of connected MCP2221s with the provided VID & PID.

Parameters:
 Inputs:
  (unsigned int)  vid      - The vendor id of the MCP2221 devices to count
  (unsigned int)  pid      - The product id of the MCP2221 devices to count

 Outputs:
  (unsigned int*) noOfDevs - The number of connected MCP2221s matching the provided VID and
                             PID

Returns: 0 if successful; error code otherwise
```

Device Connection

Mcp2221_OpenByIndex

```
void* Mcp2221_OpenByIndex(unsigned int VID, unsigned int PID, unsigned int index)
================================================================================
Description: Attempt to open a connection with a MCP2221 with a specific index.

Parameters:
 Inputs:
  (unsigned int) VID   - The vendor ID of the MCP2221 to connect to(Microchip default = 0x4D8)
  (unsigned int) PID   - The product ID of the MCP2221 to connect to(Microchip default = 0xDD)
  (unsigned int) index - The index of the MCP2221 to connect to. This value ranges from 0 to n-1,
                         where n is the number of connected devices. This value can be obtained
                         from "Mcp2221_GetConnectedDevices"

Returns: (void*) - the handle value if the connection was successfully opened or
                   INVALID_HANDLE_VALUE(-1) if not.

NOTE: If the operation failed, call the Mcp2221_GetLastError method to get the error code
```

Mcp2221_OpenBySN

```
void*  Mcp2221_OpenBySN(unsigned int VID, unsigned int PID, wchar_t * serialNo)
================================================================================
Description: Attempt to open a connection with a MCP2221 with a specific serial number.

Parameters:
 Inputs:
  (unsigned int) VID  - The vendor ID of the MCP2221 to connect to.(Microchip default = 0x4D8)
  (unsigned int) PID  - The product ID of the MCP2221 to connect to. Microchip default = 0xDD)
  (wchar_t*) serialNo - The serial number of the MCP2221 we want to connect to. Maximum 30
                        character value.

Returns: (void*) - the handle value if the connection was successfully opened or
                   INVALID_HANDLE_VALUE(-1) if not.

NOTE: If the operation failed, call the Mcp2221_GetLastError method to get the error code.
      If a connection to a matching device is already open, the function will fail with the
      "device not found" error.
```

Mcp2221_Close

```
int Mcp2221_Close(void *handle)
================================================================================
Description: Attempt to close a connection to a MCP2221.

Parameters:
 (void*) handle - The handle for the device we'll close the connection to.

Returns: (int) - 0 for success; error code otherwise.
```

Mcp2221_CloseAll

```
int Mcp2221_CloseAll()
================================================================================
Description: Attempt to close all the currently opened MCP2221 connections.
             If successful, all existing handles will be set to INVALID_HANDLE_VALUE

Returns: (int) - 0 for success or the number of devices that failed to close.
```

Mcp2221_Reset

```
int Mcp2221_Reset(void *handle)
================================================================================
Description: Reset the MCP2221 and close its associated handle.

Parameters:
 (void*) handle - The handle for the device we'll reset. If successful, the handle will also  be
                  closed.

Returns: (int) - 0 for success; error code otherwise.
```

Mcp2221_GetLastError

```
int Mcp2221_GetLastError()
================================================================================
Description: Gets the last error value. Used only for the Open methods.

Returns: (int) - The value for the last error code.
```

I2C/SMBus

Mcp2221_SetSpeed

```
int Mcp2221_SetSpeed(void* handle, unsigned int speed)
================================================================================
Description: Set the communication speed for I2C/SMBus operations.

Parameters:
 (void*)        handle - The handle for the device.
 (unsigned int) speed  - the communication speed. Accepted values are between 46875 and 500000.

Returns: (int) - 0 for success; error code otherwise.

NOTE: The speed may fail to be set if an I2C/SMBus operation is already in progress or in a
timeout situation. The "Mcp2221_I2cCancelCurrentTransfer" function can be used to free the bus
before retrying to set the speed.
```

Mcp2221_SetAdvancedCommParams

```
int Mcp2221_SetAdvancedCommParams(void *handle, unsigned char timeout, unsigned char maxRetries)
================================================================================================
Description: Set the time the MCP2221 will wait after sending the "read" command before trying to
             read back the data from the I2C/SMBus slave and the maximum number of retries if data
             couldn't be read back.

Parameters:
 (void*)          handle     - The handle for the device.
 (unsigned char)  timeout    - amount of time (in ms) to wait for the slave to send back data.
                               Default 3ms.

 (unsigned char)  maxRetries - the maximum amount of times we'll try to read data back from a
                               slave. Default = 5

Returns: (int) - 0 for success; error code otherwise.
```

Mcp2221_I2cCancelCurrentTransfer

```
int Mcp2221_I2cCancelCurrentTransfer(void* handle)
================================================================================================
Description: Cancel the current I2C/SMBus transfer

Parameters:
 Inputs:
  (void*) handle - The handle for the device.

Returns: (int) - 0 for success; error code otherwise.
```

Mcp2221_I2cRead

```
int Mcp2221_I2cRead(void* handle, unsigned int bytesToRead, unsigned char slaveAddress,
                    unsigned char use7bitAddress, unsigned char* i2cRxData)
================================================================================
Description: Read I2C data from a slave.

Parameters:
 Inputs:
  (void*)          handle         - The handle for the device.
  (unsigned int)   bytesToRead    - the number of bytes to read from the slave. Valid range  is
                                    between 1 and 65535.
  (unsigned char)  slaveAddress   - 7bit or 8bit I2C slave address, depending on the value of the
                                    "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of the
                                    address is set to 1 inside the function.
  (unsigned char)  use7bitAddress - if >0 - 7 bit address will be used for the slave. If 0 - 8 bit
                                    is used.
 Outputs:
  (unsigned char*) i2cRxData      - buffer that will contain the data bytes read from the slave.

Returns: (int) - 0 for success; error code otherwise.

NOTE: if the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

Mcp2221_I2cWrite

```
int Mcp2221_I2cWrite(void* handle, unsigned int bytesToWrite, unsigned char slaveAddress,
                     unsigned char use7bitAddress, unsigned char* i2cTxData)
================================================================================
Description: Write I2C data to a slave.

Parameters:
 Inputs:
  (void*)         handle         - The handle for the device.
  (unsigned int)  bytesToWrite   - the number of bytes to write to the slave. Valid range is
                                   between 0 and 65535.
  (unsigned char) slaveAddress   - 7bit or 8bit I2C slave address, depending on the value of the
                                   "use7bitAddress" flag.For 8 bit addresses, the R/W LSB of the
                                   address is set to 0 inside the function.
  (unsigned char) use7bitAddress - if >0 7 bit address will be used for the slave. If 0, 8 bit is
                                   used.
  (unsigned char*) i2cTxData     - buffer that will contain the data bytes to be sent to the
                                   slave.

Returns: (int) - 0 for success; error code otherwise.

NOTE: if the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

Mcp2221_I2cWriteNoStop

```
int Mcp2221_ I2cWriteNoStop(void* handle, unsigned int bytesToWrite, unsigned char slaveAddress,
                    unsigned char use7bitAddress, unsigned char* i2cTxData)
================================================================================================
Description: Write I2C data to a slave without sending the STOP bit.

Parameters:
    Inputs: (void*) handle                  - the handle for the device.
            (unsigned int) bytesToWrite     - the number of bytes to write to the slave. Valid
                                              range is between 0 and 65535.
            (unsigned char) slaveAddress    - 7bit or 8bit I2C slave address, depending on the
                                              value of the "use7bitAddress" flag. For 8 bit
                                              addresses, the R/W LSB of the address is set to 0
                                              inside the function.

            (unsigned char) use7bitAddress - if >0 7 bit address will be used for the slave. If 0,
                                              8 bit is used.
            (unsigned char*) i2cTxData      - buffer that will contain the data bytes to be sent to
                                              the slave.


Returns: (int) - 0 for success; error code otherwise.

NOTE: 1. The speed must be set via the "Mcp2221_SetSpeed" function before using this method. If
the speed has not been set an error will be returned.
      2. The SMBus Process Call command can be formed using Mcp2221_I2cWriteNoStop followed by
Mcp2221_I2cReadRestart.
```

Mcp2221_I2cReadRestart

```
int Mcp2221_I2cReadRestart(void* handle, unsigned int bytesToRead, unsigned char slaveAddress,
                           unsigned char use7bitAddress, unsigned char* i2cRxData)
================================================================================================
Description: Read I2C data from a slave starting with a Repeated START.

Parameters:
    Inputs: (void*) handle                  - the handle for the device.
            (unsigned int) bytesToRead      - the number of bytes to read from the slave. Valid
                                              range is between 1 and 65535.
            (unsigned char) slaveAddress    - 7bit or 8bit I2C slave address, depending on the
                                              value of the "use7bitAddress" flag. For 8 bit
                                              addresses, the R/W LSB of the address is set to 1
                                              inside the function.
            (unsigned char) use7bitAddress  - if >0 - 7 bit address will be used for the slave. If
                                              0 - 8 bit is used.

    Outputs: (unsigned char*) i2cRxData     - buffer that will contain the data bytes read from
                                              the slave.

Returns: (int) - 0 for success; error code otherwise.

NOTE: 1. The speed must be set via the "Mcp2221_SetSpeed" function before using this method. If
the speed has not been set an error will be returned.
      2. The SMBus Process Call command can be formed using Mcp2221_I2cWriteNoStop followed by
Mcp2221_I2cReadRestart.
```

Mcp2221_I2cWriteRestart

```
int Mcp2221_I2cWriteRestart(void* handle, unsigned int bytesToWrite, unsigned char slaveAddress,
                    unsigned char use7bitAddress, unsigned char* i2cTxData)
================================================================================================
Description: Write I2C data to a slave without sending the STOP bit.

Parameters:
    Inputs: (void*) handle                  - the handle for the device.
            (unsigned int) bytesToWrite     - the number of bytes to write to the slave. Valid
                                              range is between 0 and 65535.
            (unsigned char) slaveAddress    - 7bit or 8bit I2C slave address, depending on the
                                              value of the "use7bitAddress" flag. For 8 bit
                                              addresses, the R/W LSB of the address is set to 0
                                              inside the function.

            (unsigned char) use7bitAddress - if >0 7 bit address will be used for the slave. If 0,
                                              8 bit is used.
            (unsigned char*) i2cTxData      - buffer that will contain the data bytes to be sent to
                                              the slave.


Returns: (int) - 0 for success; error code otherwise.

NOTE: The speed must be set via the "Mcp2221_SetSpeed" function before using this method. If the
speed has not been set an error will be returned.
```

Mcp2221_SmbusSendByte

```
int Mcp2221_SmbusSendByte(void* handle, unsigned char slaveAddress, unsigned char use7bitAddress,
                          unsigned char usePec, unsigned char data)
================================================================================================
Description: SMBus Send byte. Sends one data byte.

Parameters:
 Inputs:
  (void*)         handle         - The handle for the device.
  (unsigned char) slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value of the
                                   "use7bitAddress" flag.For 8 bit addresses, the R/W LSB of the
                                   address is set to 0 inside the function.
  (unsigned char) use7bitAddress - if >0, 7 bit address will be used for the slave. If 0, 8 bit is
                                   used.
  (unsigned char) usePec         - if >0  Packet Error Checking (PEC) will be used. A PEC byte
                                   containing the CRC8 value for the sent message is appended
                                   after the data byte.
  (unsigned char) data           - The data byte.

