MCP2210 DLL User Guide

Contents

Document Revision History

Version	Release Date	Description
V1.0	08Dec2015	Initial release
V1.1	07Mar2016	Added the note about the library API C/C++ calling convention, in the “API integration considerations” paragraph.

Which library type to choose?

MCP2210 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 .dll type to use, follow the guidelines below:

Scenario:	Which .dll type 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**

Mind that the package also contains the MCP2210 control API in the form of a standard C library (.lib) that can be statically linked to the user application.

Where I can find the needed info?

The table below lists the files that contain useful information about MCP2210 DLL, the API details and relevant user application integration recommendations.

File name	Description
Release Notes.txt	The document is a brief characterization of the library package: - content overview - system requirements - revision history
MCP2210 DLL User Guide.docx (this document)	The document presents the main library design and integration concepts that users must take into consideration: - API brief description - integration "how-to" - error codes and error handling - performance optimizations Additional references are included to the documents presenting the detailed API description and to example code files.
UM API Documentation.txt	The document presents the detailed API exported by the unmanaged DLL : - function definitions - parameters role and valid ranges - return values - special notes about API behavior in certain conditions, assumptions or prerequisites

M API Documentation.txt	The document presents the detailed API exported by the managed DLL : - function definitions - parameters role and valid ranges - return values - special notes about API behavior in certain conditions, assumptions or prerequisites
mcp2210_dll_um.h	This header file contains also the configuration constants definitions and the error codes definitions. Mind that “API documentation” files makes use of the constants names instead of their numerical values.
demo_open_index0.py **list_mcp2210.py** **demo_spi_xfer.py**	These are very simple “didactical” python scripts whose objective is twofold: - demonstrate the unmanaged DLL API integration into Python applications - illustrate the main DLL integration flow (which is the same regardless the DLL type and programming language)

DLL Requirements

A breakdown of the requirements is shown below:

DLL type:	**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 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. Special care must be taken with the API functions that return variable sized strings: e.g. the device "open" functions ("OpenByIndex()","OpenBySN()") which may return device path strings larger than the user provisioned buffer. In contrast, the **managed** API has the capability to export dynamically allocated or re-allocated memory and the user should not worry about memory management which is done automatically by .NET. Because of this, the managed API counterparts dropped the assumptions about some buffer parameters and their "size" parameters companions have been eliminated.

Please refer to the individual API function descriptions from "Annex 1 - MCP2210 Unmanaged DLL API" and "Annex 2 - MCP2210 Managed DLL API" chapters, in order to ensure the appropriate API utilization for the chosen DLL type.

DLL Structure

The **managed** DLL implements a public class (named "MCP2210") 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 MCP2210 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 considerations

The **device handles**:

- The DLL makes no assumption about how many MCP2210 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 MCP2210 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 MCP2210 device are serialized and consistent with the desired functionality. (For instance, the appropriate SPI configuration API call is issued to prepare the SPI data transfer that follows and no intermediate API call that can alter the SPI configuration may occur in between).

Device **index**, **serial** number and **path**:

- The device **index** may change dynamically, if new MCP2210 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 “open” API can optionally return the device **path**, which doesn’t change as long as the device stays connected.

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. mcp2210_dll_um.h) can be adapted to indicate for each API the calling convention.
  For example, change:
  “ *MCP2210_DLL_UM_API int Mcp2210_GetLibraryVersion(wchar_t *version);*” to:
  “*MCP2210_DLL_UM_API int* ***__stdcall*** *Mcp2210_GetLibraryVersion(wchar_t *version);*”

Performance optimization

Frequent DLL API calls may consume significant amount of CPU cycles on the HOST PC and especially on the target MCP2210 device, which has a much lower CPU power budget of course. For instance, this may become a limiting factor for the SPI data transfer bandwidth, especially if each data transfer command is preceded by GPIO and SPI reconfiguration commands.

Still, there are several features provided by SPI "**xferSpiData**" API we can use in order to reduce the total number of MCP210 device control transactions:

1. Use "**xferSpiData()**" instead of "**xferSpiDataEx**" if there is no need to change the SPI configuration parameters:
    a) Use the **Mcp2210_xferSpiDataEx()** API call for the first data packet transfer. This API configures all the SPI related parameters, issuing an additional device configuration transaction before the actual SPI data transfer transaction.
    b) Use the **Mcp2210_xferSpiData()** API call for the next SPI data transfers with the same SPI device. This API still allows the user to change the baud rate, the transfer size and the chip select mask parameters, but, if the changes are not needed, the DLL skips the unnecessary device control transactions (see "2" and "3" below).
2. Use "**csmask**" parameter capability to indicate "no CS change" and/or "ignore GP8 errata":
    a) If **csmask[8..0] = 0**, current "chip select" configuration is preserved. The DLL will skip one GPIO configuration command before the SPI data transfer.
    b) If **csmask[31] = 1**, the GP8 errata is ignored, the DLL will skip one GPIO configuration command, which otherwise is issued after the SPI data transfer.
    c) If both indications above can be used – e.g. **csmask = 0x10000000** – the DLL saves three GPIO configuration commands (two before and one after the SPI data transfer command)
3. If both "**pbaudRate**" and "**ptxferSize**" parameters indicate 'no change', the DLL skips one SPI configuration command before the SPI data transfer command:
    a) **pbaudRate = 0** or **pbaudRate =** ***previous value***
    b) **ptxferSize =** ***previous value*** or **ptxferSize >= 0x10000**
4. If possible, reduce the number of "**xferSpiData()**" and "**xferSpiDataEx()**" function calls per SPI data buffer transfer session, by setting the transfer size (**ptxferSize** parameter) to the largest value supported by the SPI device. As an example, for reading/writing large amounts of data from/to SPI connected EEPROM, set the SPI transfer size to the data buffer size instead of doing repeated single byte transfers.

The python demo application ***demo_spi_xfer.py*** illustrates the efficient use of the SPI data transfer API.

DLL Function List

Unmanaged DLL	Managed DLL
Mcp2210_GetLibraryVersion()	M_Mcp2210_GetLibraryVersion()
Mcp2210_GetConnectedDevCount()	M_Mcp2210_GetConnectedDevCount()
API for USB device open	
Mcp2210_OpenByIndex()	M_Mcp2210_OpenByIndex()
Mcp2210_OpenBySN()	M_Mcp2210_OpenBySN()
Mcp2210_GetLastError()	M_Mcp2210_GetLastError()
Mcp2210_Close()	M_Mcp2210_Close()
Mcp2210_Reset()	M_Mcp2210_Reset()
API for USB settings	
Mcp2210_GetUsbKeyParams()	M_Mcp2210_GetUsbKeyParams()
Mcp2210_SetUsbKeyParams()	M_Mcp2210_SetUsbKeyParams()
Mcp2210_GetManufacturerString()	M_Mcp2210_GetManufacturerString()
Mcp2210_SetManufacturerString()	M_Mcp2210_SetManufacturerString()
Mcp2210_GetProductString()	M_Mcp2210_GetProductString()
Mcp2210_SetProductString()	M_Mcp2210_SetProductString()
Mcp2210_GetSerialNumber()	M_Mcp2210_GetSerialNumber()
API for GPIO settings	
Mcp2210_GetGpioPinDir()	M_Mcp2210_GetGpioPinDir()
Mcp2210_SetGpioPinDir()	M_Mcp2210_SetGpioPinDir()
Mcp2210_GetGpioPinVal()	M_Mcp2210_GetGpioPinVal()
Mcp2210_SetGpioPinVal()	M_Mcp2210_SetGpioPinVal()
Mcp2210_GetGpioConfig()	M_Mcp2210_GetGpioConfig()
Mcp2210_SetGpioConfig()	M_Mcp2210_SetGpioConfig()
Mcp2210_GetInterruptCount()	M_Mcp2210_GetInterruptCount()
API for SPI transfer	
Mcp2210_GetSpiConfig()	M_Mcp2210_GetSpiConfig()
Mcp2210_SetSpiConfig()	M_Mcp2210_SetSpiConfig()
Mcp2210_xferSpiData()	M_Mcp2210_xferSpiData()
Mcp2210_xferSpiDataEx()	M_Mcp2210_xferSpiDataEx()
Mcp2210_CancelSpiTxfer()	M_Mcp2210_CancelSpiTxfer()
Mcp2210_RequestSpiBusRel()	M_Mcp2210_RequestSpiBusRel()
Mcp2210_GetSpiStatus()	M_Mcp2210_GetSpiStatus()
API for EEPROM read/write	
Mcp2210_ReadEEProm()	M_Mcp2210_ReadEEProm()
Mcp2210_WriteEEProm()	M_Mcp2210_WriteEEProm()
API for device access control	
Mcp2210_GetAccessCtrlStatus()	M_Mcp2210_GetAccessCtrlStatus()
Mcp2210_EnterPassword()	M_Mcp2210_EnterPassword()
Mcp2210_SetAccessControl()	M_Mcp2210_SetAccessControl()
Mcp2210_SetPermanentLock()	M_Mcp2210_SetPermanentLock()

DLL Error Codes

Except for the “device open” functions, all the API calls return an integer value. If this is a negative number, then it must be treated as an error code and addressed accordingly. Please mind that the error code for the “device open” functions is returned by the subsequent “GetLastError “API calls. The table below lists the error codes, their description and some recommended actions.

Error code	Unmanaged DLL error name (#define) Managed DLL error name (static const int)	Description	Recommended action
0	E_SUCCESS M_E_SUCCESS	Successful API call	
-1	E_ERR_UNKOWN_ERROR M_E_ERR_UNKOWN_ERROR	Unexpected error, most likely caused by communication issues.	1.Try again 2.Reset the device, re-open it and try again 3.Exit the application
-2	E_ERR_INVALID_PARAMETER M_E_ERR_INVALID_PARAMETER	At least one API parameter is not valid.	Check the parameter validity and try again.
-3	E_ERR_BUFFER_TOO_SMALL M_E_ERR_BUFFER_TOO_SMALL	Too small buffer parameter.	Allocate the recommended buffer size and try again.
Memory access errors			
-10	E_ERR_NULL M_E_ERR_NULL	NULL pointer parameter.	Application memory management issue. Exit the application.
-20	E_ERR_MALLOC M_E_ERR_MALLOC	Memory allocation error	The system resources are very low. Exit the application.
-30	E_ERR_INVALID_HANDLE_VALUE M_E_ERR_INVALID_HANDLE_VALUE	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.
Device connection errors			
-100	E_ERR_FIND_DEV M_E_ERR_FIND_DEV	Reserved error code, not in use in this release.	No action required.
-101	E_ERR_NO_SUCH_INDEX M_E_ERR_NO_SUCH_INDEX	Attempt to open invalid device index	Use *GetConnectedDevCount()* API to check the number of connected devices. The index must be smaller.
-103	E_ERR_DEVICE_NOT_FOUND M_E_ERR_DEVICE_NOT_FOUND	There is no device connected having the given VID:PID numbers	Check the VID and PID parameters given to the “open” functions

Error code	Unmanaged DLL error name (#define) Managed DLL error name (static const int)	Description	Recommended action
-104	E_ERR_INTERNAL_BUFFER_TOO_SMALL M_E_ERR_INTERNAL_BUFFER_TOO_SMALL	Reserved error code, not in use in this release.	No action required.
-105	E_ERR_OPEN_DEVICE_ERROR M_E_ERR_OPEN_DEVICE_ERROR	Device open attempt failure.	1.Try again 2.Reset the device, and try again 3.Exit the application
-106	E_ERR_CONNECTION_ALREADY_OPENED M_E_ERR_CONNECTION_ALREADY_OPENED	Device is already open.	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.	1.Try again. 2.Exit the application.
-108	E_ERR_NO_SUCH_SERIALNR M_E_ERR_NO_SUCH_SERIALNR	No device found with the given serial number.	Check the serial number.
-110	E_ERR_HID_RW_TIMEOUT M_E_ERR_HID_RW_TIMEOUT	HID file operation timeout. Device may be disconnected.	1.Try again 2. Close the handle, the device is no longer accessible.
-111	E_ERR_HID_RW_FILEIO M_E_ERR_HID_RW_FILEIO	HID file operation unknown error. Device may be disconnected	1.Try again 2. Close the handle, the device is no longer accessible.
Device command reply errors			
-200	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.
-201	E_ERR_CMD_ECHO M_E_ERR_CMD_ECHO	The command code echoed by device doesn't match the one sent by the DLL.	This is considered a command failure indication. But, if the application is multithreaded, the issue may be also caused by concurrent device access from separate threads.
-202	E_ERR_SUBCMD_ECHO M_E_ERR_SUBCMD_ECHO	Reserved error code, not in use in this release.	No action required.
-203	E_ERR_SPI_CFG_ABORT M_E_ERR_SPI_CFG_ABORT	SPI configuration change refused because transfer is in progress	Check the device SPI status (using GetSpiStatus() API) and try again.