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

Mcp2221_SmbusReceiveByte

```
int Mcp2221_SmbusReceiveByte(void* handle, unsigned char slaveAddress, unsigned char
                             use7bitAddress, unsigned char usePec, unsigned char *readByte)
================================================================================================
Description: SMBus Receive Byte. Read one data byte back.

Parameters:
 Inputs:
  (void*)         handle         - The handle for the device.
  (unsigned char) slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value of the
                                   "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of the
                                   address is set to 1 inside the function.
  (unsigned char) use7bitAddress - if >0, 7 bit address will be used for the slave. If 0, 8 bit is
                                   used.
  (unsigned char) usePec         - if >0, Packet Error Checking (PEC) will be used.

 Outputs:
  (unsigned char*) readByte - The data byte received from the slave

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used.Otherwise, the speed will not be reconfigured.
```

Mcp2221_SmbusWriteByte

```
int Mcp2221_SmbusWriteByte(void* handle, unsigned char slaveAddress, unsigned char use7bitAddress,
                           unsigned char usePec, unsigned char command, unsigned char data)
================================================================================================
Description: SMBus write byte. The first byte of a Write Byte operation is the command code. The
             next one is the data to be written.

Parameters:
 Inputs:
  (void*)         handle         - The handle for the device.
  (unsigned char) slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value of the
                                   "use7bitAddress" flag.For 8 bit addresses, the R/W LSB of the
                                   address is set to 0 inside the function.
  (unsigned char) use7bitAddress - if >0, 7 bit address will be used for the slave. If 0, 8 bit is
                                   used.
  (unsigned char) usePec         - if >0  Packet Error Checking (PEC) will be used. A PEC byte
                                   containing the CRC8 value for the sent message is appended
                                   after the data byte.
  (unsigned char) command        - The command code byte.
  (unsigned char) data           - The data byte.

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

Mcp2221_SmbusReadByte

```
int Mcp2221_SmbusReadByte(void* handle, unsigned char slaveAddress, unsigned char use7bitAddress,
                          unsigned char usePec, unsigned char command, unsigned char *readByte)
================================================================================================
Description: SMBus Read Byte. First Write the command byte to the slave, then read one data byte
             back.

Parameters:
 Inputs:
  (void*)         handle         - The handle for the device.
  (unsigned char) slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value of the
                                   "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of the
                                   address is set to 1 inside the function.
  (unsigned char) use7bitAddress - if >0, 7 bit address will be used for the slave. If 0, 8 bit is
                                   used.
  (unsigned char) usePec         - if >0, Packet Error Checking (PEC) will be used.
  (unsigned char) command        - The command code byte.

 Outputs:
  (unsigned char*) readByte - The data byte received from the slave

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used.Otherwise, the speed will not be reconfigured.
```

Mcp2221_SmbusWriteWord

```
int Mcp2221_SmbusWriteWord(void* handle, unsigned char slaveAddress, unsigned char use7bitAddress,
                           unsigned char usePec, unsigned char command, unsigned char* data)
================================================================================================
Description: SMBus write word. The first byte of a Write Byte operation is the command code,
             followed by the data_byte_low then data_byte_high.

Parameters:
 Inputs:
  (void*)           handle        - The handle for the device.
  (unsigned char)   slaveAddress  - 7bit or 8bit SMBus slave address, depending on the value of
                                    the "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of
                                    the address is set to 0 inside the function.
  (unsigned char) use7bitAddress  - if >0, 7 bit address will be used for the slave. If 0, 8 bit
                                    is used.
  (unsigned char)  usePec         - if >0, Packet Error Checking (PEC) will be used. A PEC byte
                                    containing the CRC8 value for the sent message is appended
                                    after the data byte.
  (unsigned char)  command        - The command code byte.
  (unsigned char*) data           - Array containing the low and high data bytes to be sent to the
                                    slave.
                                           data[0] will be considered the data_byte_low
                                           data[1] will be considered the data_byte_high

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

Mcp2221_SmbusReadWord

```
int Mcp2221_SmbusReadWord(void* handle, unsigned char slaveAddress, unsigned char use7bitAddress,
                          unsigned char usePec, unsigned char command, unsigned char* readData)
================================================================================================
Description: SMBus Read Word. First Write the command byte to the slave, then read one data byte
             back.

Parameters:
 Inputs:
  (void*)         handle         - The handle for the device.
  (unsigned char) slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value of the
                                   "use7bitAddress" flag.For 8 bit addresses, the R/W LSB of the
                                   address is set to 1 inside the function.
  (unsigned char) use7bitAddress - if >0, 7 bit address will be used for the slave. If 0, 8 bit is
                                   used.
  (unsigned char) usePec         - if >0 Packet Error Checking (PEC) will be used.
  (unsigned char) command        - The command code byte.

 Outputs:
  (unsigned char*) readData - Buffer that will store the read data word.
                              readData[0] - data_byte_low
                              readData[1] - data_byte_high

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

Mcp2221_SmbusBlockWrite

```
int Mcp2221_SmbusBlockWrite(void* handle, unsigned char slaveAddress, bool use7bitAddress,
                            unsigned char usePec, unsigned char command, unsigned char byteCount,
                            unsigned char* data)
================================================================================================
Description: SMBus Block Write. The first byte of a Block Write operation is the command code,
followed by the number of data bytes, then data bytes.

Parameters:
 Inputs:
  (void*)          handle         - The handle for the device.
  (unsigned char)  slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value of
                                    the "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of
                                    the address is set to 0 inside the function.
  (unsigned char)  use7bitAddress - if >0, 7 bit address will be used for the slave. If 0, 8 bit
                                    is used.
  (unsigned char)  usePec          - if >0, Packet Error Checking (PEC) will be used. A PEC byte
                                    containing the CRC8 value for the sent message is appended
                                    after the data byte.
  (unsigned char)  command         - The command code byte.
  (unsigned char)  byteCount       - the number of data bytes that will be sent to the slave. Valid
                                    range is between 0 and 255 bytes, conforming to the smbus v3
                                    specification.
  (unsigned char*) data            - Array containing the data bytes to be sent to the slave.

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

Mcp2221_SmbusBlockRead

```
int Mcp2221_SmbusBlockRead(void* handle, unsigned char slaveAddress, unsigned char use7bitAddress,
                           unsigned char usePec, unsigned char command, unsigned char byteCount,
                           unsigned char* readData)
================================================================================================
Description: SMBus Block Read.

Parameters:
 Inputs:
  (void*)          handle          - The handle for the device.
  (unsigned char) slaveAddress     - 7bit or 8bit SMBus slave address, depending on the value of
                                     the "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of
                                     the address is set to 1 inside the function.
  (unsigned char) use7bitAddress   - if >0, 7 bit address will be used for the slave. If 0, 8 bit
                                     is used.
  (unsigned char) usePec           - if >0, Packet Error Checking (PEC) will be used. The CRC8
                                     values is computed for the SMBus packet compared with the PEC
                                     byte sent by the slave. If the two values differ the function
                                     returns an error code.
  (unsigned char) command          - The command code byte.
  (unsigned char) byteCount        - (block size) the number of data bytes that the slave will send
                                     to the master. Valid range is between 1 and 255 bytes. If
                                     there is a mismatch between this value and the byteCount the
                                     slave reports that it will send, an error will be returned.

 Outputs:
  (unsigned char*) readData         - Array containing the data bytes read from the slave. If PEC
                                     is used, the last data byte will be the PEC byte received from
                                     the slave so the array should have a length of n+1, where n is
                                     the block size.


Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

Mcp2221_SmbusBlockWriteBlockReadProcessCall

```
int Mcp2221_SmbusBlockWriteBlockReadProcessCall(void* handle, unsigned char slaveAddress, unsigned
                          char use7bitAddress, unsigned char usePec, unsigned char command,
                          unsigned char writeByteCount, unsigned char* writeData, unsigned char
                          readByteCount, unsigned char* readData)
================================================================================================
Description: SMBus Block Write Block Read Process Call.
Parameters:
Inputs:
    (void*) handle                  - The handle for the device.
    (unsigned char) slaveAddress    - 7bit or 8bit SMBus slave address, depending on the value of
                                      the "use7bitAddress" flag. For 8 bit addresses, the R/W LSB
                                      of the address is set to 0 inside the function.
    (unsigned char) use7bitAddress - if >0, 7 bit address will be used for the slave. If 0, 8 bit
                                      is used.
    (unsigned char) usePec          - if >0, Packet Error Checking (PEC) will be used. The CRC8
                                      values is computed for the SMBus packet and compared with the
                                      PEC byte sent by the slave. If the two values differ the
                                      function returns an error code.
    (unsigned char) command         - The command code byte.
    (unsigned char) writeByteCount - the number of data bytes that will be sent to the slave. The
                                      total data payload must not exceed 255 bytes (writeByteCount
                                      + readByteCound <= 255) and writeByteCount > 0
    (unsigned char*) writeData      - array containing the data bytes to be sent to the slave.
    (unsigned char) readByteCount   - the number of data bytes that the slave will send to the
                                      master. If there is a mismatch between this value and the
                                      readByteCount the slave reports that it will send, an error
                                      will be returned. The total data payload must not exceed 255
                                      bytes (writeByteCount + readByteCound <= 255) and
                                      readByteCount > 0

Outputs:
    (unsigned char*) readData       - Array containing the data bytes read from the slave. If PEC
                                      is used, the last data byte will be the PEC byte received
                                      from the slave so the array should have a length of n+1,
                                      where n is the readByteCount size.

Returns: (int) - 0 for success; error code otherwise.
```

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.

USB settings and Device Information

Mcp2221_GetManufacturerDescriptor

```
int Mcp2221_GetManufacturerDescriptor(void* handle, wchar_t* manufacturerString)
================================================================================
Description: Read USB Manufacturer Descriptor string from device.

Parameters:
 Inputs:
  (void*) handle - The handle for the device.

 Outputs:
  (wchar_t*) manufacturerString - will contain the value of the USB Manufacturer Descriptor
                                  String.
    Note: the output string can contain up to 30 characters

Returns: (int) - 0 for success; error code otherwise.
```

Mcp2221_SetManufacturerDescriptor

```
int Mcp2221_SetManufacturerDescriptor(void* handle, wchar_t* manufacturerString)
================================================================================
Description: Write USB Manufacturer Descriptor string to the device.

Parameters:
 Inputs:
  (void*)    handle             - The handle for the device.
  (wchar_t*) manufacturerString - will contain the value of the USB Manufacturer Descriptor
                                  String.
    Note: the input string can contain a maximum of 30 characters

Returns: (int) - 0 for success; error code otherwise.
```

Mcp2221_GetProductDescriptor

```
int Mcp2221_GetProductDescriptor(void* handle, wchar_t* manufacturerString)
================================================================================
Description: Read USB Product Descriptor string from device.

Parameters:
 Inputs:
  (void*) handle - The handle for the device.

 Outputs:
  (wchar_t*) productString - will contain the value of the USB Product Descriptor String.
                             Note: the output string can contain up to 30 characters

Returns: (int) - 0 for success; error code otherwise.
```

Mcp2221_SetProductDescriptor

```
int Mcp2221_SetProductDescriptor(void* handle, wchar_t* productString)
================================================================================
Description: Write USB Product Descriptor string to the device.

Parameters:
 Inputs:
  (void*)    handle        - The handle for the device.
  (wchar_t*) productString - will contain the value of the USB Product Descriptor String.

   Note: the input string can contain a maximum of 30 characters

Returns: (int) - 0 for success; error code otherwise.
```

Mcp2221_GetSerialNumberDescriptor

```
int Mcp2221_GetSerialNumberDescriptor(void* handle, wchar_t* serialNumber)
================================================================================
Description: Read USB Serial Number Descriptor string from device.

Parameters:
 Inputs:
  (void*) handle - The handle for the device.

 Outputs:
  (wchar_t*) serialNumber - will contain the value of the USB Serial Number Descriptor String.
                            Note: the output string can contain up to 30 characters

Returns: (int) - 0 for success; error code otherwise.
```

Mcp2221_SetSerialNumberDescriptor

```
int Mcp2221_SetSerialNumberDescriptor(void* handle, wchar_t* serialNumber)
================================================================================
Description: Write USB Serial Number Descriptor string to the device.

Parameters:
 Inputs:
  (void*)    handle       - The handle for the device.
  (wchar_t*) serialNumber - will contain the value of the USB Serial Number Descriptor String.
                            Note: the input string can contain a maximum of 30 characters

Returns: (int) - 0 for success; error code otherwise.
```

Mcp2221_GetFactorySerialNumber

```
int Mcp2221_GetFactorySerialNumber(void* handle, wchar_t* serialNumber)
================================================================================
Description: Read the factory serial number of the device

Parameters:
 Inputs:
  (void*) handle - The handle for the device.

 Outputs:
 (wchar_t*) serialNumber - will contain the value of the factory serial number of the device
                           Note: the output string can contain a maximum of 30 characters

Returns: (int) - 0 for success; error code otherwise.
```

Mcp2221_GetVidPid

```
int Mcp2221_GetVidPid(void* handle, unsigned int* vid, unsigned int* pid)
================================================================================
Description: Gets the VID and PID for the selected device.

Parameters:
 Inputs:
  (void*) handle - the handle for the device

Outputs:
  (unsigned int*) vid - The vendor id of the MCP2221 device
  (unsigned int*) pid - The product id of the MCP2221 device

Returns: 0 if successful; error code otherwise
```

Mcp2221_SetVidPid

```
int Mcp2221_SetVidPid(void* handle, unsigned int vid, unsigned int pid)
================================================================================
Description: Sets the VID and PID for the selected device.

Parameters:
    Inputs: (void*) handle - the handle for the device
            (unsigned int) vid - The vendor id to be set
            (unsigned int) pid - The product id to be set
Returns: 0 if successful; error code otherwise

NOTE: the new VID/PID values will take effect after a device reset.
```

Mcp2221_GetUsbPowerAttributes

```
int Mcp2221_GetUsbPowerAttributes(void* handle, unsigned char* powerAttributes, int* currentReq)
================================================================================
Description: Gets the USB power attribute values.

Parameters:
 Inputs:
  (void*) handle - the handle for the device

 Outputs:
  (unsigned char*) powerAttributes - the power attributes value from the USB descriptor.
                                     Bit meanings, based on the USB 2.0 spec:
                                       bit 7 - Reserved (Set to 1) (equivalent to Bus Powered)
                                       bit 6 - Self Powered
                                       bit 5 - Remote Wakeup
                                       bits 4..0 Reserved (reset to 0)
  (unsigned int*)  currentReq      - the requested current value (mA); This value is expressed in
                                     multiples of 2mA.

Returns: 0 if successful; error code otherwise
```

Mcp2221_SetUsbPowerAttributes

```
int Mcp2221_SetUsbPowerAttributes(void* handle, unsigned char powerAttributes, int currentReq)
================================================================================================
Description: Sets the USB power attribute values.

Parameters:
 Inputs:
  (void*)          handle           - the handle for the device
  (unsigned char)  powerAttributes  - the power attributes value from the USB descriptor.
                                      Bit meanings, based on the USB 2.0 spec:
                                        bit 7 - Reserved (Set to 1) (equivalent to Bus Powered)
                                        bit 6 - Self Powered
                                        bit 5 - Remote Wakeup
                                        bits 4..0 Reserved (reset to 0)
                                    The following constants can be OR'd to set this value:
                                    MCP2221_USB_SELF, MCP2221_USB_REMOTE, MCP2221_USB_BUS
  (unsigned int)   currentReq       - the requested current value (mA); This value is expressed in
                                      multiples of 2mA. Valid range is between 0 and 500mA. If an
                                      odd value is used, it will be rounded down to the closest even
                                      value (ex currentReq = 201mA will result in a 200mA current
                                      request).

Returns: 0 if successful; error code otherwise

NOTE: For the PowerAttributes parameter, bits 7 and 0-4 are automatically set to the correct
reserved" value.
```

Mcp2221_GetSerialNumberEnumerationEnable

```
int Mcp2221_GetSerialNumberEnumerationEnable(void* handle, unsigned char* snEnumEnabled)
================================================================================
Description: Gets the status of the Serial number enumeration bit.

Parameters:
 Inputs:
  (void*) handle - the handle for the device

 Outputs:
  (unsigned char*) snEnumEnabled - determines if the serial number descriptor will be used during
                                   the USB enumeration of the CDC interface. If 1 - the serial
                                   number descriptor is used; if 0 - no serial number descriptor
                                   will be present during enumeration.

Returns: 0 if successful; error code otherwise
```

Mcp2221_SetSerialNumberEnumerationEnable

```
int Mcp2221_SetSerialNumberEnumerationEnable(void* handle, unsigned char snEnumEnabled)
================================================================================
Description: Sets the status of the Serial number enumeration bit.

Parameters:
 Inputs:
  (void*)         handle        - the handle for the device
  (unsigned char) snEnumEnabled - determines if the serial number descriptor will be used during
                                  the USB enumeration of the CDC interface. If 1 - the serial
                                  number descriptor is used; if 0 - no serial number descriptor
                                  will be present during enumeration.

Returns: 0 if successful; error code otherwise
```

Mcp2221_GetHwFwRevisions

```
int Mcp2221_GetHwFwRevisions(void* handle, wchar_t* hardwareRevision, wchar_t* firmwareRevision)
================================================================================================
Description: Reads the hardware and firmware revision values from the device.

Parameters:
 Inputs:
  (void*) handle - the handle for the device.

 Outputs:
  (wchar_t*) hardwareRevision - will contain the hardware revision string.
  (wchar_t*) firmwareRevision - will contain the firmware revision string.

Returns: 0 if successful; error code otherwise

NOTE: the output strings must have a minimum length of 2.
```

Pin Functions

Mcp2221_GetInitialPinValues

```
int Mcp2221_GetInitialPinValues(void* handle, unsigned char* ledUrxInitVal,
                                unsigned char* ledUtxInitVal, unsigned char* ledI2cInitVal,
                                unsigned char* sspndInitVal, unsigned char* usbCfgInitVal)
================================================================================
Description: Gets the initial values for the special function pins: LEDUARTRX, LEDUARTTX, LEDI2C,
                SSPND and USBCFG

Parameters:
 Inputs:
  (void*) handle - the handle for the device

 Outputs:
  (unsigned char*) ledUrxInitVal - this value represents the logic level signaled when no Uart Rx
                                   activity takes place
                                   (inactive level)
  (unsigned char*) ledUtxInitVal - this value represents the logic level signaled when no Uart Tx
                                   activity takes place (inactive level)
  (unsigned char*) ledI2cInitVal - this value represents the logic level signaled when no I2C
                                   traffic occurs (inactive level)
  (unsigned char*) sspndInitVal  - this value represents the logic level signaled when the device
                                   is not in suspend mode (inactive level)
  (unsigned char*) usbCfgInitVal - this value represents the logic level signaled when the device
                                   is not usb configured (inactive level)Returns: 0 if
                                   successful; error code otherwise

Returns: 0 if successful; error code otherwise
```

Mcp2221_SetInitialPinValues

```
int Mcp2221_SetInitialPinValues(void* handle, unsigned char ledUrxInitVal,
                                unsigned char ledUtxInitVal, unsigned char ledI2cInitVal,
                                unsigned char sspndInitVal, unsigned char usbCfgInitVal)
================================================================================
Description: Sets the initial values for the special function pins: LEDUARTRX, LEDUARTTX, LEDI2C,
             SSPND and USBCFG. The settings are saved to flash and take effect after a device
             reset.

Parameters:
 Inputs:
  (void*)          handle        - the handle for the device
  (unsigned char) ledUrxInitVal - this value represents the logic level signaled when no Uart Rx
                                  activity takes place (inactive level)
  (unsigned char) ledUtxInitVal - this value represents the logic level signaled when no Uart Tx
                                  activity takes place (inactive level)
  (unsigned char) ledI2cInitVal - this value represents the logic level signaled when no I2C
                                  traffic occurs (inactive level)
  (unsigned char) sspndInitVal  - this value represents the logic level signaled when the device
                                  is not in suspend mode (inactive level)
  (unsigned char) usbCfgInitVal - this value represents the logic level signaled when the device
                                  is not usb configured (inactive level)

Returns: 0 if successful; error code otherwise

NOTE: Accepted values for the logic levels are 0(low) and 1(high), 0xff (leave unchanged)
```

Mcp2221_GetInterruptEdgeSetting

```
int Mcp2221_GetInterruptEdgeSetting(void* handle, unsigned char whichToGet,
                                    unsigned char* interruptPinMode)
================================================================================
Description: Gets the interrupt pin trigger configuration.

Parameters:
 Inputs:
  (void*)          handle     - the handle for the device
  (unsigned char)  whichToGet - 0 to read Flash settings, >0 to read SRAM (runtime) settings

 Outputs:
  (unsigned char*) interruptPinMode - value representing which edge will trigger the interrupt
                                      0 - none
                                      1 - positive edge
                                      2 - negative edge
                                      3 - both

Returns: 0 if successful; error code otherwise
```

Mcp2221_SetInterruptEdgeSetting

```
int Mcp2221_SetInterruptEdgeSetting(void* handle, unsigned char whichToSet,
                                    unsigned char interruptPinMode)
=============================================================================================
Description: Sets the interrupt pin trigger configuration.

Parameters:
 Inputs:
  (void*)          handle           - the handle for the device
  (unsigned char)  whichToSet       - 0 to write Flash settings, >0 to write SRAM (runtime)
                                      settings
  (unsigned char)  interruptPinMode - value representing which edge will trigger the interrupt
                                         0 - none
                                         1 - positive edge
                                         2 - negative edge
                                         3 - both

Returns: 0 if successful; error code otherwise
```

Mcp2221_ClearInterruptPinFlag

```
int Mcp2221_ClearInterruptPinFlag(void* handle)
=============================================================================================
Description: Clears the interrupt pin flag of a device.

Parameters:
 Inputs:
  (void*) handle - the handle for the device for which the flag will be cleared.

Returns: 0 if successful; error code otherwise
```

Mcp2221_GetInterruptPinFlag

```
Mcp2221_GetInterruptPinFlag(void* handle, unsigned char* flagValue);
================================================================================
Description: Reads the interrupt pin flag value of a device.

Parameters:
 Inputs:
  (void*)          handle    - the handle for the device.
  (unsigned char*) flagValue - the value of the interrupt on change flag

Returns: 0 if successful; error code otherwise
```

Mcp2221_GetClockSettings

```
int Mcp2221_GetClockSettings(void*handle, unsigned char whichToGet, unsigned char* dutyCycle,
                             unsigned char* clockDivider)
================================================================================================
Description: Gets the duty cycle and clock divider values for the clock out pin (if configured for
                this operation).

Parameters:
 Inputs:
  (void*)          handle       - the handle for the device
  (unsigned char)  whichToGet   - 0 to read Flash settings, >0 to read SRAM (runtime) settings

 Outputs:
  (unsigned char*) dutyCycle    - value of the duty cycle of the waveform on the clock pin
                                    0 - 0 %
                                    1 - 25 %
                                    2 - 50 %
                                    3 - 75 %
  (unsigned char*) clockDivider - value of the clock divider. The value provided is a power of 2.
                The 48Mhz internal clock is divided by 2^value to obtain the output waveform
                frequency. The correspondence between the divider values and output frequencies
                are as follows:
                                    1 - 24 MHz
                                    2 - 12 MHz
                                    3 - 6 MHz
                                    4 - 3 MHz
                                    5 - 1.5 MHz
                                    6 - 750 kHz
                                    7 - 375 kHz

Returns: 0 if successful; error code otherwise)
```

Mcp2221_SetClockSettings

```
int Mcp2221_SetClockSettings(void*handle, unsigned char whichToSet, unsigned char dutyCycle,
                             unsigned char clockDivider)
================================================================================================
Description: Sets the duty cycle and clock divider values for the clock out pin (if configured for
                this operation).

Parameters:
 Inputs:
  (void*)          handle       - the handle for the device
  (unsigned char) whichToSet    - 0 to write Flash settings, >0 to write SRAM (runtime) settings
  (unsigned char) dutyCycle     - value of the duty cycle of the waveform on the clock pin
                                     0 - 0 %
                                     1 - 25 %
                                     2 - 50 %
                                     3 - 75 %
  (unsigned char) clockDivider - value of the clock divider. The value provided is a power of 2.
                The 48Mhz internal clock is divided by 2^value to obtain the output waveform
                frequency. The correspondence between the divider values and output frequencies
                are as follows:
                                     1 - 24 MHz
                                     2 - 12 MHz
                                     3 - 6 MHz
                                     4 - 3 MHz
                                     5 - 1.5 MHz
                                     6 - 750 kHz
                                     7 - 375 kHz

Returns: 0 if successful; error code otherwise
```

Mcp2221_GetDacVref

```
int Mcp2221_GetDacVref(void*handle, unsigned char whichToGet, unsigned char* dacVref)
================================================================================================
Description: Gets the DAC voltage reference.