Error code	Unmanaged DLL error name (#define) Managed DLL error name (static const int)	Description	Recommended action
-204	E_ERR_SPI_EXTERN_MASTER M_E_ERR_SPI_EXTERN_MASTER	The SPI bus is owned by external master, data transfer not possible	Check the device SPI status (using GetSpiStatus() API) and try again.
-205	E_ERR_SPI_TIMEOUT M_E_ERR_SPI_TIMEOUT	SPI transfer attempts exceeded the MCP2210_XFER_RETRIES threshold	SPI data transfer is not working properly. Check the SPI data transfer settings and try again.
-206	E_ERR_SPI_RX_INCOMPLETE M_E_ERR_SPI_RX_INCOMPLETE	The number of bytes received after the SPI transfer is less than configured transfer size	SPI data transfer is not working properly. Check the SPI settings and try agin.
-207	E_ERR_SPI_XFER_ONGOING M_E_ERR_SPI_XFER_ONGOING	An SPI data transfer is in progress.	Check the device SPI status (using GetSpiStatus() API) and try again.
Device password protection errors			
-300	E_ERR_BLOCKED_ACCESS M_E_ERR_BLOCKED_ACCESS	The command cannot be executed because the device settings are either password protected or permanently locked.	Check the device protection status (use GetAccessStatus() API). If the device is not permanently locked, the password protection can be unlocked or disabled by using the access control API (SetAccessControl()).
-301	E_ERR_EEPROM_WRITE_FAIL M_E_ERR_EEPROM_WRITE_FAIL	EEPROM write failure due to FLASH memory failure	This indicates a memory hardware failure. It cannot be addressed by software.
-350	E_ERR_NVRAM_LOCKED M_E_ERR_NVRAM_LOCKED	NVRAM is permanently locked, no password is accepted	NVRAM cannot be unlocked by software.
-351	E_ERR_WRONG_PASSWD M_E_ERR_WRONG_PASSWD	Password mismatch, but number of attempts is less than 5	Check the password string and try again.
-352	E_ERR_ACCESS_DENIED M_E_ERR_ACCESS_DENIED	Password mismatch, but the number of attempts exceeded 5, so the NVRAM access is denied until the next device reset	Check the password, reset the device and try again. (Reset() API can be used to reset the device)
-353	E_ERR_NVRAM_PROTECTED M_E_ERR_NVRAM_PROTECTED	NVRAM access control protection is already enabled, so the attempt to enable it twice is rejected	Skip to the next command.

Error code	Unmanaged DLL error name (#define) Managed DLL error name (static const int)	Description	Recommended action
-354	E_ERR_PASSWD_CHANGE M_E_ERR_PASSWD_CHANGE	NVRAM access control is not enabled, so password change is not allowed	If password protection is required, use the SetAccessControl() API to enable it using the new password.
Device USB descriptors errors			
-400	E_ERR_STRING_DESCRIPTOR M_E_ERR_STRING_DESCRIPTOR	The NVRAM string descriptor is invalid	NVRAM content seems invalid ID. At least one of the two conditions is violated: - USB string descriptor ID must be 0x03 - descriptor size is not between 2 and 60 bytes The user application cannot retrieve the data.
-401	E_ERR_STRING_TOO_LARGE M_E_ERR_STRING_TOO_LARGE	The size of the input string exceeds the limit	The string length is larger than 29 UNICODE chars without counting the NULL terminator.

DLL constants

Unmanaged DLL constant name (#define) Managed DLL constant name (static const unsigned int)	Value	Description
MPC2210_LIBRARY_VERSION_SIZE **M_MPC2210_LIBRARY_VERSION_SIZE**	64	Version string maximum byte size including null character. "***version***" parameter of the "***Mcp2210_GetLibraryVersion***" API should point to a buffer of this size.
MPC2210_SERIAL_NUMBER_LENGTH **M_MPC2210_SERIAL_NUMBER_LENGTH**	10	MPC2210 HID serial number length – the number of wide characters "***serialNo***" parameter of the "***OpenBySN***" API must have this length.
MCP2210_GPIO_NR **M_MCP2210_GPIO_NR**	9	There are 9 GPIO pins "***pGpioPinDes***" buffer parameter of the "***Set/GpioConfig***" API must have this size
GPIO configuration values contained in "***pGpioPinDes***" buffer, a parameter of the "***Set/GpioConfig***" API		
MCP2210_PIN_DES_GPIO **M_MCP2210_PIN_DES_GPIO**	0x00	GPIO Pin configured as GPIO
MCP2210_PIN_DES_CS **M_MCP2210_PIN_DES_CS**	0x01	GPIO pin configured as chip select – CS
MCP2210_PIN_DES_FN **M_MCP2210_PIN_DES_FN**	0x02	GPIO pin configured as dedicated function pin
"***cfgSelector***" parameter values for "***Get/SetGpioConfig***" and "***Get/SetSpiConfig***" API		
MCP2210_VM_CONFIG **M_MCP2210_VM_CONFIG**	0	VM/NVRAM selection: Volatile Memory
MCP2210_NVRAM_CONFIG **M_MCP2210_NVRAM_CONFIG**	1	VM/NVRAM selection: NVRAM
"***rmtWkup***" parameter values for the "***GetSetUsbKeyParams***" and "***Get/SetGpioConfig***" API		
MCP2210_REMOTE_WAKEUP_ENABLED **M_MCP2210_REMOTE_WAKEUP_ENABLED**	1	Remote wake-up enable
MCP2210_REMOTE_WAKEUP_DISABLED **M_MCP2210_REMOTE_WAKEUP_DISABLED**	0	remote wake-up disable
"***intPinMd***" parameter values for the "***Get/SetGpioConfig***" API		
MCP2210_INT_MD_CNT_HIGH_PULSES **M_MCP2210_INT_MD_CNT_HIGH_PULSES**	0x4	Interrupt counting mode – count high pulses
MCP2210_INT_MD_CNT_LOW_PULSES **M_MCP2210_INT_MD_CNT_LOW_PULSES**	0x3	Interrupt counting mode – count low pulses
MCP2210_INT_MD_CNT_RISING_EDGES **M_MCP2210_INT_MD_CNT_RISING_EDGES**	0x2	Interrupt counting mode – count rising edges

Unmanaged DLL constant name (#define) Managed DLL constant name (static const unsigned int)	Value	Description
MCP2210_INT_MD_CNT_FALLING_EDGES **M_MCP2210_INT_MD_CNT_FALLING_EDGES**	0x1	Interrupt counting mode – count falling edges
MCP2210_INT_MD_CNT_NONE **M_MCP2210_INT_MD_CNT_NONE**	0x0	Interrupt counting mode – no counting
”***spiBusRelEn***” parameter values for the “***Get/SetGpioConfig***” API		
MCP2210_SPI_BUS_RELEASE_ENABLED **M_MCP2210_SPI_BUS_RELEASE_ENABLED**	0	SPI bus release enable/disable
MCP2210_SPI_BUS_RELEASE_DISABLED **M_MCP2210_SPI_BUS_RELEASE_DISABLED**	1	SPI bus release enable/disable
“***ackPinVal***” parameter value for the “***RequestSpiBusRel***” API		
MCP2210_SPI_BUS_RELEASE_ACK_LOW **M_MCP2210_SPI_BUS_RELEASE_ACK_LOW**	0	SPI bus release ACK pin value
MCP2210_SPI_BUS_RELEASE_ACK_HIGH **M_MCP2210_SPI_BUS_RELEASE_ACK_HIGH**	1	SPI bus release ACK pin value
MCP2210_XFER_RETRIES **M_MCP2210_XFER_RETRIES**	200	SPI maximum transfer attempts threshold for each SPI data transfer chunk. The maximum chunk size is the HID payload size – 64bytes.
“***currentLd***” parameter value for the “***Set/GetUsbKeyParams***” API		
MCP2210_MIN_USB_AMPERAGE **M_MCP2210_MIN_USB_AMPERAGE**	2	Min current amount required from USB host
MCP2210_MAX_USB_AMPERAGE **M_MCP2210_MAX_USB_AMPERAGE**	510	Max current amount required from USB host
MCP2210_DESCRIPTOR_STR_MAX_LEN **M_MCP2210_DESCRIPTOR_STR_MAX_LEN**	29	maximum UNICODE string length of the USB string descriptors, without NULL terminator (manufacturer and product strings)
“***pspiMd***” parameter value for the “***Get/SetSpiConfig***” and “***xferSpiData/ xferSpiDataEx***” API		
MCP2210_SPI_MODE0 **M_MCP2210_SPI_MODE0**	0x00	SPI Mode selection – Mode 0
MCP2210_SPI_MODE1 **M_MCP2210_SPI_MODE1**	0x01	SPI Mode selection – Mode 1
MCP2210_SPI_MODE2 **M_MCP2210_SPI_MODE2**	0x02	SPI Mode selection – Mode 2
MCP2210_SPI_MODE3 **M_MCP2210_SPI_MODE3**	0x03	SPI Mode selection – Mode 3

Unmanaged DLL constant name (#define) Managed DLL constant name (static const unsigned int)	Value	Description
MCP2210_GP8CE_MASK **M_MCP2210_GP8CE_MASK**	0x80000000	GP8 firmware error workaround bit Valid value for "***csmask***" parameter of the "***xferSpiData/ xferSpiDataEx***" API
MCP2210_NVRAM_PASSWD_LEN **M_MCP2210_NVRAM_PASSWD_LEN**	8	The password must be a NULL terminated string of 8 characters (bytes) Password strings are parameters for "***EnterPassword***" and "***SetAccessControl***" API
"***accessConfig***" parameter value for the "***SetAccessControl***" API		
MCP2210_NVRAM_NO_PROTECTION **M_MCP2210_NVRAM_NO_PROTECTION**	0x00	NVRAM chip settings protection – not enabled
MCP2210_NVRAM_PROTECTED **M_MCP2210_NVRAM_PROTECTED**	0x40	NVRAM chip settings protection – enabled
MCP2210_NVRAM_LOCKED **M_MCP2210_NVRAM_LOCKED**	0x80	NVRAM chip settings protection – locked
MCP2210_NVRAM_PASSWD_CHANGE **M_MCP2210_NVRAM_PASSWD_CHANGE**	0xA5	NVRAM chip settings protection – password change

Annex 1 – MCP2210 Unmanaged DLL API

Mcp2210_GetLibraryVersion

```
int Mcp2210_GetLibraryVersion(wchar_t *version)
==========================================================================================
Description: Provides the library version as NULL terminated UNICODE string.
Parameters:
Inputs:
    - none
Outputs:
    - wchar_t *version - Pointer to buffer string of 64 bytes (MPC2210_LIBRARY_VERSION_SIZE) in size
                         (So it can accommodate max 30 wide characters.)
                         The version string, including the NULL terminator is copied in this buffer.
                         Cannot be NULL.
Returns:
        - positive value: version string size (bytes) including the null character
        - negative value: E_ERR_NULL (-10) if *version is NULL
```

Mcp2210_GetLastError

```
int Mcp2210_GetLastError();
===================================================================================================
Description: Returns the detailed error code of the last library function call whose prototype is
not making this possible. The library functions that need this support are:
- Mcp2210_OpenByIndex()
- Mcp2210_OpenBySN()
!!! Important note !!!
In order to avoid the last error code being altered by other API calls, please make sure that
Mcp2210_GetLastError() is called immediately after the API listed above and save the returned value
accordingly.

Parameters:
Inputs:
    - none
Outputs:
    - none
Returns:
    - E_SUCCESS or negative error code: please see the description of the above listed API calls
```

Mcp2210_GetConnectedDevCount

```
int Mcp2210_GetConnectedDevCount(unsigned short vid, unsigned short pid)
====================================================================================================
Description: provides the number of connected devices with the given VID:PID.
Returns negative number in case of error.
Parameters:
Inputs:
    - unsigned short vid - MCP2210 device VendorID (Microchip default = 0x4D8)
    - unsigned short pid - MCP2210 device ProductID (Microchip default = 0xDE)
Outputs:
    - none
Returns:
    - the number of connected devices with the given VID:PID or
    - negative error code: E_ERR_UNKOWN_ERROR, E_ERR_MALLOC
```

Mcp2210_OpenByIndex

```
void* Mcp2210_OpenByIndex(unsigned short vid, unsigned short pid, unsigned int index,
                          wchar_t *devPath, unsigned long *devPathsize)
====================================================================================================
Description:  Opens the connection with the MCP2210 device identified by the given VID:PID
and the index number and returns the file handle.
Also, the function can provide the device path as NULL terminated UNICODE string, if a valid pointer
to a buffer of sufficient size is provisioned. If the buffer size is smaller than required,
the function call fails, the device is not opened, but the devPathsize parameter contains the value
required for the path string size.
The function provides also the option to skip the path string if this is not needed.

!!! Important note !!!
In case of failure, the function returns an invalid handle value (-1) and updates an internal variable
with the specific error code (negative values). This can be retrieved with Mcp2210_GetLastError() call.
The error codes that Mcp2210_GetLastError() might return after Mcp2210_OpenByIndex() call are:
    - E_SUCCESS (0), if no error occurred
    - E_ERR_BUFFER_TOO_SMALL
    - E_ERR_UNKOWN_ERROR, E_ERR_MALLOC, E_ERR_NULL,
      E_ERR_DEVICE_NOT_FOUND, E_ERR_NO_SUCH_INDEX,
      E_ERR_CONNECTION_ALREADY_OPENED, E_ERR_OPEN_DEVICE_ERROR

Parameters:
Inputs:
    - unsigned short vid         - MCP2210 device VendorID (Microchip default = 0x4D8)
    - unsigned short pid         - MCP2210 device ProductID (Microchip default = 0xDE)
    - unsigned int   index       - the index of the MCP2210 device to connect to. This value ranges
                                   from 0 to n-1, where n is the number of connected devices.
                                   This value can be obtained with "Mcp2210_GetConnectedDevCount"
    - unsigned long *devPathsize - pointer to the variable that contains the size of the buffer
                                   in which the device path must be copied. The size is expressed
                                   in bytes and must accommodate also the string NULL terminator.
                                   If the pointer is NULL, or if the *devPathsize is 0, the path
                                   is not returned and the *devPath parameter is ignored.

Outputs:
    - wchar_t *devPath           - path string buffer pointer.
                                   If *devPath pointer is NULL, the path is not reported,
                                   *devPathsize parameter is ignored, but the device is opened
                                   and the function returns the handle.
    - unsigned long *devPathsize - if the function call fails because of insufficient buffer size
                                   (Mcp2210_GetLastError() returns E_ERR_BUFFER_TOO_SMALL), this
                                   parameter indicates the necessary buffer size to accommodate the
                                   path string size (in bytes).

Returns:
    - valid handle value in case of success
    - INVALID_HANDLE_VALUE (-1) in case of error.
```

Mcp2210_OpenBySN

```
void* Mcp2210_OpenBySN(unsigned short vid, unsigned short pid, wchar_t *serialNo,
                       wchar_t *devPath, unsigned long *devPathsize)
====================================================================================================
Description:  Opens the connection with the MCP2210 device identified by the given VID:PID
and the serial number string and returns the file handle.
Also, the function can provide the device path as NULL terminated UNICODE string, if a valid pointer
to a buffer of sufficient size is provisioned. If the buffer size is smaller than required,
the function call fails, the device is not opened, but the devPathsize parameter contains the value
required for the path string size.
The function provides also the option to skip the path string if this is not needed.

!!! Important note !!!
In case of failure, the function returns an invalid handle value (-1) and updates an internal variable
with the specific error code (negative values). This can be retrieved with Mcp2210_GetLastError() call.
The error codes that Mcp2210_GetLastError() might return after Mcp2210_OpenBySN() call are:
    - E_SUCCESS (0), if no error occurred
    - E_ERR_BUFFER_TOO_SMALL
    - E_ERR_UNKOWN_ERROR, E_ERR_MALLOC, E_ERR_NULL,
      E_ERR_DEVICE_NOT_FOUND, E_ERR_NO_SUCH_SERIALNR,
      E_ERR_CONNECTION_ALREADY_OPENED, E_ERR_OPEN_DEVICE_ERROR

Parameters:
Inputs:
    - unsigned short vid         - MCP2210 device VendorID (Microchip default = 0x4D8)
    - unsigned short pid         - MCP2210 device ProductID (Microchip default = 0xDE)
    - wchar_t        *serialNo   - The serial number of the MCP2210 we want to connect to:
                                   NULL terminated string of max MPC2210_SERIAL_NUMBER_LENGTH wide
characters
                                   Cannot be NULL.
    - unsigned long *devPathsize - pointer to the variable that contains the size of the buffer
                                   in which the device path must be copied. The size is expressed
                                   in bytes and must accommodate also the string NULL terminator.
                                   If the pointer is NULL, or if the *devPathsize is 0, the path
                                   is not returned and the *devPath parameter is ignored.
Outputs:
    - wchar_t *devPath           - path string buffer pointer.
                                   If *devPath pointer is NULL, the path is not reported,
                                   *devPathsize parameter is ignored, but the device is opened
                                   and the function returns the handle.
    - unsigned long *devPathsize - if the function call fails because of insufficient buffer size
                                   (Mcp2210_GetLastError() returns E_ERR_BUFFER_TOO_SMALL), this
                                   parameter indicates the necessary buffer size to accommodate the
                                   path string size (in bytes).

Returns:
    - valid handle value in case of success
    - INVALID_HANDLE_VALUE (-1) in case of error.
```

Mcp2210_Close

```
int Mcp2210_Close(void *handle)
====================================================================================================
Description: Attempt to close a connection to a MCP2210.
Parameters:
Inputs:
    - void *handle - The pointer to the device handle we'll close the connection to.
                     Cannot be NULL nor contain invalid handle value.
Outputs:
    - none
Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_CLOSE_FAILED
```

Mcp2210_Reset

```
int Mcp2210_Reset(void *handle)
====================================================================================================
Description: Reset the USB device.
In case of successful execution, the function closes the device handle and returns 0 (E_SUCCESS).
In case of failure, negative error code is returned.
!!! Important note !!!
The user application must take into account the time needed for the device reset and for
the USB enumeration process. Trying to open the device too soon after reset may fail.
Please mind also that the time needed for USB enumeration may be different on different systems, and
may vary also on the same system, depending on the configuration, CPU load a.s.o.

Parameters:
Inputs:
    - void *handle          - The pointer to the USB device handle. Cannot be NULL.

Outputs:
    - none

Returns:
    - 0 for success:        E_SUCCESS
    - negative error code:  E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                            E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                            E_ERR_CLOSE_FAILED
```

Mcp2210_GetUsbKeyParams

```
int Mcp2210_GetUsbKeyParams(void *handle, unsigned short *pvid, unsigned short *ppid,
                            unsigned char *ppwrSrc, unsigned char *prmtWkup, unsigned short *pcurrentLd)
====================================================================================================
Description: Provides the USB key configuration attributes: VID, PID and power settings.
In case of successful execution, the function returns 0 (E_SUCCESS) and updates the pointer
parameters with the settings values. In case of failure, negative error code is returned.

Parameters:
Inputs:
    - void *handle                - The pointer to the USB device handle. Cannot be NULL.

Outputs:
    - unsigned short *pvid        - device VID
                                    Parameter cannot be NULL.

    - unsigned short *ppid        - device PID
                                    Parameter cannot be NULL.

    - unsigned char *ppwrSrc      - device power source, according to USB 2.0 specs, chapter 9:
                                        - bit 7 = Host-Powered: 1=yes.
                                          This bit is reserved and must be set to 1 for historical
                                          reasons.
                                        - bit 6 = Self-Powered: 1=yes; 0=no.
                                        - bits 5..0 = 0.
                                    The parameter cannot be NULL.

    - unsigned char *prmtWkup      - device is remote wake-up capable: 1=yes; 0=no.
                                   Parameter cannot be NULL.

    - unsigned short *pcurrentLd   - Maximum current consumption from the USB Host, expressed in mA.
                                     The value is a multiple of 2, ranging from 2mA to 510mA.
                                     Parameter cannot be NULL.

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_SetUsbKeyParams

```
int Mcp2210_SetUsbKeyParams(void *handle, unsigned short vid, unsigned short pid,
                            unsigned char pwrSrc, unsigned char rmtWkup, unsigned short currentLd)
====================================================================================================
Description: Configures the USB key attributes: VID, PID and power settings.
In case of successful execution, the function returns 0 (E_SUCCESS).
In case of failure, negative error code is returned.
If the error code is "E_ERR_BLOCKED_ACCESS" the write failure is due to the device being either
password protected or permanently locked. Mcp2210_GetAccessStatus() API provides the details about
device protection status. If the device is not permanently locked, the password protection can be
unlocked or disabled by using the access control API.

Parameters:
Inputs:
    - void *handle              - The pointer to the USB device handle. Cannot be NULL.

    - unsigned short vid        - device VID

    - unsigned short  pid       - device PID

    - unsigned char pwrSrc      - device power source, according to USB 2.0 specs, chapter 9:
                                      - bit 7 = Host-Powered: 1=yes.
                                        This bit is reserved and must be set to 1 for historical
                                        reasons.
                                      - bit 6 = Self-Powered: 1=yes; 0=no.
                                      - bits 5..0 must be set to 0 (as an extra safety measure)
                                  The parameter cannot be NULL.

    - unsigned char rmtWkup     - device is remote wake-up capable: 1=yes; 0=no.

    - unsigned short currentLd    - Maximum current consumption from the USB Host, expressed in mA.
                                  The value must be a multiple of 2 value, ranging from 2mA to 510mA.
                                  If the currentLd value is not multiple by 2, the API will
                                  round it up to the next multiple by 2 number and store it in NVRAM.
Outputs:
    - none

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE, E_ERR_INVALID_PARAMETER
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
                           E_ERR_BLOCKED_ACCESS
```

Mcp2210_GetManufacturerString

```
int Mcp2210_GetManufacturerString(void *handle, wchar_t *manufacturerStr)
====================================================================================================
Description: Read from device the USB Manufacturer string.
In case of successful execution, the function returns 0 (E_SUCCESS) and updates the manufacturerStr
buffer.
In case of failure, negative error code is returned.
If the NVARM string descriptor is invalid, the function returns E_ERR_DESCRIPTOR (-400) error code.
The string descriptor is invalid if the String Descriptor ID is not 0x03 or the descriptor size
is less than 2 or larger than 60 bytes.

Parameters:
Inputs:
    - void *handle              - The pointer to the USB device handle. Cannot be NULL.

Outputs:
    - wchar_t *manufacturerStr  - pointer to UNICODE string buffer. Cannot be NULL.
                                  !! Important details !!
                                  - the string is NULL (\0) terminated
                                  - maximum string length is 29 UNICODE characters =>
                                    buffer size must be 60 bytes.

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
                           E_ERR_STRING_DESCRIPTOR
```

Mcp2210_SetManufacturerString

```
int Mcp2210_SetManufacturerString(void *handle, wchar_t *manufacturerStr)
====================================================================================================
Description: Write the USB Manufacturer string into device NVRAM.
In case of successful execution, the function returns 0 (E_SUCCESS).
In case of failure, negative error code is returned.
If the error code is "E_ERR_BLOCKED_ACCESS" the write failure is due to the device being either
password protected or permanently locked. Mcp2210_GetAccessStatus() API provides the details about
device protection status. If the device is not permanently locked, the password protection can be
unlocked or disabled by using the access control API.

Parameters:
Inputs:
    - void *handle              - The pointer to the USB device handle. Cannot be NULL.

    - wchar_t *manufacturerStr  - pointer to UNICODE string buffer. Cannot be NULL.
                                  !! Important details !!
                                  - the string must be NULL (\0) terminated
                                  - maximum string length must be 29 UNICODE characters without
                                    counting the NULL termination (MCP2210_DESCRIPTOR_STR_MAX_LEN).
                                    If the input string is larger, negative error code is returned.
Outputs:
    - none

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_STRING_TOO_LARGE
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
                           E_ERR_BLOCKED_ACCESS
```

Mcp2210_GetProductString

```
int Mcp2210_GetProductString(void *handle, wchar_t *productStr)
=============================================================================================
Description: Read from device the USB Product string.
In case of successful execution, the function returns 0 (E_SUCCESS) and updates the productStr
buffer.
In case of failure, negative error code is returned.
If the NVARM string descriptor is invalid, the function returns E_ERR_DESCRIPTOR (-400) error code.
The string descriptor is invalid if the String Descriptor ID is not 0x03 or the descriptor size is
less than 2 or larger than 60 bytes.

Parameters:
Inputs:
- void *handle              - The pointer to the USB device handle. Cannot be NULL.

Outputs:
- wchar_t *productStr       - pointer to UNICODE string buffer. Cannot be NULL.
                            !! Important details !!
                            - the string is NULL (\0) terminated
                            - maximum string length is 29 UNICODE characters =>
                              buffer size must be 60 bytes.

Returns:
    - 0 for success:        E_SUCCESS
    - negative error code:  E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                            E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                            E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
                            E_ERR_STRING_DESCRIPTOR
```

Mcp2210_SetProductString

```
int Mcp2210_SetProductString(void *handle, wchar_t *productStr)
=====================================================================================================
Description: Write the USB Product string into device NVRAM.
In case of successful execution, the function returns 0 (E_SUCCESS).
In case of failure, negative error code is returned.
If the error code is "E_ERR_BLOCKED_ACCESS" the write failure is due to the device being either
password protected or permanently locked. Mcp2210_GetAccessStatus() API provides the details about
device protection status. If the device is not permanently locked, the password protection can be
unlocked or disabled by using the access control API.

Parameters:
Inputs:
    - void *handle              - The pointer to the USB device handle. Cannot be NULL.

    - wchar_t *productStr       - pointer to UNICODE string buffer. Cannot be NULL.
                                  !! Important details !!
                                  - the string must be NULL (\0) terminated
                                  - maximum string length must be 29 UNICODE characters without
                                    counting the NULL termination (MCP2210_DESCRIPTOR_STR_MAX_LEN).
                                    If the input string is larger, negative error code is returned.
Outputs:
    - none

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_STRING_TOO_LARGE
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
                           E_ERR_BLOCKED_ACCESS
```

Mcp2210_GetSerialNumber

```
int Mcp2210_GetSerialNumber(void *handle, wchar_t *serialStr)
====================================================================================================
Description: Read from device the USB Serial Number string.
In case of successful execution, the function returns 0 (E_SUCCESS) and updates the serialStr
buffer.
In case of failure, negative error code is returned.
If the NVARM string descriptor is invalid, the function returns E_ERR_DESCRIPTOR (-400) error code.
The string descriptor is invalid if the String Descriptor ID is not 0x03 or the descriptor size is
less than 2 or larger than 60 bytes.

Parameters:
Inputs:
    - void *handle          - The pointer to the USB device handle. Cannot be NULL.

Outputs:
    - wchar_t *serialStr    - pointer to UNICODE string buffer. Cannot be NULL.
                            !! Important details !!
                            - the string is NULL (\0) terminated
                            - maximum string length is 29 UNICODE characters =>
                              buffer size must be 60 bytes.