Parameters:
 Inputs:
  (void*)          handle     - the handle for the device
  (unsigned char)  whichToGet - 0 to read Flash settings, >0 to read SRAM (runtime) settings

 Outputs:
  (unsigned char*) dacVref    - The voltage reference for the DAC:
                                    0 - Vdd
                                    1 - 1.024 V
                                    2 - 2.048 V
                                    3 - 4.096 V

Returns: 0 if successful; error code otherwise
```

Mcp2221_SetDacVref

```
int Mcp2221_SetDacVref(void*handle, unsigned char whichToSet, unsigned char dacVref)
================================================================================================
Description: Sets the DAC voltage reference.

Parameters:
 Inputs:
  (void*)          handle     - the handle for the device
  (unsigned char) whichToSet - 0 to write Flash settings, >0 to write SRAM (runtime) settings
  (unsigned char) dacVref    - The voltage reference for the DAC:
                                  0 - Vdd
                                  1 - 1.024 V
                                  2 - 2.048 V
                                  3 - 4.096 V

Returns: 0 if successful; error code otherwise
```

Mcp2221_GetAdcData

```
int Mcp2221_GetAdcData(void* handle, unsigned int* adcDataArray)
================================================================================
Description: Reads the ADC data for all 3 analog pins.

Parameters:
 Inputs:
  (void*)          handle       - the handle for the device.
  (unsigned int*) adcDataArray - Array containing the ADC values. Entry 0 will contain the value
                                  for ADC1, entry 1 - ADC2, entry 2 - ADC3

Returns: 0 if successful; error code otherwise

NOTE: the array must have a minimum length of 3.
```

Mcp2221_GetAdcVref

```
int Mcp2221_GetAdcVref(void*handle, unsigned char whichToGet, unsigned char* adcVref)
================================================================================
Description: Gets the DAC voltage reference.

Parameters:
 Inputs:
  (void*)          handle     - the handle for the device
  (unsigned char) whichToGet - 0 to read Flash settings, >0 to read SRAM (runtime) settings

 Outputs:
  (unsigned char*) adcVref    - The voltage reference for the ADC:
                                  0 - Vdd
                                  1 - 1.024 V
                                  2 - 2.048 V
                                  3 - 4.096 V

Returns: 0 if successful; error code otherwise
```

Mcp2221_SetAdcVref

```
int Mcp2221_SetAdcVref(void*handle, unsigned char whichToSet, unsigned char adcVref)
================================================================================
Description: Sets the ADC voltage reference.

Parameters:
 Inputs:
  (void*)          handle     - the handle for the device
  (unsigned char) whichToSet - 0 to write Flash settings, >0 to write SRAM (runtime) settings
  (unsigned char) dacVref    - The voltage reference for the ADC:
                               0 - Vdd
                               1 - 1.024 V
                               2 - 2.048 V
                               3 - 4.096 V

Returns: 0 if successful; error code otherwise
```

Mcp2221_GetDacValue

```
int Mcp2221_GetDacValue(void*handle, unsigned char whichToGet, unsigned char* dacValue)
================================================================================
Description: Gets the DAC value.

Parameters:
 Inputs:
  (void*)          handle     - the handle for the device
  (unsigned char) whichToGet - 0 to read Flash settings, >0 to read SRAM (runtime) settings

 Outputs:
  (unsigned char*) dacValue  - The DAC output value. Valid range is between 0 and 31.

Returns: 0 if successful; error code otherwise
```

Mcp2221_SetDacValue

```
int Mcp2221_SetDacValue(void*handle, unsigned char whichToSet, unsigned char dacValue)
================================================================================
Description: Sets the DAC value.

Parameters:
 Inputs:
  (void*)          handle     - the handle for the device
  (unsigned char) whichToSet - 0 to write Flash settings, >0 to write SRAM (runtime) settings
  (unsigned char) dacValue   - The DAC output value. Valid range is between 0 and 31.

Returns: 0 if successful; error code otherwise
```

Mcp2221_GetGpioSettings

```
int Mcp2221_GetGpioSettings(void* handle, unsigned char whichToGet, unsigned char* pinFunctions,
                            unsigned char* pinDirections, unsigned char* outputValues)
================================================================================================
Description: Gets the GPIO settings.

Parameters:
 Inputs:
  (void*)         handle     - the handle for the device
  (unsigned char) whichToGet - 0 to read Flash settings, >0 to read SRAM (runtime) settings

 Outputs:
  (unsigned char*) pinFunctions - Array containing the values for the pin functions.
                                  pinFunction[i] will contain the value for pin GP"i.
                                  Possible values: 0 to 3. 0 - GPIO, 1 - Dedicated function, 2 -
                                  alternate function 0, 3 - alternate function 1, 4 - alternate
                                  function 2.

                     GP0: 0 - GPIO       GP1: 0 - GPIO          GP2: 0 - GPIO    GP3: 0 - GPIO
                          1 - SSPND           1 - Clock Out          1 - USBCFG       1 - LED I2C
                          2 - LED UART RX     2 - ADC1               2 - ADC2         2 - ADC3
                          3 - LED UART TX     3 - DAC1               3 - DAC2
                                              4 - Interrupt detection

  (unsigned char*) pinDirections - Array containing the pin direction of the IO pins.
                                           0 - output
                                           1 - input
  (unsigned char*) outputValues  - Array containing the value present on the output pins.
                                           0 - logic low
                                           1 - logic high

Returns: 0 if successful; error code otherwise

NOTE: all output arrays must have a minimum length of 4.
```

Mcp2221_SetGpioSettings

```
int int Mcp2221_SetGpioSettings(void* handle, unsigned char whichToSet,
                                unsigned char* pinFunctions, unsigned char* pinDirections,
                                unsigned char* outputValues)
=============================================================================================
Description: Sets the GPIO settings.

Parameters:
 Inputs:
  (void*)          handle       - the handle for the device
  (unsigned char)  whichToSet   - 0 to write Flash settings, >0 to read SRAM (runtime) settings
  (unsigned char*) pinFunctions - Array containing the values for the pin functions.
                                  pinFunction[i] will contain the value for pin GP"i". Possible
                                  values: 0 to 3. 0 - GPIO, 1 - Dedicated function, 2 - alternate
                                  function 0, 3 - alternate function 1, 4 - alternate function 2,
                                  0xFF - leave the pin unchanged.

                    GP0: 0 - GPIO       GP1: 0 - GPIO           GP2: 0 - GPIO    GP3: 0 - GPIO
                         1 - SSPND           1 - Clock Out           1 - USBCFG       1 - LED I2C
                         2 - LED UART RX     2 - ADC1                2 - ADC2         2 - ADC3
                         3 - LED UART TX     3 - DAC1                3 - DAC2
                                             4 - Interrupt detection

  (unsigned char*) pinDirections - Array containing the pin direction of the IO pins.
                                           0 - output
                                           1 - input
                                           0xff - leave unchanged
  (unsigned char*) outputValues  - Array containing the value present on the output pins.
                                           0 - logic low
                                           1 - logic high
                                           0xff - leave unchanged

Returns: 0 if successful; error code otherwise

NOTE: all arrays must have a minimum length of 4.
```

Mcp2221_GetGpioValues

```
int Mcp2221_GetGpioValues(void* handle, unsigned char* gpioValues)
================================================================================
Description: Gets the GPIO pin values.

Parameters:
 Inputs:
  (void*) handle - the handle for the device

 Outputs:
  (unsigned char*) gpioValues - Array containing the value present on the IO pins.
                                          0 - logic low
                                          1 - logic high
                                          0xEE - GPx not set for GPIO operation

Returns: 0 if successful; error code otherwise

NOTE: the output array must have a minimum length of 4.
```

Mcp2221_SetGpioValues

```
int Mcp2221_SetGpioValues(void* handle, unsigned char* gpioValues)
================================================================================
Description: Sets the runtime GPIO pin values.  Sets the runtime GPIO pin directions; flash values
                are not changed.
Parameters:
 Inputs:
  (void*) handle - the handle for the device
  (unsigned char*) gpioValues - Array containing the value of the output IO pins.
                                          0 - logic low
                                          1 - logic high
                                          0xFF - no change

Returns: 0 if successful; error code otherwise

NOTE: the output array must have a minimum length of 4.
```

Mcp2221_GetGpioDirection

```
int Mcp2221_GetGpioDirection(void* handle, unsigned char* gpioDir)
================================================================================
Description: Gets the GPIO pin directions.

Parameters:
 Inputs:
  (void*) handle - the handle for the device

 Outputs:
  (unsigned char*) gpioDir - Array containing the direction of an IO pin.
                                          0 - output
                                          1 - input
                                          0xEF - GPx not set for GPIO operation

Returns: 0 if successful; error code otherwise

NOTE: the output array must have a minimum length of 4.
```

Mcp2221_SetGpioDirection

```
int Mcp2221_SetGpioDirection(void* handle, unsigned char* gpioDir)
================================================================================
Description: Sets the runtime GPIO pin directions; flash values are not changed.

Parameters:
 Inputs:
  (void*)          handle  - the handle for the device
  (unsigned char*) gpioDir - Array containing the direction of an IO pin. gpioDir[i] will set pin
                             "i"
                                          0 - output
                                          1 - input
                                          0xff - no change

Returns: 0 if successful; error code otherwise

NOTE: the output array must have a minimum length of 4.
```

Security

Mcp2221_GetSecuritySetting

```
int Mcp2221_GetSecuritySetting(void* handle, unsigned char* securitySetting)
================================================================================
Description: Gets the state of flash protection for the device

Parameters:
 Inputs:
  (void*) handle - the handle for the device
Outputs:
  (unsigned char*) securitySetting - the value of the chip security option
                                     0 - unsecured
                                     1 - password protected
                                     2 - permanently locked

Returns: 0 if successful; error code otherwise
```

Mcp2221_SetSecuritySettings

```
int Mcp2221_SetSecuritySettings(void* handle, unsigned char securitySetting,
                                unsigned char* currentPassword, unsigned char* newPassword)
================================================================================
Description: Sets the state of flash protection for the device

Parameters:
 Inputs:
  (void*)         handle          - the handle for the device
  (unsigned char) securitySetting - the value of the chip security option. If any other values are
                                    used, the E_ERR_INVALID_PARAMETER (-4) error is returned.
                                     0 - disable password protection
                                     1 - enable password protection
                                     0xff - change current password
  (char*)         currentPassword - the value for the currently set password. This is used for
                                    when the password "disable" or "change" operations are taking
                                    place.
  (char*)         newPassword     - the value for the new password. Must be an 8 character string.
                                    This is only for the "enable" or "change" operations.

Returns: 0 if successful; error code otherwise
```

Mcp2221_SetPermanentLock

```
int Mcp2221_SetPermanentLock(void* handle)
================================================================================
Description: Permanently lock the device flash settings -- this action CAN'T be undone.

Parameters:
 Inputs:
  (void*) handle - the handle for the device to be locked

Returns: 0 if successful; error code otherwise

!!! WARNING !!! -- USE THIS FUNCTION WITH GREAT CAUTION.  THE CHIP FLASH SETTINGS (boot-up
                   defaults) CANNOT BE CONFIGURED AFTER THIS FUNCTION HAS BEEN INVOKED!!
```

Mcp2221_SendPassword

```
int Mcp2221_SendPassword(void* handle, char* password){
================================================================================
Description: Sends the access password to the device.

Parameters:
 Inputs:
  (void*) handle   - the handle for the device
  (char*) password - the password that will be sent to the device to unlock writing to flash. Must
                     be an 8 character string.