Returns:
    - 0 for success:        E_SUCCESS
    - negative error code:  E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                            E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                            E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
                            E_ERR_STRING_DESCRIPTOR
```

Mcp2210_GetGpioPinDir

```
int Mcp2210_GetGpioPinDir(void *handle, unsigned int *pgpioDir)
====================================================================================================
Description: Provides the current GPIO pin direction. The values are valid only for the pins
             previously configured as GPIOs.
Parameters:
Inputs:
    - void *handle              - The pointer to the device handle. Cannot be NULL.
Outputs:
    - unsigned int *pgpioDir    - GPIO pin direction, if successful call. In case of error the value
                                  is not changed.
                                  Cannot be NULL pointer.
Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_UNKOWN_ERROR, E_ERR_INVALID_PARAMETER
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_SetGpioPinDir

```
int Mcp2210_SetGpioPinDir(void *handle, unsigned int gpioSetDir)
====================================================================================================
Description: Set the GPIO direction.
             The values are valid only for the pins previously configured as GPIOs.
Parameters:
Inputs:
    - void *handle              - The pointer to the device handle. Cannot be NULL.
    - unsigned int gpioSetdir   - GPIO new pin direction.
Outputs:
    - none
Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_UNKOWN_ERROR, E_ERR_INVALID_PARAMETER
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_GetGpioPinVal

```
int Mcp2210_GetGpioPinVal(void *handle, unsigned int *pgpioPinVal)
====================================================================================================
Description: Provides the current GPIO values. The values are valid only for the pins
             previously configured as GPIOs.
Parameters:
Inputs:
    - void *handle              - The pointer to the device handle. Cannot be NULL.
Outputs:
    - unsigned int *pgpioPinVal - GPIO pin values, if successful call. In case of error the value
                                  is not changed.
                                  Cannot be NULL pointer.
Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_UNKOWN_ERROR, E_ERR_INVALID_PARAMETER
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_SetGpioPinVal

```
int Mcp2210_SetGpioPinVal(void *handle, unsigned int gpioSetVal, unsigned int *pgpioPinVal)
====================================================================================================
Description: Set the GPIO current values. Also returns back the current values.
             The values are valid only for the pins previously configured as GPIOs.
Parameters:
Inputs:
    - void *handle              - The pointer to the device handle. Cannot be NULL.
    - unsigned int gpioSetVal   - GPIO pin new values.
Outputs:
    - unsigned int *pgpioPinVal - GPIO pin values reported back, if successful call.
                                  In case of error the value is not changed.
                                  Cannot be NULL pointer.
Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_UNKOWN_ERROR, E_ERR_INVALID_PARAMETER
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_GetGpioConfig

```
int Mcp2210_GetGpioConfig(void* handle, unsigned char cfgSelector, unsigned char *pGpioPinDes,
                          unsigned int *pdfltGpioOutput, unsigned int *pdfltGpioDir,
                          unsigned char *prmtWkupEn, unsigned char *pintPinMd,
                          unsigned char *pspiBusRelEn)
====================================================================================================
Description: Provides the current GPIO configuration or the power-up default (NVRAM) GPIO configuration.
Parameters:
Inputs:
    - void *handle                 - The pointer to the device handle. Cannot be NULL.
    - unsigned char cfgSelector    - current/power-up chip setting selection. Valid values are:
                                     - MCP2210_VM_CONFIG: designates current chip setting (Volatile
                                       Memory)
                                     - MCP2210_NVRAM_CONFIG: designates power-up chip setting (NVRAM)
Outputs:
    - unsigned char *pGpioPinDes    - GPIO Pin Designation array. Cannot be NULL.
                                     Array length is MCP2210_GPIO_NR.
                                     Valid values for pin designation are:
                                     - MCP2210_PIN_DES_GPIO
                                     - MCP2210_PIN_DES_CS
                                     - MCP2210_PIN_DES_FN
    - unsigned int *pdfltGpioOutput - GPIO pin output values. Cannot be NULL
    - unsigned int *pdfltGpioDir    - GPIO pin direction. Cannot be NULL
    - unsigned char *prmtWkupEn     - remote wake-up setting. Cannot be NULL
                                     Valid values:
                                     - MCP2210_REMOTE_WAKEUP_ENABLED
                                     - MCP2210_REMOTE_WAKEUP_DISABLED
    - unsigned char *pintPinMd      - interrupt pulse count mode. Cannot be NULL. Valid values are:
                                     - MCP2210_INT_MD_CNT_HIGH_PULSES
                                     - MCP2210_INT_MD_CNT_LOW_PULSES
                                     - MCP2210_INT_MD_CNT_RISING_EDGES
                                     - MCP2210_INT_MD_CNT_FALLING_EDGES
                                     - MCP2210_INT_MD_CNT_NONE
    - unsigned char *pspiBusRelEn   - SPI Bus Release option. Cannot be NULL. Valid values are:
                                     - MCP2210_SPI_BUS_RELEASE_ENABLED
                                     - MCP2210_SPI_BUS_RELEASE_DISABLED
Returns:
    - 0 for success:        E_SUCCESS
    - negative error code:  E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                            E_ERR_UNKOWN_ERROR, E_ERR_INVALID_PARAMETER
                            E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO
                            E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_SetGpioConfig

```
int Mcp2210_SetGpioConfig(void *handle, unsigned char cfgSelector, unsigned char *pGpioPinDes,
                          unsigned int dfltGpioOutput, unsigned int dfltGpioDir,
                          unsigned char rmtWkupEn, unsigned char intPinMd,
                          unsigned char spiBusRelEn)
====================================================================================================
Description: Set the current GPIO configuration or the power-up default (NVRAM) GPIO configuration.
If the chip NVRAM is password protected, the power-up GPIO configuration cannot be executed, the API
returns  E_ERR_BLOCKED_ACCESS.
Parameters:
Inputs:
    - void *handle                  - The pointer to the device handle. Cannot be NULL.
    - unsigned char cfgSelector     - current/power-up chip setting selection. Valid values are:
                                        - MCP2210_VM_CONFIG: designates current chip setting (Volatile
                                          Memory)
                                        - MCP2210_NVRAM_CONFIG: designates power-up chip setting (NVRAM)
    - unsigned char *pGpioPinDes    - GPIO Pin Designation array. Cannot be NULL.
                                      Array length is MCP2210_GPIO_NR
                                      Valid values for pin designation are:
                                        - MCP2210_PIN_DES_GPIO
                                        - MCP2210_PIN_DES_CS
                                        - MCP2210_PIN_DES_FN
    - unsigned int dfltGpioOutput   - GPIO pin output values.
    - unsigned int dfltGpioDir      - GPIO pin direction.
    - unsigned char rmtWkupEn       - remote wake-up setting. Valid values:
                                        - MCP2210_REMOTE_WAKEUP_ENABLED
                                        - MCP2210_REMOTE_WAKEUP_DISABLED
    - unsigned char intPinMd        - interrupt pulse count mode. Valid values are:
                                        - MCP2210_INT_MD_CNT_HIGH_PULSES
                                        - MCP2210_INT_MD_CNT_LOW_PULSES
                                        - MCP2210_INT_MD_CNT_RISING_EDGES
                                        - MCP2210_INT_MD_CNT_FALLING_EDGES
                                        - MCP2210_INT_MD_CNT_NONE
    - unsigned char spiBusRelEn    - SPI Bus Release option. Valid values are:
                                        - MCP2210_SPI_BUS_RELEASE_ENABLED
                                        - MCP2210_SPI_BUS_RELEASE_DISABLED
Outputs:
    - none
Returns:
    - 0 for success:         E_SUCCESS
    - negative error code:   E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                             E_ERR_UNKOWN_ERROR, E_ERR_INVALID_PARAMETER
                             E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO
                             E_ERR_BLOCKED_ACCESS
                             E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_GetInterruptCount

```
int Mcp2210_GetInterruptCount(void *handle, unsigned int *pintCnt, unsigned char reset)
====================================================================================================
Description: Reads the event counter value. It may also reset the counter, depending on the reset
parameter value. The function returns negative error code if something went wrong.
Note: In order to have the MCP2210 record the number of external interrupt events,
GP6 must be configured to have its dedicated function active.

Parameters:
Inputs:
    - void *handle                  - The pointer to the device handle. Cannot be NULL.
    - unsigned char reset           - if set to 0, the counter is reset. Any other value has no effect.
Outputs:
    - unsigned int *pintCnt         - the value of the counter if the function returns E_SUCCESS.
                                    In case of error *pintCnt is not modified.
                                    Note that the counter has 16bit resolution, so the maximum
                                    returned value is 65535.
                                    Cannot be NULL.

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_UNKOWN_ERROR
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_GetSpiConfig

```
int Mcp2210_GetSpiConfig(void *handle, unsigned char cfgSelector, unsigned int *pbaudRate,
                            unsigned int *pidleCsVal, unsigned int *pactiveCsVal,
                            unsigned int *pCsToDataDly, unsigned int *pdataToCsDly,
                            unsigned int *pdataToDataDly, unsigned int *ptxferSize,
                            unsigned char *pspiMd)
====================================================================================================
Description: Get the SPI transfer settings for current (VM) configuration or for power-up
default (NVRAM) SPI configuration.
Parameters:
Inputs:
    - void *handle                  - The pointer to the device handle. Cannot be NULL.
    - unsigned char cfgSelector     - current/power-up chip setting selection. Valid values are:
                                        - MCP2210_VM_CONFIG: designates current chip setting (Volatile
                                          Memory)
                                        - MCP2210_NVRAM_CONFIG: designates power-up chip setting (NVRAM)
Outputs:
    - unsigned int *pbaudRate       - transfer speed. Cannot be NULL.
    - unsigned int *pidleCsVal      - IDLE chip select value. Cannot be NULL.
                                       bit31  -  -  -  -  bit8                                    bit0
                                          x  x  ...  x  x  CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0
    - unsigned int *pactiveCsVal    - Active Chip Select Value. Cannot be NULL.
                                      bit31  -  -  -  -  bit8                                     bit0
                                          x  x  ...  x  x  CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0
    - unsigned int *pCsToDataDly    - Chip Select to Data Delay (quanta of 100 us)
                                        - 16-bit value => max value is (2^16-1).
                                        - Cannot be NULL.
    - unsigned int *pdataToCsDly    - Last Data Byte to CS (de-asserted) Delay (quanta of 100 us)
                                        - 16-bit value => max value is (2^16)-1.
                                        - Cannot be NULL.
    - unsigned int *pdataToDataDly  - Delay Between Subsequent Data Bytes (quanta of 100 us)
                                        - 16-bit value => max value is (2^16)-1.
                                        - Cannot be NULL.
    - unsigned int *ptxferSize      - Bytes to Transfer per SPI Transaction.
                                        - 16-bit value => max value is (2^16)-1.
                                        - Cannot be NULL.
    - unsigned char *pspiMd         - SPI Mode. Cannot be NULL.
                                        Valid values are:
                                        - MCP2210_SPI_MODE0
                                        - MCP2210_SPI_MODE1
                                        - MCP2210_SPI_MODE2
                                        - MCP2210_SPI_MODE3
Returns:
    - 0 for success:         E_SUCCESS
    - negative error code:   E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                             E_ERR_UNKOWN_ERROR, E_ERR_INVALID_PARAMETER
                             E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO
                             E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_SetSpiConfig

```
int Mcp2210_SetSpiConfig(void *handle, unsigned char cfgSelector, unsigned int *pbaudRate,
                            unsigned int *pidleCsVal, unsigned int *pactiveCsVal,
                            unsigned int *pCsToDataDly, unsigned int *pdataToCsDly,
                            unsigned int *pdataToDataDly, unsigned int *ptxferSize,
                            unsigned char *pspiMd)
====================================================================================================
Description: Set the SPI transfer settings for current (VM) configuration or for power-up
default (NVRAM) SPI configuration.
If SPI transfer is in progress, the configuration change is not accepted and the API returns
E_ERR_SPI_CFG_ABORT error.
In the case of the current (VM) SPI configuration change, the API call returns the actual configuration
values that will be used by the device, derived from the user supplied ones but adapted to the
device capabilities.
If the NVRAM is password protected, the power-up SPI configuration cannot be changed and
the API returns E_ERR_BLOCKED_ACCESS error.
Parameters:
Inputs:
    - void *handle                  - The pointer to the device handle. Cannot be NULL.
    - unsigned char cfgSelector     - current/power-up chip setting selection. Valid values are:
                                        - MCP2210_VM_CONFIG: designates current chip setting (Volatile
                                          Memory)
                                        - MCP2210_NVRAM_CONFIG: designates power-up chip setting (NVRAM)
Inputs / Outputs:
    - unsigned int *pbaudRate       - transfer speed. Cannot be NULL.
    - unsigned int *pidleCsVal      - IDLE chip select value. Cannot be NULL.
                                       bit31  -  -  -  -  bit8                                 bit0
                                          x  x  ...  x  x  CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0
    - unsigned int *pactiveCsVal    - Active Chip Select Value. Cannot be NULL.
                                      bit31  -  -  -  -  bit8                                  bit0
                                          x  x  ...  x  x  CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0
    - unsigned int *pCsToDataDly    - Chip Select to Data Delay (quanta of 100 us)
                                        - 16-bit value => max value is (2^16-1).
                                        - Cannot be NULL.
    - unsigned int *pdataToCsDly    - Last Data Byte to CS (de-asserted) Delay (quanta of 100 us)
                                        - 16-bit value => max value is (2^16)-1.
                                        - Cannot be NULL.
    - unsigned int *pdataToDataDly  - Delay Between Subsequent Data Bytes (quanta of 100 us)
                                        - 16-bit value => max value is (2^16)-1.
                                        - Cannot be NULL.
    - unsigned int *ptxferSize      - Bytes to Transfer per SPI Transaction.
                                        - 16-bit value => max value is (2^16)-1.
                                        - Cannot be NULL.
    - unsigned char *pspiMd         - SPI Mode. Cannot be NULL.
                                        Valid values are:
                                        - MCP2210_SPI_MODE0
                                        - MCP2210_SPI_MODE1
                                        - MCP2210_SPI_MODE2
                                        - MCP2210_SPI_MODE3
Returns:
    - 0 for success:        E_SUCCESS
    - negative error code:  E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                            E_ERR_UNKOWN_ERROR, E_ERR_INVALID_PARAMETER
                            E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO
                            E_ERR_BLOCKED_ACCESS, E_ERR_SPI_CFG_ABORT
```

Mcp2210_xferSpiData

```
int Mcp2210_xferSpiData(void *handle, unsigned char *pdataTx, unsigned char *pdataRx,
                        unsigned int *pbaudRate, unsigned int *ptxferSize, unsigned int csmask)
====================================================================================================
Description: Data transfer over SPI bus. This API assumes that GPIO configuration and SPI
communication settings are already configured, either as power-up default settings or through the use
of Mcp2210_SetGpioConfig() and Mcp2210_SetSPIConfig(). In addition, this API offers the user the
option to change the current (VM) SPI transfer length and/or to indicate which GPIO pin(s) to be used
as CS pin(s). Please note that these changes are not reverted at the end of the SPI data transfer but
remain in effect until a new GPIO/SPI configuration change is initiated by the user.

If SPI configuration change is not accepted (because SPI transfer is in progress) the API returns
E_ERR_SPI_CFG_ABORT error.
In the case of the current (VM) SPI configuration change, the API call returns the actual configuration
values that will be used by the device, derived from the user supplied ones but adapted to the
device capabilities.
This API can also reverse the effect of the GP8 firmware error: when GP8 is set as GPIO "out",
the SPI transfer command is changing it to GPIO "in". The user has the option to skip this fix if the
application execution speed gain is relevant.

Parameters:
Inputs:
    - void *handle                  - The pointer to the device handle. Cannot be NULL.
    - unsigned char *pdataTx        - data buffer to be transmitted. Cannot be NULL.
                                      The user must ensure that buffer size matches the SPI transfer
                                      size setting.
    - unsigned int *pbaudRate       - SPI transfer speed. Cannot be NULL.
                                      - If the value is different than the current speed,
                                      the API attempts to change it as requested.
                                      - If the value is 0, the current SPI transfer speed is preserved.
    - unsigned int *ptxferSize      - Bytes to transfer per SPI transaction.
                                        - 16-bit value => max valid value is (2^16)-1.
                                        - A larger values than (2^16)-1 indicates that
                                          the current SPI transfer size setting must remain unchanged.
                                        - If set to 0, all the configuration changes are accepted if
                                          valid (baudRate, CS/GPIO configuration, transfer size set 0),
                                          but no SPI transfer is performed.
                                        - Cannot be NULL.
    - unsigned int csmask           - bit mask indicating which GPIO pins must be active chip select
                                        (CS) pins for the SPI transfer. It is assumed that CS settings
                                        (idle/active values and delays) are properly configured
                                        already either as power-up defaults or with prior use
                                        of the Mcp2210_SetSPIConfig() API.
                                       bit31  -   ...  -  -  bit8                                    bit0
                                       GP8CE  x   ...  x  x   CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0
                                        - A bit value of '1' configures the GPIO pins as CS
                                          (alternate function 1) during the SPI data transfer.
                                          A bit value of 0 configures the pins as GPIO (standard
                                          function) except the pins which are configuration as
                                          dedicated function (alternate function 2).
                                          The GP direction and default values must be already configured
                                          as needed either as power-up defaults or with prior use of
                                          Mcp2210_SetGpioConfig().
                                        - If all CS[0..8] bit values are '0', current GPIO configuration
                                          is preserved.
                                        - If csmask[31] bit (GP8CE) is set ('1'), the GP8 errata is
                                          ignored, assuming that user application is not affected by the
                                          GP8 pin being reconfigured as GPIO "input" after the SPI
                                          transfer command.
Outputs:
    - unsigned char *pdataRx        - receive data buffer. Cannot be NULL.
                                      The user must ensure that buffer size matches the SPI transfer
                                      size setting.
    - unsigned int *pbaudRate       - If the function call returns E_SUCCESS, this parameter contains
                                      the current transfer speed value accepted by the device.
                                      The value may differ from the user supplied one - it may be
                                      derived from the user supplied value but adapted to the device
                                      capabilities.
    - unsigned int *ptxferSize      - If the function call returns E_SUCCESS, this parameter contains
                                      the current transfer size used by the device.
Returns:
    - 0 for success:        E_SUCCESS
    - negative error code:  E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                            E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO
                            E_ERR_UNKOWN_ERROR, E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
                            E_ERR_SPI_TIMEOUT, E_ERR_SPI_EXTERN_MASTER, E_ERR_SPI_RX_INCOMPLETE
                            E_ERR_SPI_CFG_ABORT
```

Mcp2210_xferSpiDataEx

```
int Mcp2210_xferSpiDataEx(
    void *handle, unsigned char *pdataTx, unsigned char *pdataRx,
    unsigned int *pbaudRate, unsigned int *ptxferSize, unsigned int csmask,
    unsigned int *pidleCsVal, unsigned int *pactiveCsVal, unsigned int *pCsToDataDly,
    unsigned int *pdataToCsDly, unsigned int *pdataToDataDly, unsigned char *pspiMd)
====================================================================================================
Description: This function configures the SPI transfer settings (VM parameters only) and initiate the
SPI data transfer. Basically, this API combines the Mcp2210_SetSPIConfig() and Mcp2210_xferSpiData()
function calls. Also, this function allows the user to specify which GPIO pins to be configured as
SPI chip select, but it is assumed that the other GPIO pins configuration is already done, either as
power-up default settings or through the use of Mcp2210_SetGpioConfig() or the other GPIO related API.
If the function call succeeds (returns E_SUCCESS), the parameters listed below as "Inputs/outputs"
contain the configuration values used by device. In some cases, the values are derived from the user
supplied ones, being adapted to the device capabilities (e.g. the baudRate).
Please note that these changes are not reverted at the end of the SPI data transfer but
remain in effect until a new GPIO/SPI configuration change is initiated by the user.
Mind that, if there is already an SPI transfer in progress, the configuration change is not accepted
and the API returns E_ERR_SPI_CFG_ABORT error.
This API can also reverse the effect of the GP8 firmware error: when GP8 is set as GPIO "out",
the SPI transfer command is changing it to GPIO "in". The user has the option to skip this fix if the
application execution speed gain is relevant.

Parameters:
Inputs:
    - void *handle                  - The pointer to the device handle. Cannot be NULL.

    - unsigned char *pdataTx        - data buffer to be transmitted. Cannot be NULL.
                                      The user must ensure that buffer size matches the SPI transfer
                                      size setting.

    - unsigned int csmask           - bit mask indicating which GPIO pins must be active chip select
                                      (CS) pins for the SPI transfer.
                                      bit31  -  ...  -  -  bit8                                    bit0
                                      GP8CE  x  ...  x  x   CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0
                                      - A bit value of '1' configures the GPIO pins as CS
                                        (alternate function 1) during the SPI data transfer.
                                        A bit value of 0 configures the pins as GPIO (standard function)
                                        except the pins which are configuration as
                                        dedicated function (alternate function 2).
                                        The GPIO direction and default values must be already configured
                                        as needed either as power-up defaults or with prior use of
                                        Mcp2210_SetGpioConfig().
                                      - If all CS[0..8] bit values are '0', current GPIO configuration
                                        is preserved.
                                      - If csmask[31] bit (GP8CE) is set ('1'), the GP8 errata is
                                        ignored, assuming that user application is not affected by the
                                        GP8 pin being reconfigured as GPIO "input" after the SPI
                                        transfer command.
Output:
    - unsigned char *pdataRx        - receive data buffer. Cannot be NULL.
                                      The user must ensure that buffer size matches the SPI transfer
                                      size setting.

Inputs / Outputs:
    - unsigned int *pbaudRate       - transfer speed. Cannot be NULL.

    - unsigned int *ptxferSize      - Bytes to Transfer per SPI Transaction.
                                      - 16-bit value => max value is (2^16)-1.
                                      - Cannot be NULL.
    - unsigned int *pidleCsVal      - IDLE chip select value. Cannot be NULL.
                                       bit31  -  -  -  -  bit8                                    bit0
                                          x  x  ...  x  x  CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0

    - unsigned int *pactiveCsVal    - Active Chip Select Value. Cannot be NULL.
                                      bit31  -  -  -  -  bit8                                     bit0
                                          x  x  ...  x  x  CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0

    - unsigned int *pCsToDataDly    - Chip Select to Data Delay (quanta of 100 us)
                                        - 16-bit value => max value is (2^16-1).
                                        - Cannot be NULL.

    - unsigned int *pdataToCsDly    - Last Data Byte to CS (de-asserted) Delay (quanta of 100 us)
                                        - 16-bit value => max value is (2^16)-1.
                                        - Cannot be NULL.

    - unsigned int *pdataToDataDly  - Delay Between Subsequent Data Bytes (quanta of 100 us)
                                        - 16-bit value => max value is (2^16)-1.
                                        - Cannot be NULL.

    - unsigned char *pspiMd         - SPI Mode. Cannot be NULL.
                                      Valid values are:
                                      - MCP2210_SPI_MODE0
                                      - MCP2210_SPI_MODE1
                                      - MCP2210_SPI_MODE2
                                      - MCP2210_SPI_MODE3

Returns:
    - 0 for success:        E_SUCCESS
    - negative error code:  E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE, E_ERR_INVALID_PARAMETER
                            E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                            E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
                            E_ERR_SPI_CFG_ABORT, E_ERR_SPI_TIMEOUT, E_ERR_SPI_EXTERN_MASTER,
                            E_ERR_SPI_RX_INCOMPLETE
```

Mcp2210_CancelSpiTxfer

```
int Mcp2210_CancelSpiTxfer(void *handle, unsigned char *pspiExtReqStat, unsigned char *pspiOwner)
====================================================================================================
Description: Cancel the current SPI transfer and provides information about:
- SPI Bus release external request status,
- SPI Bus ownership prior the command (who is the Master).
The command also releases the bus ownership.

Parameters:
Inputs:
    - void *handle                  - The pointer to the device handle. Cannot be NULL.

Outputs:
    - unsigned char *pspiExtReqStat - SPI Bus release external request status:
                                        - 0x01 – No External Request for SPI Bus Release
                                        - 0x00 – Pending External Request for SPI Bus Release
                                       Cannot be NULL.
    - unsigned char *pspiOwner      - SPI Bus current owner:
                                        - 0x00 - No owner
                                        - 0x01 - USB Bridge
                                        - 0x02 - External Master.
                                      Cannot be NULL.
Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_UNKOWN_ERROR
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_RequestSpiBusRel

```
int Mcp2210_RequestSpiBusRel(void *handle, unsigned char ackPinVal)
====================================================================================================
Description: Request the release of the SPI bus and set the value of the "SPI Bus Release ACK" pin
if GP7 is assigned to this dedicated function and if the bus can be released.
If SPI transfer is in progress, the bus is not released, the command returns E_ERR_SPI_XFER_ONGOING
errorcode and the host application should either wait the transfer completion or cancel it
(by using Mcp2210_CancelSpiTxfer) and request again the SPI bus release.

Parameters:
Inputs:
    - void *handle                  - The pointer to the device handle. Cannot be NULL.
    - unsigned char ackPinVal       - the value of the "SPI Bus Release ACK" pin if the
                                      command succeeds and the GP7 is assigned to alternate function 2.
Outputs:
    - none

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE, E_ERR_INVALID_PARAMETER
                           E_ERR_UNKOWN_ERROR
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
                           E_ERR_SPI_XFER_ONGOING
```

Mcp2210_GetSpiStatus

```
int Mcp2210_GetSpiStatus(void *handle, unsigned char *pspiExtReqStat, unsigned char *pspiOwner,
                         unsigned char *pspiTxferStat);
====================================================================================================
Description: If there is no device communication error, this command returns 0 (E_SUCCESS) and
updates the pointers with information about SPI transfer status. In case of error, the function
returns negative error code.

Parameters:
Inputs:
    - void *handle                   - The pointer to the USB device handle. Cannot be NULL.

Outputs:
    - unsigned char *pspiExtReqStat - SPI Bus release external request status:
                                        - 0x01 - No External Request for SPI Bus Release
                                        - 0x00 - Pending External Request for SPI Bus Release
                                       Cannot be NULL.
    - unsigned char *pspiOwner       - SPI Bus current owner:
                                        - 0x00 - No owner
                                        - 0x01 - USB Bridge
                                        - 0x02 - External Master.
                                      Cannot be NULL.
    - unsigned char *pspiTxferStat   - SPI transfer status:
                                        - 0x00 - No SPI transfer in progress
                                        - 0x01 - SPI transfer is in progress
                                     Cannot be NULL.

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_ReadEEProm

```
int Mcp2210_ReadEEProm(void *handle, unsigned char address, unsigned char *pcontent)
====================================================================================================
Description: Read the byte at the EEPROM memory address provided as parameter.
The function returns negative error code in case of failure.
The EEPROM size is 256 bytes.

Parameters:
Inputs:
    - void *handle                  - The pointer to the USB device handle. Cannot be NULL.
    - unsigned char address         - the EEPROM location to be read.
Outputs:
    - unsigned char *pcontent       - the value stored at the EEPROM address, if the function returns
                                      E_SUCCESS. In case of failure, *pcontent is not changed.
                                      The parameter cannot be NULL.