Returns: 0 if successful; error code otherwise

NOTE: If 3 flash writes are attempted with an incorrect password, the chip won't accept any more
passwords. This function doesn't validate the password, it just sends it to the device. The
password is checked only during a flash write.
```

Managed Function List

M_Mcp2221_GetLibraryVersion

```
String^ M_Mcp2221_GetLibraryVersion()
================================================================================
Description: Returns the version number of the DLL

Returns: String containing the library version

Use Marshal.GetLastWin32Error() to determine the error code if the function fails.
```

M_Mcp2221_GetConnectedDevices

```
int M_Mcp2221_GetConnectedDevices(unsigned int vid, unsigned int pid, unsigned int% noOfDevs)
================================================================================
Description: Gets the number of connected MCP2221s with the provided VID & PID.

Parameters:
 Inputs:
  (unsigned int) vid - The vendor id of the MCP2221 devices to count
  (unsigned int) pid - The product id of the MCP2221 devices to count

 Outputs:
  (unsigned int%) noOfDevs - the number of connected MCP2221s matching the provided VID&PID

Returns (int) - 0 for success; error code otherwise.
```

Device Connection

M_Mcp2221_OpenByIndex

```
IntPtr M_Mcp2221_OpenByIndex(unsigned int vid, unsigned int pid, unsigned int index)
================================================================================
Description: Attempt to open a connection with a MCP2221 with a specific index.

Parameters:
 Inputs:
  (unsigned int) vid   - The vendor ID of the MCP2221 to connect to.
                         (Microchip default = 0x4D8)
  (unsigned int) pid   - The product ID of the MCP2221 to connect to.
                         (Microchip default = 0xDD)
  (unsigned int) index - The index of the MCP2221 to connect to. This value ranges from 0 to n-1,
                         where n is the number of connected devices. This value can be obtained
                         from "GetConnectedDevices"

Returns: (IntPtr) - the handle value if the connection was successfully opened or
                    INVALID_HANDLE_VALUE if not.

NOTE: If the operation failed, call the M_Mcp2221_GetLastError() method to get the error code.
```

M_Mcp2221_OpenBySN

```
IntPtr M_Mcp2221_OpenBySN(unsigned int vid, unsigned int pid, String^ serialNo)
================================================================================
Description: Attempt to open a connection with a MCP2221 with a specific serial number.

Parameters:
 Inputs:
  (unsigned int) vid - The vendor ID of the MCP2221 to connect to.(Microchip default = 0x4D8)
  (unsigned int) pid - The product ID of the MCP2221 to connect to.(Microchip default = 0xDD)
  (String^) serialNo - The serial number of the MCP2221 we want to connect to. Maximum 10
                       character value.

Returns: (IntPtr) - the handle value if the connection was successfully opened or
                INVALID_HANDLE_VALUE if not.

NOTE: If the operation failed, call the M_Mcp2221_GetLastError() method to get the error code.
If a connection to a matching device is already open, the function will fail with the "device not
found" error.
```

M_Mcp2221_Close

```
int M_Mcp2221_Close(IntPtr handle)
================================================================================
Description: Attempt to close a connection to a MCP2221.

Parameters:
 (IntPtr ) handle - The handle for the device we'll close the connection to.

Returns: (int) - 0 for success; error code otherwise.
```

M_Mcp2221_CloseAll

```
int M_Mcp2221_CloseAll()
================================================================================
Description: Attempt to close all the currently opened MCP2221 connections.
             If successful, all existing handles will be set to INVALID_HANDLE_VALUE

Returns: (int) - 0 for success or the number of devices that failed to close.
```

M_Mcp2221_Reset

```
int M_Mcp2221_Reset(IntPtr handle)
================================================================================
Description: Reset the MCP2221 and close its associated handle.

Parameters:
 (IntPtr) handle - The handle for the device we'll reset. If successful, the handle will also be
                   closed.

Returns: (int) - 0 for success; error code otherwise.
```

M_Mcp2221_GetLastError

```
int M_Mcp2221_GetLastError()
================================================================================
Description: Gets the last error value. Used only for the Open methods.

Returns: (int) - The value for the last error code.
```

I2C/SMBus

M_Mcp2221_SetSpeed

```
int M_Mcp2221_SetSpeed(IntPtr handle, unsigned int speed)
================================================================================
Description: Set the communication speed for I2C/SMBus operations.

Parameters:
 (IntPtr)       handle - The handle for the device.
 (unsigned int) speed  - the communication speed. Accepted values are between 46875 and 500000.

Returns: (int) - 0 for success; error code otherwise.
```

M_Mcp2221_SetAdvancedCommParams

```
int M_Mcp2221_SetAdvancedCommParams(IntPtr handle, BYTE timeout, BYTE maxRetries)
================================================================================
Description: Set the time the MCP2221 will wait after sending the "read" command before trying to
               read back the data from the I2C/SMBus slave and the maximum number of retries if
               data couldn't be read back.

Parameters:
 (IntPtr) handle     - The handle for the device.
 (BYTE)   timeout    - amount of time (in ms) to wait for the slave to send back data.Default 3ms.
 (BYTE)   maxRetries - the maximum amount of times we'll try to read data back from a slave.
                       default = 5

Returns: (int) - 0 for success; error code otherwise.
```

M_Mcp2221_I2cCancelCurrentTransfer

```
int M_Mcp2221_I2cCancelCurrentTransfer(IntPtr handle)
================================================================================
Description: Cancel the current I2C/SMBus transfer

Parameters:
 Inputs:
  (IntPtr) handle - The handle for the device.

Returns: (int) - 0 for success; error code otherwise.
```

M_Mcp2221_I2cRead

```
int M_Mcp2221_I2cRead(IntPtr handle, UINT32 bytesToRead, BYTE slaveAddress, BYTE  use7bitAddress,
                      array<System::Byte>^ i2cRxData)
================================================================================
Description: Read I2C data from a slave.

Parameters:
 Inputs:
  (IntPtr) handle         - The handle for the device.
  (UINT32) bytesToRead    - the number of bytes to read from the slave. Valid range is between 1
                            and 65535.
  (BYTE)   slaveAddress   - 7bit or 8bit I2C slave address, depending on the value of the
                            "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of the address
                            is set to 1 inside the function.
  (BYTE)   use7bitAddress - if > 0 - 7 bit address will be used for the slave. If set to 0 - 8 bit
                            is used.

 Outputs:
  (array<System::Byte>^) i2cRxData - buffer that will contain the data bytes read from the slave.

Returns: (int) - 0 for success; error code otherwise.

NOTE: if the "M_Mcp2221_SetSpeed" function has not been called for the provided handle, the
default speed of 100kbps will be configured and used. Otherwise, the speed will not be
reconfigured.
```

M_Mcp2221_I2cWrite

```
int M_Mcp2221_I2cWrite(IntPtr handle, UINT32 bytesToWrite, Byte slaveAddress, BYTE use7bitAddress,
                       array<System::Byte>^ i2cTxData)
================================================================================================
Description: Write I2C data to a slave.

Parameters:
 Inputs:
  (IntPtr)                handle         - The handle for the device.
  (UINT32)                bytesToWrite   - the number of bytes to write to the slave. Valid range
                                           is between 0 and 65535.
  (Byte)                  slaveAddress   - 7bit or 8bit I2C slave address, depending on the value
                                           of the "use7bitAddress" flag. For 8 bit addresses, the
                                           R/W LSB of the address is set to 0 inside the function.
  (BYTE)                  use7bitAddress - if > 0 - 7 bit address will be used for the slave. If 0
                                           - 8 bit is used.
  (array<System::Byte>^) i2cTxData       - buffer that will contain the data bytes to be sent to
                                           the slave.

Returns: (int) - 0 for success; error code otherwise.

NOTE: if the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

M_Mcp2221_I2cWriteNoStop

```
int M_Mcp2221_I2cWriteNoStop(IntPtr handle, UINT32 bytesToWrite, Byte slaveAddress, BYTE
                             use7bitAddress, array<System::Byte>^ i2cTxData)
================================================================================================
Description: Write I2C data to a slave without sending the STOP bit.

Parameters:
Inputs: (IntPtr) handle                   - The handle for the device.
        (UINT32) bytesToWrite             - the number of bytes to write to the slave. Valid range
                                            is between 0 and 65535.
        (Byte) slaveAddress               - 7bit or 8bit I2C slave address, depending on the value
                                            of the "use7bitAddress" flag. For 8 bit addresses, the
                                            R/W LSB of the address is set to 0 inside the function.
        (BYTE) use7bitAddress             - if > 0 7 bit address will be used for the slave. If 0 8
                                            bit is used.
        (array<System::Byte>^) i2cTxData - buffer that will contain the data bytes to be sent to
                                            the slave.

Returns: (int) - 0 for success; error code otherwise.

NOTE: 1. The speed must be set via the "Mcp2221_SetSpeed" function before using this method. If
      the speed has not been set an error will be returned.
      2. The SMBus Process Call command can be formed using M_Mcp2221_I2cWriteNoStop followed by
      M_Mcp2221_I2cReadRestart
```

M_Mcp2221_I2cReadRestart

```
int M_Mcp2221_I2cReadRestart(IntPtr handle, UINT32 bytesToRead, BYTE slaveAddress, BYTE
                             use7bitAddress, array<System::Byte>^ i2cRxData)
================================================================================================
Description: Read I2C data from a slave starting with a Repeated START.

Parameters:
    Inputs: (IntPtr) handle                - The handle for the device.
            (UINT32) bytesToRead           - the number of bytes to read from the slave. Valid
                                             range is between 1 and 65535.
            (BYTE) slaveAddress            - 7bit or 8bit I2C slave address, depending on the value
                                             of the "use7bitAddress" flag.For 8 bit addresses, the
                                             R/W LSB of the address is set to 1 inside the
                                             function.
            (BYTE) use7bitAddress          - if > 0 7 bit address will be used for the slave. If
                                             set to 0 8 bit is used.

Outputs: (array<System::Byte>^) i2cRxData - buffer that will contain the data bytes read from the
                                            slave.

Returns: (int) - 0 for success; error code otherwise.

NOTE: 1. The speed must be set via the "Mcp2221_SetSpeed" function before using this method. If
      the speed has not been set an error will be returned.
      2. The SMBus Process Call command can be formed using M_Mcp2221_I2cWriteNoStop followed by
      M_Mcp2221_I2cReadRestart
```

M_Mcp2221_I2cWriteRestart

```
int M_Mcp2221_I2cWriteRestart(IntPtr handle, UINT32 bytesToWrite, Byte slaveAddress, BYTE
                    use7bitAddress, array<System::Byte>^ i2cTxData)
================================================================================================
Description:  Write I2C data to a slave starting with a Repeated START.

Parameters:
Inputs: (IntPtr) handle                   - The handle for the device.
        (UINT32) bytesToWrite             - the number of bytes to write to the slave. Valid range
                                            is between 0 and 65535.
        (Byte) slaveAddress               - 7bit or 8bit I2C slave address, depending on the value
                                            of the "use7bitAddress" flag. For 8 bit addresses, the
                                            R/W LSB of the address is set to 0 inside the function.
        (BYTE) use7bitAddress             - if > 0 7 bit address will be used for the slave. If 0 8
                                            bit is used.
        (array<System::Byte>^) i2cTxData - buffer that will contain the data bytes to be sent to
                                            the slave.

Returns: (int) - 0 for success; error code otherwise.

NOTE: The speed must be set via the "Mcp2221_SetSpeed" function before using this method. If the
      speed has not been set an error will be returned.
```

M_Mcp2221_SmbusSendByte

```
int M_Mcp2221_SmbusSendByte(IntPtr handle, BYTE slaveAddress, BYTE use7bitAddress, BYTE usePec,
                            BYTE data)
================================================================================================
Description: SMBus send byte. Sends one data byte.

Parameters:
 Inputs:
  (IntPtr) handle         - The handle for the device.
  (BYTE)   slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value of the
                            "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of the address
                            is set to 0 inside the function.
  (BYTE)   use7bitAddress - if > 0 - 7 bit address will be used for the slave. If 0 - 8 bit is
                            used.
  (BYTE)   usePec         - if > 0 Packet Error Checking (PEC) will be used. A PEC byte containing
                            the CRC8 value for the sent message is appended after the data byte.
  (BYTE)   data           - The data byte.

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

M_Mcp2221_SmbusReceiveByte

```
int M_Mcp2221_SmbusReceiveByte(IntPtr handle, BYTE slaveAddress, BYTE use7bitAddress, BYTE usePec,
                               BYTE% readByte)
================================================================================================
Description: SMBus Receive Byte. Read one data byte back.