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_WriteEEProm

```
int Mcp2210_WriteEEProm(void *handle, unsigned char address, unsigned char content)
====================================================================================================
Description: Write one byte at the EEPROM memory address provided as parameter.
The EEPROM size is 256 bytes.
The function returns negative error code in case of failure.
If the error code is "E_ERR_BLOCKED_ACCESS" the write failure is due to the device being either
password protected or permanently locked. Mcp2210_GetAccessStatus() API provides the details about
device protection status. If the device is not permanently locked, the password protection can be
unlocked or disabled by using the access control API.


Parameters:
Inputs:
    - void *handle              - The pointer to the USB device handle. Cannot be NULL.
    - unsigned char address     - the EEPROM location to write to.
    - unsigned char content     - the value to store at the EEPROM address.

Outputs:
    - none

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
                           E_ERR_BLOCKED_ACCESS
```

Mcp2210_GetAccessCtrlStatus

```
int Mcp2210_GetAccessCtrlStatus(void *handle, unsigned char *pAccessCtrl,
                                unsigned char *pPasswdAttemptCnt, unsigned char *pPasswdAccepted)
====================================================================================================
Description: This command provides information about the NVARM access control status.
In case of successful execution, the function returns 0 (E_SUCCESS) and updates the pointer
parameters with the status codes. In case of failure, negative error code is returned.

Parameters:
Inputs:
    - void *handle                      - The pointer to the USB device handle. Cannot be NULL.

Outputs:
    - unsigned char *pAccessCtrl        - NVRAM chip settings access control:
                                            - 0x00 - settings not protected
                                            - 0x40 - settings protected by Password
                                            - 0x80 - settings permanently locked
                                            Parameter cannot be NULL.

    - unsigned char *pPasswdAttemptCnt  - counts how many times the NVRAM password was tried.
                                          The value will saturate to 5 even if the number of wrong
                                          attempts exceeds 5. After the password is accepted, the
                                          counter is reset to 0.
                                          Parameter cannot be NULL.

    - unsigned char *pPasswdAccepted    - If the NVRAM access is password protected
                                          (indicated by *pAccessCtrl == 0x40), this flag indicates
                                          whether the password has been accepted or not after using the
                                          Mcp2210_EnterAccessPasswd() command:
                                            - 0x00 - Password NOT accepted
                                            - 0x01 - Password accepted.
                                        - If the NVRAM access is not password protected
                                          (indicated by *pAccessCtrl == 0x00), this flag returns 0x00.
                                          The parameter cannot be NULL.

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_EnterPassword

```
int Mcp2210_EnterPassword(void *handle, char *passwd)
====================================================================================================
Description: Sends the NVRAM unlock password to the device.
The function returns 0 (E_SUCCESS) either if the password is accepted or if the device is not
password protected. In both cases, the NVRAM changes are permitted.
The function returns negative error code in case of device communication failure or if the password
is not accepted by the device.
Rejected password error codes:
    - E_ERR_NVRAM_LOCKED        - NVRAM is permanently locked, no password is accepted
    - E_ERR_WRONG_PASSWD        - password mismatch, but number of attempts is less than 5
    - E_ERR_ACCESS_DENIED       - password mismatch, but the number of attempts exceeded 5,
                                  so the NVRAM writes are blocked until the next device reset

Parameters:
Inputs:
    - void *handle         - The pointer to the USB device handle. Cannot be NULL.
    - char *passwd         - string of 8 characters (8 bytes), NULL terminated.
                           !!! Important !!!
                           The password must have exactly 8 characters. If there is a different
                           number of characters, -2 (E_ERR_INVALID_PARAMETER) error code is returned.
                           The pointer parameter cannot be NULL.
Outputs:
    - none

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE, E_ERR_INVALID_PARAMETER
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                           E_ERR_NVRAM_LOCKED, E_ERR_WRONG_PASSWD, E_ERR_ACCESS_DENIED
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_SetAccessControl

```
int Mcp2210_SetAccessControl(void *handle, unsigned char accessConfig, char *currentPasswd, char
*newPasswd)
====================================================================================================
Description: Configures the NVRAM password protection.
The function returns 0 (E_SUCCESS) if the new access control configuration is accepted
or negative error code in case of device communication failure or if the access control configuration
change is not accepted by the device.
Error codes in case of rejected configuration change:
    - E_ERR_NVRAM_LOCKED        - NVRAM is permanently locked, no access control change is accepted
    - E_ERR_WRONG_PASSWD        - old password mismatch, but the number of attempts is less than 5
    - E_ERR_ACCESS_DENIED       - old password mismatch, but the number of attempts exceeded 5,
                                  so the NVRAM writes are blocked until the next device reset

!!! Important notes !!!
1. The password strings must have exactly 8 characters.
If they have a different number of characters, -2 (E_ERR_INVALID_PARAMETER) error code is returned.

2. The new password string becomes effective after the next chip reset or power cycle. Therefore,
if maximum protection is required, a chip reset is recommended after calling Mcp2210_SetAccessControl()
with accessConfig parameter set to MCP2210_NVRAM_PROTECTED or MCP2210_NVRAM_PASSWD_CHANGE.
The chip reset can be achieved using Mcp2210_Reset() API.

Parameters:
Inputs:
    - void *handle          - The pointer to the USB device handle. Cannot be NULL.
    - unsigned char accessConfig
                            - indicates which type of access control change is performed by the API call.
                                 - 0x0 (MCP2210_NVRAM_NO_PROTECTION) - disable the password protection:
                                   If the password protection is enabled, the currentPassword must be
                                   valid, otherwise the function returns negative error code:
                                   E_ERR_WRONG_PASSWD or E_ERR_ACCESS_DENIED
                                   If the password protection is already disabled, the function
                                   ignores the currentPassword and returns E_SUCCESS.

                                 - 0x40 (MCP2210_NVRAM_PROTECTED) - enable the password protection:
                                   If the password protection is not enabled, newPassword must be a valid
                                   password string.
                                   If the password protection is already enabled, newPassword is ignored
                                   and the function returns negative error code: E_ERR_NVRAM_PROTECTED.

                                 - 0xA5 (MCP2210_NVRAM_PASSWD_CHANGE) - change the current password with
                                   the new one if the password protection is already enabled.
                                   Both currentPasswd and newPasswd must be valid password strings.
                                   If the password protection is not enabled, the function returns
                                   negative error code: E_ERR_PASSWD_CHANGE

    - char *currentPasswd     - string of 8 characters (8 bytes), NULL terminated.
    - char *newPasswd         - string of 8 characters (8 bytes), NULL terminated.
Outputs:
    - none

Returns:
    - 0 for success:        E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE, E_ERR_INVALID_PARAMETER
                            E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                            E_ERR_NVRAM_LOCKED, E_ERR_WRONG_PASSWD, E_ERR_ACCESS_DENIED
                            E_ERR_NVRAM_PROTECTED, E_ERR_PASSWD_CHANGE
                            E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Mcp2210_SetPermanentLock

```
int Mcp2210_SetPermanentLock(void *handle)
====================================================================================================
Description: Permanently locks the device NVRAM settings.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!    WARNING    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
USE THIS FUNCTION WITH GREAT CAUTION. THERE IS NO WAY TO REMOVE THE PERMANENT LOCK
THE CHIP POWER-UP DEFAULT SETTINGS CANNOT BE CHANGED AFTER THIS FUNCTION CALL
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

The function returns 0 (E_SUCCESS) if the lock configuration is accepted
or negative error code in case of device communication failure or if the access control configuration
change is not accepted by the device.

Error codes in case of rejected configuration change:
    - E_ERR_NVRAM_LOCKED        - NVRAM is already permanently locked
    - E_ERR_BLOCKED_ACCESS      - NVRAM is password protected. NVRAM change protection must be
                                  deactivated first, using Mcp2210_EnterPassword() or
                                  Mcp2210_SetAccessControl()

Parameters:
Inputs:
    - void *handle         - The pointer to the USB device handle. Cannot be NULL.

Outputs:
    - none

Returns:
    - 0 for success:       E_SUCCESS
    - negative error code: E_ERR_NULL, E_ERR_INVALID_HANDLE_VALUE, E_ERR_INVALID_PARAMETER
                           E_ERR_HID_TIMEOUT, E_ERR_HID_RW_FILEIO, E_ERR_UNKOWN_ERROR
                           E_ERR_NVRAM_LOCKED, E_ERR_BLOCKED_ACCESS
                           E_ERR_CMD_ECHO, E_ERR_CMD_FAILED
```

Annex 2 - MCP2210 Managed DLL API

M_Mcp2210_GetLibraryVersion

```
int M_Mcp2210_GetLibraryVersion(StringBuilder ^version)
===================================================================================================
Description: Provides the library version as StringBuilder object.
Parameters:
Inputs:
    - none
Outpus:
    - StringBuilder ^version - reference to string.

Returns:
    - positive value: version string size (bytes) including the null character
```

M_Mcp2210_GetLastError

```
int M_Mcp2210_GetLastError();
=====================================================================================================
Description: Returns the detailed error code of the last library function call whose prototype is
not making this possible. The library functions that need this support are:
- M_Mcp2210_OpenByIndex()
- M_Mcp2210_OpenBySN()
!!! Important note !!!
In order to avoid the last error code being altered by other API calls, please make sure that
M_Mcp2210_GetLastError() is called immediately after the API listed above and save the returned value
accordingly.

Parameters:
Inputs:
    - none
Outputs:
    - none
Returns:
    - M_E_SUCCESS or negative error code: please see the description of the above listed API calls
```

M_Mcp2210_GetConnectedDevCount

```
int M_Mcp2210_GetConnectedDevCount(unsigned short vid, unsigned short pid)
====================================================================================================
Description: provides the number of connected devices with the given VID:PID.
Returns negative number in case of error.
Parameters:
Inputs:
    - unsigned short vid - MCP2210 device VendorID (Microchip default = 0x4D8)
    - unsigned short pid - MCP2210 device ProductID (Microchip default = 0xDE)
Outputs:
    - none
Returns:
    - the number of connected devices with the given VID:PID or
    - negative error code: M_E_ERR_UNKOWN_ERROR, M_E_ERR_MALLOC
```

M_Mcp2210_OpenByIndex

```
IntPtr M_Mcp2210_OpenByIndex(unsigned short vid, unsigned short pid,
                             unsigned int index, StringBuilder ^devPath)
====================================================================================================
Description:  Opens the connection with the MCP2210 device identified by the given VID:PID
and the index number and returns the file handle.
Also, the function can provide the device path as UNICODE string, if a valid reference
to a StringBuilder object is provided.
If the system cannot allocate enough memory for the path string, the function call fails and
the device is not opened.

!!! Important note !!!
In case of failure, the function returns an invalid handle value (-1) and updates an internal variable
with the specific error code (negative values). This can be retrieved with M_Mcp2210_GetLastError()
call.
The error codes that Mcp2210_GetLastError() might return after M_Mcp2210_OpenByIndex() call are:
    - M_E_SUCCESS (0), if no error occurred
    - M_E_ERR_BUFFER_TOO_SMALL
    - M_E_ERR_UNKOWN_ERROR, M_E_ERR_MALLOC, M_E_ERR_NULL,
      M_E_ERR_DEVICE_NOT_FOUND, M_E_ERR_NO_SUCH_INDEX,
      M_E_ERR_CONNECTION_ALREADY_OPENED, M_E_ERR_OPEN_DEVICE_ERROR

Parameters:
Inputs:
    - unsigned short vid         - MCP2210 device VendorID (Microchip default = 0x4D8)
    - unsigned short pid         - MCP2210 device ProductID (Microchip default = 0xDE)
    - unsigned int   index       - the index of the MCP2210 device to connect to. This value ranges
                                   from 0 to n-1, where n is the number of connected devices.
                                   This value can be obtained with "M_Mcp2210_GetConnectedDevCount"

Outputs:
    - StringBuilder ^devPath     - path string.
                                   If ^devPath is NULL, the path is not reported but
                                   the device is opened and the function returns the handle.

Returns:
    - valid handle value in case of success
    - INVALID_HANDLE_VALUE (-1) in case of error.
```

M_Mcp2210_OpenBySN

```
IntPtr M_Mcp2210_OpenBySN(unsigned short vid, unsigned short pid,
                          String ^serialNo, StringBuilder ^devPath)
====================================================================================================
Description:  Opens the connection with the MCP2210 device identified by the given VID:PID
and the serial number string and returns the file handle.
Also, the function can provide the device path as NULL terminated UNICODE string, if a valid reference
to a StringBuilder object is provided.
If the system cannot allocate enough memory for the path string, the function call fails and
the device is not opened.

!!! Important note !!!
In case of failure, the function returns an invalid handle value (-1) and updates an internal variable
with the specific error code (negative values). This can be retrieved with M_Mcp2210_GetLastError()
call.
The error codes that M_Mcp2210_GetLastError() might return after M_Mcp2210_OpenBySN() call are:
    - M_E_SUCCESS (0), if no error occurred
    - M_E_ERR_BUFFER_TOO_SMALL
    - M_E_ERR_UNKOWN_ERROR, M_E_ERR_MALLOC, M_E_ERR_NULL,
      M_E_ERR_DEVICE_NOT_FOUND, M_E_ERR_NO_SUCH_SERIALNR,
      M_E_ERR_CONNECTION_ALREADY_OPENED, M_E_ERR_OPEN_DEVICE_ERROR

Parameters:
Inputs:
    - unsigned short vid         - MCP2210 device VendorID (Microchip default = 0x4D8)
    - unsigned short pid         - MCP2210 device ProductID (Microchip default = 0xDE)
    - String^  serialNo          - The serial number of the MCP2210 we want to connect to:
                                   NULL terminated string of max MPC2210_SERIAL_NUMBER_LENGTH wide
characters
                                   Cannot be NULL.
Outputs:
    - StringBuilder ^devPath     - path string.
                                   If ^devPath is NULL, the path is not reported but
                                   the device is opened and the function returns the handle.

Returns:
    - valid handle value in case of success
    - INVALID_HANDLE_VALUE (-1) in case of error.
```

M_Mcp2210_Close

```
int M_Mcp2210_Close(IntPtr handle)
====================================================================================================
Description: Attempt to close a connection to a MCP2210.
Parameters:
Inputs:
    - IntPtr handle - Device handle we'll close the connection to.
                      Cannot be NULL nor contain invalid handle value.
Outputs:
    - none
Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_CLOSE_FAILED
```

M_Mcp2210_Reset

```
int M_Mcp2210_Reset(IntPtr handle)
====================================================================================================
Description: Reset the USB device.
In case of successful execution, the function closes the device handle and returns 0 (M_E_SUCCESS).
In case of failure, negative error code is returned.
!!! Important note !!!
The user application must take into account the time needed for the device reset and for
the USB enumeration process. Trying to open the device too soon after reset may fail.
Please mind also that the time needed for USB enumeration may be different on different systems, and
may vary also on the same system, depending on the configuration, CPU load a.s.o.

Parameters:
Inputs:
    - IntPtr handle       - USB device handle. Cannot be NULL or invalid.

Outputs:
    - none

Returns:
    - 0 for success:        M_E_SUCCESS
    - negative error code:  M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                            M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                            M_E_ERR_CLOSE_FAILED
```

M_Mcp2210_GetUsbKeyParams

```
int M_Mcp2210_GetUsbKeyParams(IntPtr handle, unsigned short %pvid, unsigned short %ppid,
                              unsigned char %ppwrSrc, unsigned char %prmtWkup,
                              unsigned short %pcurrentLd
====================================================================================================
Description: Provides the USB key configuration attributes: VID, PID and power settings.
In case of successful execution, the function returns 0 (M_E_SUCCESS) and updates the reference
parameters with the settings values. In case of failure, negative error code is returned.

Parameters:
Inputs:
    - IntPtr handle              - USB device handle. Cannot be NULL.

Outputs:
    - unsigned short %pvid       - device VID

    - unsigned short %ppid       - device PID

    - unsigned char %ppwrSrc     - device power source, according to USB 2.0 specs, chapter 9:
                                       - bit 7 = Host-Powered: 1=yes.
                                         This bit is reserved and must be set to 1 for historical
                                         reasons.
                                       - bit 6 = Self-Powered: 1=yes; 0=no.
                                       - bits 5..0 = 0.

    - unsigned char %prmtWkup     - device is remote wake-up capable: 1=yes; 0=no.

    - unsigned short %pcurrentLd  - Maximum current consumption from the USB Host, expressed in mA.
                                    The value is a multiple of 2, ranging from 2mA to 510mA.
Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_SetUsbKeyParams

```
int M_Mcp2210_SetUsbKeyParams(IntPtr handle, unsigned short vid, unsigned short pid,
                              unsigned char pwrSrc, unsigned char rmtWkup, unsigned short currentLd)
====================================================================================================
Description: Configures the USB key attributes: VID, PID and power settings.
In case of successful execution, the function returns 0 (M_E_SUCCESS).
In case of failure, negative error code is returned.
If the error code is "M_E_ERR_BLOCKED_ACCESS" the write failure is due to the device being either
password protected or permanently locked. M_Mcp2210_GetAccessStatus() API provides the details about
device protection status. If the device is not permanently locked, the password protection can be
unlocked or disabled by using the access control API.

Parameters:
Inputs:
    - IntPtr handle              - USB device handle. Cannot be NULL.

    - unsigned short vid         - device VID

    - unsigned short  pid        - device PID

    - unsigned char pwrSrc       - device power source, according to USB 2.0 specs, chapter 9:
                                       - bit 7 = Host-Powered: 1=yes.
                                         This bit is reserved and must be set to 1 for historical
                                         reasons.
                                       - bit 6 = Self-Powered: 1=yes; 0=no.
                                       - bits 5..0 must be set to 0 (as an extra safety measure)

    - unsigned char rmtWkup      - device is remote wake-up capable: 1=yes; 0=no.

    - unsigned short currentLd   - Maximum current consumption from the USB Host, expressed in mA.
                                   The value must be a multiple of 2 value, ranging from 2mA to 510mA.
                                   If the currentLd value is not multiple by 2, the API will
                                   round it up to the next multiple by 2 number and store it in NVRAM.
Outputs:
    - none

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE, M_E_ERR_INVALID_PARAMETER
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
                           M_E_ERR_BLOCKED_ACCESS
```

M_Mcp2210_GetManufacturerString

```
int M_Mcp2210_GetManufacturerString(IntPtr handle, StringBuilder ^manufacturerStr)
====================================================================================================
Description: Read from device the USB Manufacturer string.
In case of successful execution, the function returns 0 (M_E_SUCCESS) and updates the
manufacturerStr string object.
In case of failure, negative error code is returned.
If the NVARM string descriptor is invalid, the function returns M_E_ERR_DESCRIPTOR (-400) error code.
The string descriptor is invalid if the String Descriptor ID is not 0x03 or the descriptor size
is less than 2 or larger than 60 bytes.

Parameters:
Inputs:
    - IntPtr handle                   - USB device handle. Cannot be NULL.

Outputs:
    - StringBuilder^ manufacturerStr  - StringBuilder object containing the manufacturer string.
                                        Cannot be NULL.
                                      !! Important detail !!
                                      - maximum string length is 29 UNICODE characters

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
                           M_E_ERR_STRING_DESCRIPTOR
```

M_Mcp2210_SetManufacturerString

```
int M_Mcp2210_SetManufacturerString(IntPtr handle, String ^manufacturerStr)
====================================================================================================
Description: Write the USB Manufacturer string into device NVRAM.
In case of successful execution, the function returns 0 (M_E_SUCCESS).
In case of failure, negative error code is returned.
If the error code is "M_E_ERR_BLOCKED_ACCESS" the write failure is due to the device being either
password protected or permanently locked. M_Mcp2210_GetAccessStatus() API provides the details about
device protection status. If the device is not permanently locked, the password protection can be
unlocked or disabled by using the access control API.

Parameters:
Inputs:
    - IntPtr handle             - USB device handle. Cannot be NULL.

    - String ^manufacturerStr   - String object containing the manufacturer UNICODE string.
                                  Cannot be NULL.
                                  !! Important detail !!
                                  - maximum string length must be 29 UNICODE characters without
                                    counting the NULL termination (M_MCP2210_DESCRIPTOR_STR_MAX_LEN).
                                    If the input string is larger, negative error code is returned.
Outputs:
    - none

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_STRING_TOO_LARGE
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
                           M_E_ERR_BLOCKED_ACCESS
```

M_Mcp2210_GetProductString

```
int M_Mcp2210_GetProductString(IntPtr handle, StringBuilder ^productStr)
====================================================================================================
Description: Read from device the USB Product string.
In case of successful execution, the function returns 0 (M_E_SUCCESS) and updates the
productStr string object.
In case of failure, negative error code is returned.
If the NVARM string descriptor is invalid, the function returns M_E_ERR_DESCRIPTOR (-400) error code.
The string descriptor is invalid if the String Descriptor ID is not 0x03 or the descriptor size is
less than 2 or larger than 60 bytes.

Parameters:
Inputs:
    - IntPtr handle                   - USB device handle. Cannot be NULL.

Outputs:
    - StringBuilder^ productStr       - StringBuilder object containing the product ID string.
                                        Cannot be NULL.
                                      !! Important detail !!
                                      - maximum string length is 29 UNICODE characters

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
                           M_E_ERR_STRING_DESCRIPTOR
```

M_Mcp2210_SetProductString

```
int M_Mcp2210_SetProductString(IntPtr handle, String ^productStr)
=======================================================================================================
Description: Write the USB Product string into device NVRAM.
In case of successful execution, the function returns 0 (M_E_SUCCESS).
In case of failure, negative error code is returned.
If the error code is "M_E_ERR_BLOCKED_ACCESS" the write failure is due to the device being either
password protected or permanently locked. M_Mcp2210_GetAccessStatus() API provides the details about
device protection status. If the device is not permanently locked, the password protection can be
unlocked or disabled by using the access control API.

Parameters:
Inputs:
    - IntPtr handle             - USB device handle. Cannot be NULL.

    - String ^productStr        - String object containing the product ID UNICODE string.
                                  Cannot be NULL.
                                  !! Important detail !!
                                  - maximum string length must be 29 UNICODE characters without
                                    counting the NULL termination (M_MCP2210_DESCRIPTOR_STR_MAX_LEN).
                                    If the input string is larger, negative error code is returned.
Outputs:
    - none

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_STRING_TOO_LARGE
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
                           M_E_ERR_BLOCKED_ACCESS
```

M_Mcp2210_GetSerialNumber

```
int M_Mcp2210_GetSerialNumber(IntPtr handle, StringBuilder ^serialStr)
====================================================================================================
Description: Read from device the USB Serial Number string.
In case of successful execution, the function returns 0 (M_E_SUCCESS) and updates the
serialStr string object.
In case of failure, negative error code is returned.
If the NVARM string descriptor is invalid, the function returns M_E_ERR_DESCRIPTOR (-400) error code.
The string descriptor is invalid if the String Descriptor ID is not 0x03 or the descriptor size is
less than 2 or larger than 60 bytes.

Parameters:
Inputs:
    - IntPtr handle                   - USB device handle. Cannot be NULL.

Outputs:
    - StringBuilder^ serialStr        - StringBuilder object containing the serial number string.
                                        Cannot be NULL.
                                      !! Important detail !!
                                      - maximum string length is 29 UNICODE characters

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
                           M_E_ERR_STRING_DESCRIPTOR
```

M_Mcp2210_GetGpioPinDir

```
int M_Mcp2210_GetGpioPinDir(IntPtr handle, unsigned int %pgpioDir)
====================================================================================================
Description: Provides the current GPIO pin direction. The values are valid only for the pins
             previously configured as GPIOs.
Parameters:
Inputs:
    - IntPtr handle             - The device handle. Cannot be NULL.
Outputs:
    - unsigned int% pgpioDir    - GPIO pin direction, if successful call. In case of error the value
                                  is not changed.

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_UNKOWN_ERROR, M_E_ERR_INVALID_PARAMETER
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_SetGpioPinDir

```
int M_Mcp2210_SetGpioPinDir(IntPtr handle, unsigned int gpioDir)
====================================================================================================
Description: Set the GPIO direction.
             The values are valid only for the pins previously configured as GPIOs.
Parameters:
Inputs:
    - IntPtr handle          - The device handle. Cannot be NULL.
    - unsigned int gpiodir   - GPIO new pin direction.
Outputs:
    - none

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_UNKOWN_ERROR, M_E_ERR_INVALID_PARAMETER
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_GetGpioPinVal

```
int M_Mcp2210_GetGpioPinVal(IntPtr handle, unsigned int %pgpioPinVal)
====================================================================================================
Description: Provides the current GPIO values. The values are valid only for the pins
             previously configured as GPIOs.
Parameters:
Inputs:
    - IntPtr handle             - The device handle. Cannot be NULL.
Outputs:
    - unsigned int %pgpioPinVal - GPIO pin values, if successful call. In case of error the value
                                  is not changed.

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_UNKOWN_ERROR, M_E_ERR_INVALID_PARAMETER
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_SetGpioPinVal

```
int M_Mcp2210_SetGpioPinVal(IntPtr handle, unsigned int gpioSetVal, unsigned int %pgpioPinVal)
====================================================================================================
Description: Set the GPIO current values. Also returns back the current values.
             The values are valid only for the pins previously configured as GPIOs.
Parameters:
Inputs:
    - IntPtr handle             - The device handle. Cannot be NULL.
    - unsigned int gpioSetVal   - GPIO pin new values.
Outputs:
    - unsigned int %pgpioPinVal - GPIO pin values reported back, if successful call.
                                  In case of error the value is not changed.