Parameters:
 Inputs:
  (IntPtr) handle         - The handle for the device.
  (BYTE)   slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value of the
                            "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of the address
                            is set to 1 inside the function.
  (BYTE)   use7bitAddress - if > 0 - 7 bit address will be used for the slave. If 0 - 8 bit is
                            used.
  (BYTE)   usePec         - if > 0 - Packet Error Checking (PEC) will be used.

 Outputs:
  (BYTE%) readByte - The data byte received from the slave

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

M_Mcp2221_SmbusWriteByte

```
int M_Mcp2221_SmbusWriteByte(IntPtr handle, BYTE slaveAddress, BYTE use7bitAddress, BYTE usePec,
                             BYTE command, BYTE data)
================================================================================================
Description: SMBus write byte. The first byte of a Write Byte operation is the command code. The
               next one is the data to be written.

Parameters:
 Inputs:
  (IntPtr) handle         - The handle for the device.
  (BYTE)   slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value of the
                            "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of the address
                            is set to 0 inside the function.
  (BYTE)   use7bitAddress - if > 0 - 7 bit address will be used for the slave. If 0 - 8 bit is
                            used.
  (BYTE)   usePec         - if > 0 Packet Error Checking (PEC) will be used. A PEC byte containing
                            the CRC8 value for the sent message is appended after the data byte.
  (BYTE)   command        - The command code byte.
  (BYTE)   data           - The data byte.

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

M_Mcp2221_SmbusReadByte

```
int M_Mcp2221_SmbusReadByte(IntPtr handle, BYTE slaveAddress, BYTE use7bitAddress, BYTE usePec,
                            BYTE command, BYTE% readByte)
================================================================================================
Description: SMBus Read Byte. First Write the command byte to the slave, then read one data byte
               back.

Parameters:
 Inputs:
  (IntPtr) handle         - The handle for the device.
  (BYTE)   slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value of the
                            "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of the address
                            is set to 1 inside the function.
  (BYTE)   use7bitAddress - if > 0 - 7 bit address will be used for the slave. If 0 - 8 bit is
                            used.
  (BYTE)   usePec         - if > 0 - Packet Error Checking (PEC) will be used.
  (BYTE)   command        - The command code byte.

 Outputs:
  (BYTE%) readByte - The data byte received from the slave

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

M_Mcp2221_SmbusWriteWord

```
int M_Mcp2221_SmbusWriteWord(IntPtr handle, BYTE slaveAddress, BYTE use7bitAddress, BYTE usePec,
                             BYTE command, array<System::Byte>^ data)
================================================================================================
Description: SMBus write word. The first byte of a Write Byte operation is the command code,
                followed by the data_byte_low then data_byte_high.

Parameters:
 Inputs:
  (IntPtr)                handle         - The handle for the device.
  (BYTE)                  slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value
                                           of the "use7bitAddress" flag. For 8 bit addresses, the
                                           R/W LSB of the address is set to 0 inside the function.
  (BYTE)                  use7bitAddress - if > 0 - 7 bit address will be used for the slave. If 0
                                           - 8 bit is used.
  (BYTE)                  usePec         - if > 0 - Packet Error Checking (PEC) will be used. A PEC
                                           byte containing the CRC8 value for the sent message is
                                           appended after the data byte.
  (BYTE)                  command        - The command code byte.
  (array<System::Byte>^)  data           - Array containing the low and high data bytes to be sent
                                           to the slave.
                                               data[0] will be considered the data_byte_low
                                               data[1] will be considered the data_byte_high

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

M_Mcp2221_SmbusReadWord

```
int M_Mcp2221_SmbusReadWord(IntPtr handle, BYTE slaveAddress, BYTE use7bitAddress, BYTE usePec,
                            BYTE command, array<System::Byte>^ readData)
=============================================================================================
Description: SMBus Read Word. First Write the command byte to the slave, then read one data byte
               back.

Parameters:
 Inputs:
  (IntPtr) handle         - The handle for the device.
  (BYTE)   slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value of the
                            "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of the address
                            is set to 1 inside the function.
  (BYTE)   use7bitAddress - if > 0 - 7 bit address will be used for the slave. If 0 - 8 bit is
                            used.
  (BYTE)   usePec         - if > 0 - Packet Error Checking (PEC) will be used.
  (BYTE)   command        - The command code byte.

 Outputs:
  (array<System::Byte>^) readData - Buffer that will store the read data word.
                                    readData[0] - data_byte_low
                                    readData[1] - data_byte_high

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

M_Mcp2221_SmbusBlockWrite

```
int M_Mcp2221_SmbusBlockWrite(IntPtr handle, BYTE slaveAddress, BYTE use7bitAddress, BYTE usePec,
                              BYTE command, BYTE byteCount, array<System::Byte>^ data)
================================================================================================
Description: SMBus Block Write. The first byte of a Block Write operation is the command code,
                followed by the number of data bytes, then data bytes.

Parameters:
 Inputs:
  (IntPtr)                handle          - The handle for the device.
  (BYTE)                  slaveAddress    - 7bit or 8bit SMBus slave address, depending on the value
                                            of the "use7bitAddress" flag. For 8 bit addresses, the
                                            R/W LSB of the address is set to 0 inside the function.
  (BYTE)                  use7bitAddress - if > 0 - 7 bit address will be used for the slave. If 0
                                            - 8 bit is used.
  (BYTE)                  usePec          - if > 0 - Packet Error Checking (PEC) will be used. A PEC
                                            byte containing the CRC8 for the sent message is
                                            appended after the data byte.
  (BYTE)                  command         - The command code byte.
  (BYTE)                  byteCount       - the number of data bytes that will be sent to the slave.
                                            Valid range is between 0 and 255 bytes, conforming to
                                            the smbus v3 specification.
  (array<System::Byte>^) data             - Array containing the data bytes to be sent to the slave.

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

M_Mcp2221_SmbusBlockRead

```
int M_Mcp2221_SmbusBlockRead(IntPtr handle, BYTE slaveAddress, BYTE use7bitAddress, BYTE usePec,
                             BYTE command, BYTE byteCount, array<System::Byte>^ readData)
================================================================================================
Description: SMBus Block Read.
Parameters:
 Inputs:
  (IntPtr) handle         - The handle for the device.
  (BYTE)   slaveAddress   - 7bit or 8bit SMBus slave address, depending on the value of the
                            "use7bitAddress" flag. For 8 bit addresses, the R/W LSB of the address
                            is set to 1 inside the function.
  (BYTE)   use7bitAddress - if > 0- 7 bit address will be used for the slave. If 0 - 8 bit is
                            used.
  (BYTE)   usePec         - if > 0 Packet Error Checking (PEC) will be used. The CRC8 values is
                            computed for the SMBus packet compared with the PEC byte sent by the
                            slave. If the two values differ the function returns an error code.
  (BYTE)   command        - The command code byte.
  (BYTE)   byteCount      - the number of data bytes that the slave will send to the master. Valid
                            range is between 1 and 255 bytes, conforming to the smbus v3
                            specification. If there is a mismatch between this value and the
                            byteCount the slave reports that it will send, an error will be
                            returned.

 Outputs:
  (array<System::Byte>^) data - Array containing the data bytes read from the slave. If PEC is
                                used, the last data byte will be the PEC byte received from the slave
                                so the array should have a length of n+1, where n is the block size.

Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
speed of 100kbps will be configured and used. Otherwise, the speed will not be reconfigured.
```

M_Mcp2221_SmbusBlockWriteBlockReadProcessCall

```
M_Mcp2221_SmbusBlockWriteBlockReadProcessCall(IntPtr handle, BYTE slaveAddress, BYTE
                                use7bitAddress, BYTE usePec, BYTE command, BYTE writeByteCount,
                                array<System::Byte>^ writeData, BYTE readByteCount,
                                array<System::Byte>^ readData)
=============================================================================================
Description: SMBus Block Write Block Read Process Call.

Parameters:
Inputs:
    (IntPtr) handle                 - The handle for the device.
    (BYTE) slaveAddress             - 7bit or 8bit SMBus slave address, depending on the value of
                                      the "use7bitAddress" flag. For 8 bit addresses, the R/W LSB
                                      of the address is set to 0 inside the function.
    (BYTE) use7bitAddress           - if >0, 7 bit address will be used for the slave. If 0, 8
                                      bit is used.
    (BYTE) usePec                   - if >0, Packet Error Checking (PEC) will be used. The CRC8
                                      values is computed for the SMBus packet and compared with
                                      the PEC byte sent by the slave. If the two values differ
                                      the function returns an error code.
    (BYTE) command                  - The command code byte.
    (BYTE) writeByteCount           - the number of data bytes that will be sent to the slave.
                                      The total data payload must not exceed 255 bytes
                                      (writeByteCount + readByteCound <= 255) and writeByteCount
                                      > 0
    (array<System::Byte>^) writeData - array containing the data bytes to be sent to the slave.
    (BYTE) readByteCount            - the number of data bytes that the slave will send to the
                                      master. If there is a mismatch between this value and the
                                      readByteCount the slave reports that it will send, an error
                                      will be returned. The total data payload must not exceed
                                      255 bytes (writeByteCount + readByteCound <= 255) and
                                      readByteCount > 0

Outputs:
    (array<System::Byte>^) readData  - Array containing the data bytes read from the slave. If PEC
                                      is used, the last data byte will be the PEC byte received
                                      from the slave so the array should have a length of n+1,
                                      where n is the readByteCount size.
```

```
Returns: (int) - 0 for success; error code otherwise.

NOTE: If the "Mcp2221_SetSpeed" function has not been called for the provided handle, the default
      speed of 100kbps will be configured and used.Otherwise, the speed will not be
      reconfigured.speed of 100kbps will be configured and used.
```

USB settings and Device Information

M_Mcp2221_GetManufacturerDescriptor

```
String^ M_Mcp2221_GetManufacturerDescriptor(IntPtr handle)
================================================================================
Description: Returns the manufacturer descriptor string of the device.

Parameters
 Inputs: (IntPtr) handle - the handle of the device

Returns: String containing the manufacturer descriptor
Use Marshal.GetLastWin32Error() to determine the error code if the function fails.

Note: the output string can contain up to 30 characters
```

M_Mcp2221_SetManufacturerDescriptor

```
int M_Mcp2221_SetManufacturerDescriptor(IntPtr handle, String^ manufacturerString)
================================================================================
Description: Sets the manufacturer descriptor string of the device.

Inputs:
  (IntPtr)  handle             - handle for the device
  (String^) manufacturerString - will contain the value of the USB Manufacturer Descriptor String.

Returns: (int) 0 for success, error code otherwise

Note: the input string can contain up to 30 characters
```

M_Mcp2221_GetProductDescriptor

```
String^ M_Mcp2221_GetProductDescriptor(IntPtr handle)
================================================================================
Description: Returns the product descriptor string of the device.

Parameters
 Inputs:
  (IntPtr) handle - the handle of the device

Returns: String containing the product descriptor
Use Marshal.GetLastWin32Error() to determine the error code if the function fails.

Note: the output string can contain up to 30 characters
```

M_Mcp2221_SetProductDescriptor

```
int M_Mcp2221_SetProductDescriptor(IntPtr handle, String^ productString)
================================================================================
Description: Sets the product descriptor string of the device.

Inputs:
 (IntPtr)  handle        - handle for the device
 (String^) productString - the string to be set

Returns: (int) 0 for success, error code otherwise

Note: The input string can contain a maximum of 30 characters.
```

M_Mcp2221_GetSerialNumberDescriptor

```
String^ M_Mcp2221_GetSerialNumberDescriptor(IntPtr handle)
================================================================================
Description: Returns the serial number descriptor string of the device.

Parameters
 Inputs:
  (IntPtr) handle - the handle of the device

Returns: String containing the Serial number descriptor. This can have up to 30 characters.
Use Marshal.GetLastWin32Error() to determine the error code if the function fails.
```

M_Mcp2221_SetSerialNumberDescriptor

```
int M_Mcp2221_SetSerialNumberDescriptor(IntPtr handle, String^ serialNumber)
================================================================================
Description: Sets the serial number descriptor string of the device.

Inputs:
 (IntPtr)  handle       - handle for the device
 (String^) serialNumber - the string to be set

Returns: (int) 0 for success, error code otherwise

Note: the input string can contain a maximum of 30 characters
```

M_Mcp2221_GetFactorySerialNumber

```
String^ M_Mcp2221_GetFactorySerialNumber(IntPtr handle)
================================================================================
Description: Returns the factory serial number of the device.

Parameters
 Inputs:
  (IntPtr) handle - handle for the device

Returns: String containing the serial number.
Use Marshal.GetLastWin32Error() to determine the error code if the function fails.

Note: the output string can contain a maximum of 30 characters
```

M_Mcp2221_GetVidPid

```
int M_Mcp2221_GetVidPid(IntPtr handle, unsigned int% vid, unsigned int% pid)
================================================================================
Description: Gets the VID and PID for the selected device.

Parameters:
 Inputs:
  (IntPtr) handle - the handle for the device

 Outputs:
  (unsigned int%) vid - The vendor id of the MCP2221 device
  (unsigned int%) pid - The product id of the MCP2221 device

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_SetVidPid

```
int M_Mcp2221_SetVidPid(IntPtr handle, unsigned int vid, unsigned int pid)
================================================================================
Description: Sets the VID and PID for the selected device.

Parameters:
 Inputs:
  (IntPtr)       handle - the handle for the device
  (unsigned int) vid    - The vendor id to be set
  (unsigned int) pid    - The product id to be set

Returns: 0 if successful; error code otherwise

NOTE: the new VID/PID values will take effect after a device reset.
```

M_Mcp2221_GetUsbPowerAttributes

```
int M_Mcp2221_GetUsbPowerAttributes(IntPtr handle, unsigned char% powerAttributes,
                                    unsigned int% currentReq)
================================================================================
Description: Gets the USB power attribute values.

Parameters:
 Inputs:
  (IntPtr) handle - the handle for the device

 Outputs:
  (unsigned char%) powerAttributes - the power attributes value from the USB descriptor.
                                     Bit meanings, based on the USB 2.0 spec:
                                       bit 7 - Reserved (Set to 1) (equivalent to Bus Powered)
                                       bit 6 - Self Powered
                                       bit 5 - Remote Wakeup
                                       bits 4..0 Reserved (reset to 0)
  (unsigned int%)  currentReq      - the requested current value (mA); This value is expressed in
                                     multiples of 2mA.
Returns: 0 if successful; error code otherwise
```

M_Mcp2221_SetUsbPowerAttributes

```
int M_Mcp2221_SetUsbPowerAttributes(IntPtr handle, unsigned char powerAttributes,
                                    unsigned int currentReq)
================================================================================
Description: Sets the USB power attribute values.

Parameters:
 Inputs:
  (IntPtr)        handle          - the handle for the device
  (unsigned char) powerAttributes - the power attributes value from the USB descriptor.
                                    Bit meanings, based on the USB 2.0 spec:
                                      bit 7 - Reserved (Set to 1) (equivalent to Bus Powered)
                                      bit 6 - Self Powered
                                      bit 5 - Remote Wakeup
                                      bits 4..0 Reserved (reset to 0)
  (unsigned int) currentReq       - the requested current value (mA); This value is expressed in
                                    multiples of 2mA.
Returns: 0 if successful; error code otherwise
```

M_Mcp2221_GetSerialNumberEnumerationEnable

```
int M_Mcp2221_GetSerialNumberEnumerationEnable(IntPtr handle, unsigned char% snEnumEnabled)
================================================================================
Description: Gets the status of the Serial number enumeration bit.

Parameters:
 Inputs:
  (IntPtr) handle - the handle for the device

 Outputs:
  (unsigned char%) snEnumEnabled - determines if the serial number descriptor will be used during
                                   the USB enumeration of the CDC interface. If 1 - the serial
                                   number descriptor is used; if 0 - no serial number descriptor
                                   will be present during enumeration.

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_SetSerialNumberEnumerationEnable

```
int M_Mcp2221_SetSerialNumberEnumerationEnable(IntPtr handle, unsigned char snEnumEnabled)
================================================================================
Description: Sets the status of the Serial number enumeration bit.

Parameters:
 Inputs:
  (IntPtr)         handle        - the handle for the device
  (unsigned char%) snEnumEnabled - determines if the serial number descriptor will be used during
                                   the USB enumeration of the CDC interface. If 1 - the serial
                                   number descriptor is used; if 0 - no serial number descriptor
                                   will be present during enumeration.

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_GetHardwareRevision

```
String^ M_Mcp2221_GetHardwareRevision(IntPtr handle)
================================================================================
Description: Reads the hardware revision value from the device.

Parameters:
 Inputs:
  (IntPtr) handle - the handle for the device.

Returns: String containing the hardware revision.
    Use Marshal.GetLastWin32Error() to determine the error code if the function fails.

NOTE: the output string must have a minimum length of 2.
```

M_Mcp2221_GetFirmwareRevision

```
String^ M_Mcp2221_GetFirmwareRevision(IntPtr handle)
================================================================================
Description: Reads the firmware revision value from the device.

Parameters:
 Inputs:
  (IntPtr) handle - the handle for the device.

Returns: String containing the firmware revision,
    Use Marshal.GetLastWin32Error() to determine the error code if the function fails.

NOTE: the output string must have a minimum length of 2.
```

Pin Functions

M_Mcp2221_GetInitialPinValues

```
int M_Mcp2221_GetInitialPinValues(IntPtr handle, unsigned char% ledUrxInitVal,
                                  unsigned char% ledUtxInitVal, unsigned char% ledI2cInitVal,
                                  unsigned char% sspndInitVal, unsigned char% usbCfgInitVal)
================================================================================
Description: Gets the initial values for the special function pins: LEDUARTRX, LEDUARTTX, LEDI2C,
               SSPND and USBCFG

Parameters:

 Inputs:
  (IntPtr) handle - the handle for the device

 Outputs:
  (unsigned char%) ledUrxInitVal - this value represents the logic level signaled when no Uart Rx
                                   activity takes place (inactive level)
  (unsigned char%) ledUtxInitVal - this value represents the logic level signaled when no Uart Tx
                                   activity takes place (inactive level)
  (unsigned char%) ledI2cInitVal - this value represents the logic level signaled when no I2C
                                   traffic occurs (inactive level)
  (unsigned char%) sspndInitVal  - this value represents the logic level signaled when the device
                                   is not in suspend mode (inactive level)
  (unsigned char%) usbCfgInitVal - this value represents the logic level signaled when the device
                                   is not usb configured (inactive level)

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_SetInitialPinValues

```
int M_Mcp2221_SetInitialPinValues(IntPtr handle, unsigned char ledUrxInitVal,
                                  unsigned char ledUtxInitVal, unsigned char ledI2cInitVal,
                                  unsigned char sspndInitVal, unsigned char usbCfgInitVal)
================================================================================================
Description: Sets the initial values for the special function pins: LEDUARTRX, LEDUARTTX, LEDI2C,
               SSPND and USBCFG

Parameters:
 Inputs:
  (IntPtr)          handle        - the handle for the device
  (unsigned char%) ledUrxInitVal - this value represents the logic level signaled when no Uart Rx
                                   activity takes place (inactive level)
  (unsigned char%) ledUtxInitVal - this value represents the logic level signaled when no Uart Tx
                                   activity takes place (inactive level)
  (unsigned char%) ledI2cInitVal - this value represents the logic level signaled when no I2C
                                   traffic occurs (inactive level)
  (unsigned char%) sspndInitVal  - this value represents the logic level signaled when the device
                                   is not in suspend mode (inactive level)
  (unsigned char%) usbCfgInitVal - this value represents the logic level signaled when the device
                                   is not usb configured (inactive level)

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_GetInterruptEdgeSetting

```
int M_Mcp2221_GetInterruptEdgeSetting(IntPtr handle, unsigned char whichToGet,
                                      unsigned char% interruptPinMode)
================================================================================
Description: Gets the interrupt pin trigger configuration.

Parameters:
 Inputs:
  (IntPtr)        handle     - the handle for the device
  (unsigned char) whichToGet - 0 to read Flash settings, >0 to read SRAM (runtime) settings

 Outputs:
  (unsigned char%) interruptPinMode - value representing which edge will trigger the interrupt
                                                  0 - none
                                                  1 - positive edge
                                                  2 - negative edge
                                                  3 - both
Returns: 0 if successful; error code otherwise
```

M_Mcp2221_SetInterruptEdgeSetting

```
int M_Mcp2221_SetInterruptEdgeSetting(IntPtr handle, unsigned char whichToSet,
                                      unsigned char interruptPinMode)
================================================================================
Description: Sets the interrupt pin trigger configuration.

Parameters:
 Inputs:
  (IntPtr)        handle           - the handle for the device
  (unsigned char) whichToSet       - 0 to write Flash settings, >0 to write SRAM (runtime)
                                     settings
  (unsigned char) interruptPinMode - value representing which edge will trigger the interrupt
                                                  0 - none
                                                  1 - positive edge
                                                  2 - negative edge
                                                  3 - both

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_ClearInterruptPinFlag

```
int M_Mcp2221_ClearInterruptPinFlag(IntPtr handle)
================================================================================================
Description: Clears the interrupt pin flag of a device.

Parameters:
 Inputs:
  (IntPtr) handle - the handle for the device for which the flag will be cleared.

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_GetInterruptPinFlag

```
M_Mcp2221_GetInterruptPinFlag(IntPtr handle, unsigned char% flagValue);
================================================================================================
Description: Reads the interrupt pin flag value of a device.

Parameters:
 Inputs:
  (IntPtr)         handle    - the handle for the device.
  (unsigned char%) flagValue - the value of the interrupt on change flag

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_GetClockSettings

```
int M_Mcp2221_GetClockSettings(IntPtr handle, unsigned char whichToGet, unsigned char% dutyCycle,
                               unsigned char% clockDivider)
================================================================================================
Description: Gets the duty cycle and clock divider values for the clock out pin (if configured for
               this operation).

Parameters:
 Inputs:
 (IntPtr)        handle     - the handle for the device
 (unsigned char) whichToGet - 0 to read Flash settings, >0 to read SRAM (runtime) settings

 Outputs:
  (unsigned char%) dutyCycle    - value of the duty cycle of the waveform on the clock pin
                                            0 - 0 %
                                            1 - 25 %
                                            2 - 50 %
                                            3 - 75 %
  (unsigned char%) clockDivider - value of the clock divider. The value provided is a power of 2.
                                  The 48Mhz internal clock is divided by 2^value to obtain the
                                  output waveform frequency. The correspondence between the
                                  divider values and output frequencies are as follows:
                                             1 - 24 MHz
                                             2 - 12 MHz
                                             3 - 6 MHz
                                             4 - 3 MHz
                                             5 - 1.5 MHz
                                             6 - 750 kHz
                                             7 - 375 kHz

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_SetClockSettings

```
int M_Mcp2221_SetClockSettings(IntPtr handle, unsigned char whichToSet, unsigned char dutyCycle,
                               unsigned char clockDivider)
================================================================================================
Description: Sets the duty cycle and clock divider values for the clock out pin (if configured for
               this operation).

Parameters:
 Inputs:
  (IntPtr)        handle     - the handle for the device
  (unsigned char) whichToSet - 0 to write Flash settings, >0 to write SRAM (runtime) settings
  (unsigned char) dutyCycle  - value of the duty cycle of the waveform on the clock pin
                                          0 - 0 %
                                          1 - 25 %
                                          2 - 50 %
                                          3 - 75 %
  (unsigned char) clockDivider - value of the clock divider. The value provided is a power of 2.
                                 The 48Mhz internal clock is divided by 2^value to obtain the
                                 output waveform frequency. The correspondence between the divider
                                 values and output frequencies are as follows:
                                             1 - 24 MHz
                                             2 - 12 MHz
                                             3 - 6 MHz
                                             4 - 3 MHz
                                             5 - 1.5 MHz
                                             6 - 750 kHz
                                             7 - 375 kHz

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_GetDacVref

```
int M_Mcp2221_GetDacVref(IntPtr handle, unsigned char whichToGet, unsigned char% dacVref)
================================================================================================
Description: Gets the DAC voltage reference.

Parameters:
 Inputs:
  (IntPtr)        handle     - the handle for the device
  (unsigned char) whichToGet - 0 to read Flash settings, >0 to read SRAM (runtime) settings

 Outputs:
  (unsigned char%) dacVref - The voltage reference for the DAC:
                                        0 - Vdd
                                        1 - 1.024 V
                                        2 - 2.048 V
                                        3 - 4.096 V
Returns: 0 if successful; error code otherwise
```

M_Mcp2221_SetDacVref

```
int M_Mcp2221_SetDacVref(IntPtr handle, unsigned char whichToSet, unsigned char dacVref)
================================================================================================
Description: Sets the DAC voltage reference.

Parameters:
 Inputs:
  (IntPtr)        handle     - the handle for the device
  (unsigned char) whichToSet - 0 to write Flash settings, >0 to write SRAM (runtime) settings
  (unsigned char) dacVref    - The voltage reference for the DAC:
                                        0 - Vdd
                                        1 - 1.024 V
                                        2 - 2.048 V
                                        3 - 4.096 V
Returns: 0 if successful; error code otherwise
```

M_Mcp2221_GetAdcData

```
int M_Mcp2221_GetAdcData(void* handle, unsigned int* adcDataArray)
================================================================================
Description: Reads the ADC data for all 3 analog pins.

Parameters:
 Inputs:
  (IntPtr)                  handle       - the handle for the device.
  (array<System::UInt32>^) adcDataArray - Array containing the ADC values. Entry 0 will contain
                                          the value for ADC1, entry 1 - ADC2, entry 2 - ADC3

Returns: 0 if successful; error code otherwise

NOTE: the array must have a minimum length of 3.
```

M_Mcp2221_GetAdcVref

```
int M_Mcp2221_GetAdcVref(IntPtr handle, unsigned char whichToGet, unsigned char% adcVref)
================================================================================
Description: Gets the ADC voltage reference.

Parameters:
 Inputs:
  (IntPtr)        handle     - the handle for the device
  (unsigned char) whichToGet - 0 to read Flash settings, >0 to read SRAM (runtime) settings
 Outputs:
  (unsigned char%) adcVref - The voltage reference for the DAC:
                                        0 - Vdd
                                        1 - 1.024 V
                                        2 - 2.048 V
                                        3 - 4.096 V

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_SetAdcVref

```
int M_Mcp2221_SetAdcVref(IntPtr handle, unsigned char whichToSet, unsigned char adcVref)
================================================================================================
Description: Sets the ADC voltage reference.

Parameters:
 Inputs:
  (IntPtr)        handle     - the handle for the device
  (unsigned char) whichToSet - 0 to write Flash settings, >0 to write SRAM (runtime) settings
  (unsigned char) adcVref    - The voltage reference for the ADC:
                                          0 - Vdd
                                          1 - 1.024 V
                                          2 - 2.048 V
                                          3 - 4.096 V

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_GetDacValue

```
int M_Mcp2221_GetDacValue(IntPtr handle, unsigned char whichToGet, unsigned char% dacValue)
================================================================================================
Description: Gets the DAC value.

Parameters:
 Inputs:
  (IntPtr)        handle     - the handle for the device
  (unsigned char) whichToGet - 0 to read Flash settings, >0 to read SRAM (runtime) settings

 Outputs:
  (unsigned char%) dacValue - The DAC output value. Valid range is between 0 and 31.

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_SetDacValue

```
int M_Mcp2221_SetDacValue(IntPtr handle, unsigned char whichToSet, unsigned char dacValue)
================================================================================================
Description: Sets the DAC value.

Parameters:
 Inputs:
  (IntPtr)        handle     - the handle for the device
  (unsigned char) whichToSet - 0 to write Flash settings, >0 to write SRAM (runtime) settings
  (unsigned char) dacValue   - The DAC output value. Valid range is between 0 and 31.

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_GetGpioSettings

```
int M_Mcp2221_GetGpioSettings(IntPtr handle, unsigned char whichToGet,
                              array<System::Byte>^ pinFunctions,
                              array<System::Byte>^ pinDirections,
                              array<System::Byte>^ outputValues)
================================================================================
Description: Gets the GPIO settings.

Parameters:
 Inputs:
  (IntPtr)        handle     - the handle for the device
  (unsigned char) whichToGet - 0 to read Flash settings, >0 to read SRAM (runtime) settings

 Outputs:
  (array<System::Byte>^) pinFunctions - Array containing the values for the pin functions.
                                        pinFunction[i] will contain the value for pin GP"i.
                                        Possible values: 0 to 3. 0 - GPIO, 1 - Dedicated
                                        function, 2 - alternate function 0, 3 - alternate
                                        function 1, 4 - alternate function 2.

                    GP0: 0 - GPIO       GP1: 0 - GPIO          GP2: 0 - GPIO    GP3: 0 - GPIO
                         1 - SSPND           1 - Clock Out          1 - USBCFG       1 - LED I2C
                         2 - LED UART RX     2 - ADC1               2 - ADC2         2 - ADC3
                         3 - LED UART TX     3 - DAC1               3 - DAC2
                                             4 - Interrupt detection
  (array<System::Byte>^) pinDirections - Array containing the pin direction of the IO pins.
                                          0 - output
                                          1 - input
  (array<System::Byte>^) outputValues  - Array containing the value present on the output pins.
                                         0 - logic low
                                         1 - logic high

Returns: 0 if successful; error code otherwise

NOTE: all output arrays must have a minimum length of 4.
```

M_Mcp2221_SetGpioSettings

```
int M_Mcp2221_SetGpioSettings(IntPtr handle, unsigned char whichToSet,
                              array<System::Byte>^ pinFunctions,
                              array<System::Byte>^ pinDirections,
                              array<System::Byte>^ outputValues)
================================================================================================
Description: Sets the GPIO settings.

Parameters:
 Inputs:
  (IntPtr)                handle         - the handle for the device
  (unsigned char)         whichToSet     - 0 to write Flash settings, >0 to write SRAM (runtime)
                  settings
  (array<System::Byte>^) pinFunctions   - Array containing the values for the pin functions.
                                          pinFunction[i] will contain the value for pin GP"i".
                                          Possible values: 0 to 3. 0 - GPIO, 1 - Dedicated
                                          function, 2 - alternate function 0, 3 - alternate
                                          function 1, 4 - alternate function 2.

                     GP0: 0 - GPIO       GP1: 0 - GPIO          GP2: 0 - GPIO    GP3: 0 - GPIO
                          1 - SSPND           1 - Clock Out          1 - USBCFG       1 - LED I2C
                          2 - LED UART RX     2 - ADC1               2 - ADC2         2 - ADC3
                          3 - LED UART TX     3 - DAC1               3 - DAC2
                                              4 - Interrupt detection

  (array<System::Byte>^) pinDirections - Array containing the pin direction of the IO pins.
                                          0 - output
                                          1 - input
  (array<System::Byte>^) outputValues   - Array containing the value present on the output pins.
                                          0 - logic low
                                          1 - logic high

Returns: 0 if successful; error code otherwise

NOTE: all output arrays must have a minimum length of 4.
```

M_Mcp2221_GetGpioValues

```
int M_Mcp2221_GetGpioValues(IntPtr handle, array<System::Byte>^ gpioValues)
================================================================================
Description: Gets the GPIO pin values.

Parameters:
 Inputs:
  (IntPtr) handle - the handle for the device

 Outputs:
  (array<System::Byte>^) gpioValues - Array containing the value present on the IO pins.
                                      0 - logic low
                                      1 - logic high
                                      0xEE - GPx not set for GPIO operation

Returns: 0 if successful; error code otherwise

NOTE: the output array must have a minimum length of 4.
```

M_Mcp2221_SetGpioValues

```
int M_Mcp2221_SetGpioValues(IntPtr handle, array<System::Byte>^ gpioValues)
================================================================================
Description: Sets the GPIO pin values.

Parameters:
 Inputs:
  (IntPtr)                handle     - the handle for the device
  (array<System::Byte>^) gpioValues - Array containing the value present on the IO pins.
                                      0 - logic low
                                      1 - logic high

Returns: 0 if successful; error code otherwise

NOTE: the array must have a minimum length of 4.
```

M_Mcp2221_GetGpioDirection

```
int M_Mcp2221_GetGpioDirection(IntPtr handle, array<System::Byte>^ gpioDir)
================================================================================
Description: Gets the GPIO pin direction.

Parameters:
 Inputs:
  (IntPtr) handle - the handle for the device

 Outputs:
  (array<System::Byte>^) gpioDir - Array containing the direction of the IO pin
                                   0 - output
                                   1 - input
                                   0xEF - GPx not set for GPIO operation

Returns: 0 if successful; error code otherwise

NOTE: the output array must have a minimum length of 4.
```

M_Mcp2221_SetGpioDirection

```
int M_Mcp2221_SetGpioDirection(IntPtr handle, array<System::Byte>^ gpioDir)
================================================================================
Description: Sets the GPIO pin direction.

Parameters:
 Inputs:
  (IntPtr)               handle  - the handle for the device
  (array<System::Byte>^) gpioDir - Array containing the direction of the IO pin
                                   0 - output
                                   1 - input

Returns: 0 if successful; error code otherwise

NOTE: the array must have a minimum length of 4.
```

Security

M_Mcp2221_GetSecuritySetting

```
int M_Mcp2221_GetSecuritySetting(IntPtr handle, unsigned char% securitySetting)
================================================================================
Description: Gets the state of flash protection for the device

Parameters:
 Inputs:
  (IntPtr) handle - the handle for the device

 Outputs:
  (unsigned char%) securitySetting - the value of the chip security option
                                     0 - unsecured
                                     1 - password protected
                                     2 or 3 - permanently locked
Returns: 0 if successful; error code otherwise
```

M_Mcp2221_SetSecuritySetting

```
int M_Mcp2221_SetSecuritySetting(IntPtr handle, unsigned char securitySetting,
                                 String^ currentPassword, String^ newPassword)
================================================================================
Description: Sets the state of flash protection for the device

Parameters:
 Inputs:
  (IntPtr)        handle          - the handle for the device
  (unsigned char) securitySetting - the value of the chip security option. If any other values are
                                    used, the E_ERR_INVALID_PARAMETER (-4) error is returned.
                                     0 - disable password protection
                                     1 - enable password protection
                                     0xff - change current password
  (String^)       currentPassword - the value for the currently set password. This is used for
                                    when the password "disable" or "change" operations are taking
                                    place.
  (String^)       newPassword     - the value for the new password. Must be an 8 character string.
                                    This is only for the "enable" or "change" operations.

Returns: 0 if successful; error code otherwise
```

M_Mcp2221_SetPermanentLock

```
int M_Mcp2221_SetPermanentLock(IntPtr handle)
================================================================================
Description: Permanently lock the device flash settings -- this action CAN'T be undone.

Parameters:
 Inputs:
  (IntPtr) handle - the handle for the device to be locked

Returns: 0 if successful; error code otherwise

!!! WARNING !!! -- USE THIS FUNCTION WITH GREAT CAUTION.  THE CHIP FLASH SETTINGS (boot-up
defaults) CANNOT BE CONFIGURED AFTER THIS FUNCTION HAS BEEN INVOKED!!
```

M_Mcp2221_SendPassword

```
int M_Mcp2221_SendPassword(IntPtr handle, String^ password)
================================================================================
Description: Sends the access password to the device.

Parameters:
 Inputs:
  (IntPtr)  handle   - the handle for the device
  (String^) password - the password that will be sent to the device to unlock writing to flash.
                       Must be an 8 character string.

Returns: 0 if successful; error code otherwise
```