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_UNKOWN_ERROR, M_E_ERR_INVALID_PARAMETER
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_GetGpioConfig

```
int M_Mcp2210_GetGpioConfig(IntPtr handle, unsigned char cfgSelector, array<Byte> ^pGpioPinDes,
                            unsigned int %pdfltGpioOutput, unsigned int %pdfltGpioDir,
                            unsigned char %prmtWkupEn, unsigned char %pintPinMd,
                            unsigned char %pspiBusRelEn)
====================================================================================================
Description: Provides the current GPIO configuration or the power-up default (NVRAM) GPIO configuration.
Parameters:
Inputs:
    - IntPtr handle                 - The device handle. Cannot be NULL.
    - unsigned char cfgSelector     - current/power-up chip setting selection. Valid values are:
                                      - M_MCP2210_VM_CONFIG: designates current chip setting (Volatile
                                        Memory)
                                      - M_MCP2210_NVRAM_CONFIG: designates power-up chip setting (NVRAM)
Outputs:
    - array<Byte>^ pGpioPinDes      - GPIO Pin Designation array. Cannot be NULL.
                                      Array length must be M_MCP2210_GPIO_NR.
                                      Valid values for pin designation are:
                                        - M_MCP2210_PIN_DES_GPIO
                                        - M_MCP2210_PIN_DES_CS
                                        - M_MCP2210_PIN_DES_FN
    - unsigned int %pdfltGpioOutput - GPIO pin output values.
    - unsigned int %pdfltGpioDir    - GPIO pin direction.
    - unsigned char %prmtWkupEn     - remote wake-up setting. Valid values:
                                        - M_MCP2210_REMOTE_WAKEUP_ENABLED
                                        - M_MCP2210_REMOTE_WAKEUP_DISABLED
    - unsigned char %pintPinMd      - interrupt pulse count mode. Valid values are:
                                        - M_MCP2210_INT_MD_CNT_HIGH_PULSES
                                        - M_MCP2210_INT_MD_CNT_LOW_PULSES
                                        - M_MCP2210_INT_MD_CNT_RISING_EDGES
                                        - M_MCP2210_INT_MD_CNT_FALLING_EDGES
                                        - M_MCP2210_INT_MD_CNT_NONE
    - unsigned char %pspiBusRelEn   - SPI Bus Release option. Valid values are:
                                        - M_MCP2210_SPI_BUS_RELEASE_ENABLED
                                        - M_MCP2210_SPI_BUS_RELEASE_DISABLED
Returns:
    - 0 for success:        M_E_SUCCESS
    - negative error code:  M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                            M_E_ERR_UNKOWN_ERROR, M_E_ERR_INVALID_PARAMETER
                            M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO
                            M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_SetGpioConfig

```
int M_Mcp2210_SetGpioConfig(IntPtr handle, unsigned char cfgSelector, array<Byte> ^pGpioPinDes,
                            unsigned int dfltGpioOutput, unsigned int dfltGpioDir,
                            unsigned char rmtWkupEn, unsigned char intPinMd,
                            unsigned char spiBusRelEn)
====================================================================================================
Description: Set the current GPIO configuration or the power-up default (NVRAM) GPIO configuration.
If the chip NVRAM is password protected, the power-up GPIO configuration cannot be executed, the API
returns  M_E_ERR_BLOCKED_ACCESS.
Parameters:
Inputs:
    - IntPtr handle                 - The pointer to the device handle. Cannot be NULL.
    - unsigned char cfgSelector     - current/power-up chip setting selection. Valid values are:
                                      - M_MCP2210_VM_CONFIG: designates current chip setting (Volatile
                                        Memory)
                                      - M_MCP2210_NVRAM_CONFIG: designates power-up chip setting (NVRAM)
    - array<Byte> ^pGpioPinDes      - GPIO Pin Designation array. Cannot be NULL.
                                      Array length is M_MCP2210_GPIO_NR
                                      Valid values for pin designation are:
                                        - M_MCP2210_PIN_DES_GPIO
                                        - M_MCP2210_PIN_DES_CS
                                        - M_MCP2210_PIN_DES_FN
    - unsigned int dfltGpioOutput   - GPIO pin output values.
    - unsigned int dfltGpioDir      - GPIO pin direction.
    - unsigned char rmtWkupEn       - remote wake-up setting. Valid values:
                                        - M_MCP2210_REMOTE_WAKEUP_ENABLED
                                        - M_MCP2210_REMOTE_WAKEUP_DISABLED
    - unsigned char intPinMd        - interrupt pulse count mode. Valid values are:
                                        - M_MCP2210_INT_MD_CNT_HIGH_PULSES
                                        - M_MCP2210_INT_MD_CNT_LOW_PULSES
                                        - M_MCP2210_INT_MD_CNT_RISING_EDGES
                                        - M_MCP2210_INT_MD_CNT_FALLING_EDGES
                                        - M_MCP2210_INT_MD_CNT_NONE
    - unsigned char spiBusRelEn   - SPI Bus Release option. Valid values are:
                                        - M_MCP2210_SPI_BUS_RELEASE_ENABLED
                                        - M_MCP2210_SPI_BUS_RELEASE_DISABLED
Outputs:
    - none
Returns:
    - 0 for success:         M_E_SUCCESS
    - negative error code:  M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                            M_E_ERR_UNKOWN_ERROR, M_E_ERR_INVALID_PARAMETER
                            M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO
                            M_E_ERR_BLOCKED_ACCESS
                            M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_GetInterruptCount

```
int M_Mcp2210_GetInterruptCount(IntPtr handle, unsigned int %intCnt, unsigned char reset)
====================================================================================================
Description: Reads the event counter value. It may also reset the counter, depending on the reset
parameter value. The function returns negative error code if something went wrong.
Note: In order to have the MCP2210 record the number of external interrupt events,
GP6 must be configured to have its dedicated function active.

Parameters:
Inputs:
    - IntPtr handle                 - The device handle. Cannot be NULL.
    - unsigned char reset           - if set to 0, the counter is reset. Any other value has no effect.
Outputs:
    - unsigned int %intCnt          - the value of the counter if the function returns M_E_SUCCESS.
                                      In case of error intCnt is not modified.
                                      Note that the counter has 16 bit resolution, so the maximum
                                      returned value is 65535.
Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_GetSpiConfig

```
int M_Mcp2210_GetSpiConfig( IntPtr handle, unsigned char cfgSelector, unsigned int %pbaudRate,
                            unsigned int %pidleCsVal, unsigned int %pactiveCsVal,
                            unsigned int %pCsToDataDly, unsigned int %pdataToCsDly,
                            unsigned int %pdataToDataDly, unsigned int %ptxferSize,
                            unsigned char %pspiMd)
====================================================================================================
Description: Get the SPI transfer settings for current (VM) configuration or for power-up
default (NVRAM) SPI configuration.
Parameters:
Inputs:
    - IntPtr handle                - The device handle. Cannot be NULL.
    - unsigned char cfgSelector    - current/power-up chip setting selection. Valid values are:
                                     - M_MCP2210_VM_CONFIG: designates current chip setting (Volatile
                                       Memory)
                                     - M_MCP2210_NVRAM_CONFIG: designates power-up chip setting (NVRAM)
Outputs:
    - unsigned int %pbaudRate      - transfer speed.
    - unsigned int %pidleCsVal     - IDLE chip select value.
                                      bit31  -  -  -  -  bit8                                    bit0
                                         x  x  ...  x  x  CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0
    - unsigned int %pactiveCsVal   - Active Chip Select Value.
                                     bit31  -  -  -  -  bit8                                     bit0
                                         x  x  ...  x  x  CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0
    - unsigned int %pCsToDataDly   - Chip Select to Data Delay (quanta of 100 us)
                                       - 16-bit value => max value is (2^16-1).
    - unsigned int %pdataToCsDly   - Last Data Byte to CS (de-asserted) Delay (quanta of 100 us)
                                       - 16-bit value => max value is (2^16)-1.
    - unsigned int %pdataToDataDly - Delay Between Subsequent Data Bytes (quanta of 100 us)
                                       - 16-bit value => max value is (2^16)-1.
    - unsigned int %ptxferSize     - Bytes to Transfer per SPI Transaction.
                                       - 16-bit value => max value is (2^16)-1.
    - unsigned char %pspiMd        - SPI Mode. Valid values are:
                                       - M_MCP2210_SPI_MODE0
                                       - M_MCP2210_SPI_MODE1
                                       - M_MCP2210_SPI_MODE2
                                       - M_MCP2210_SPI_MODE3
Returns:
    - 0 for success:         M_E_SUCCESS
    - negative error code:   M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                             M_E_ERR_UNKOWN_ERROR, M_E_ERR_INVALID_PARAMETER
                             M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO
                             M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_SetSpiConfig

```
int M_Mcp2210_SetSpiConfig(IntPtr handle, unsigned char cfgSelector, unsigned int %pbaudRate,
                            unsigned int %pidleCsVal, unsigned int %pactiveCsVal,
                            unsigned int %pCsToDataDly, unsigned int %pdataToCsDly,
                            unsigned int %pdataToDataDly, unsigned int %ptxferSize,
                            unsigned char %pspiMd)
====================================================================================================
Description: Set the SPI transfer settings for current (VM) configuration or for power-up
default (NVRAM) SPI configuration.
If SPI transfer is in progress, the configuration change is not accepted and the API returns
M_E_ERR_SPI_CFG_ABORT error.
In the case of the current (VM) SPI configuration change, the API call returns the actual configuration
values that will be used by the device, derived from the user supplied ones but adapted to the
device capabilities.
If the NVRAM is password protected, the power-up SPI configuration cannot be changed and
the API returns M_E_ERR_BLOCKED_ACCESS error.
Parameters:
Inputs:
    - IntPtr handle                - The device handle. Cannot be NULL.
    - unsigned char cfgSelector    - current/power-up chip setting selection. Valid values are:
                                     - M_MCP2210_VM_CONFIG: designates current chip setting (Volatile
                                       Memory)
                                     - M_MCP2210_NVRAM_CONFIG: designates power-up chip setting (NVRAM)
Inputs / Outputs:
    - unsigned int %pbaudRate      - transfer speed.
    - unsigned int %pidleCsVal     - IDLE chip select value.
                                    bit31  -  -  -  -  bit8                                    bit0
                                       x  x  ...  x  x  CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0
    - unsigned int %pactiveCsVal   - Active Chip Select Value.
                                   bit31  -  -  -  -  bit8                                     bit0
                                       x  x  ...  x  x  CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0
    - unsigned int %pCsToDataDly   - Chip Select to Data Delay (quanta of 100 us)
                                     - 16-bit value => max value is (2^16-1).
    - unsigned int %pdataToCsDly   - Last Data Byte to CS (de-asserted) Delay (quanta of 100 us)
                                     - 16-bit value => max value is (2^16)-1.
    - unsigned int %pdataToDataDly - Delay Between Subsequent Data Bytes (quanta of 100 us)
                                     - 16-bit value => max value is (2^16)-1.
    - unsigned int %ptxferSize     - Bytes to Transfer per SPI Transaction.
                                     - 16-bit value => max value is (2^16)-1.
    - unsigned char %pspiMd        - SPI Mode. Valid values are:
                                     - M_MCP2210_SPI_MODE0
                                     - M_MCP2210_SPI_MODE1
                                     - M_MCP2210_SPI_MODE2
                                     - M_MCP2210_SPI_MODE3
Returns:
    - 0 for success:         M_E_SUCCESS
    - negative error code:   M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                             M_E_ERR_UNKOWN_ERROR, M_E_ERR_INVALID_PARAMETER
                             M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO
                             M_E_ERR_BLOCKED_ACCESS, M_E_ERR_SPI_CFG_ABORT
```

M_Mcp2210_xferSpiData

```
int M_Mcp2210_xferSpiData(IntPtr handle, array<Byte> ^pdataTx, array<Byte> ^pdataRx,
                          unsigned int %pbaudRate, unsigned int %ptxferSize, unsigned int csmask)
==================================================================================================
Description: Data transfer over SPI bus. This API assumes that GPIO configuration and SPI
communication settings are already configured, either as power-up default settings or through the use
of M_Mcp2210_SetGpioConfig() and M_Mcp2210_SetSPIConfig(). In addition, this API offers the user the
option to change the current (VM) SPI transfer length and/or to indicate which GPIO pin(s) to be used
as CS pin(s). Please note that these changes are not reverted at the end of the SPI data transfer but
remain in effect until a new GPIO/SPI configuration change is initiated by the user.

If SPI configuration change is not accepted (because SPI transfer is in progress) the API returns
M_E_ERR_SPI_CFG_ABORT error.
In the case of the current (VM) SPI configuration change, the API call returns the actual configuration
values that will be used by the device, derived from the user supplied ones but adapted to the
device capabilities.
This API can also reverse the effect of the GP8 firmware error: when GP8 is set as GPIO "out",
the SPI transfer command is changing it to GPIO "in". The user has the option to skip this fix if the
application execution speed gain is relevant.

Parameters:
Inputs:
    - IntPtr handle                - The pointer to the device handle. Cannot be NULL.
    - array<Byte> ^pdataTx         - data buffer to be transmitted. Cannot be NULL.
                                     The user must ensure that buffer size matches the SPI transfer
                                     size setting. If ptxferSize parameter indicates a transfer size
                                     change and the array Length property is smaller than ptxferSize,
                                     negative error code is returned - M_E_ERR_INVALID_PARAMETER
    - unsigned int %pbaudRate      - SPI transfer speed.
                                     - If the value is different than the current speed,
                                     the API attempts to change it as requested.
                                     - If the value is 0, the current SPI transfer speed is preserved.
    - unsigned int %ptxferSize     - Bytes to transfer per SPI transaction.
                                       - 16-bit value => max valid value is (2^16)-1.
                                       - A larger values than (2^16)-1 indicates that
                                         the current SPI transfer size setting must remain unchanged.
                                       - If set to 0, all the configuration changes are accepted if
                                         valid (baudRate, CS/GPIO configuration, transfer size set 0),
                                         but no SPI transfer is performed.
    - unsigned int csmask          - bit mask indicating which GPIO pins must be active chip select
                                       (CS) pins for the SPI transfer. It is assumed that CS settings
                                       (idle/active values and delays) are properly configured
                                       already either as power-up defaults or with prior use
                                       of the Mcp2210_SetSPIConfig() API.
                                      bit31  -  ...  -  -  bit8                                bit0
                                      GP8CE  x  ...  x  x   CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0
                                       - A bit value of '1' configures the GPIO pins as CS
                                         (alternate function 1) during the SPI data transfer.
                                         A bit value of 0 configures the pins as GPIO (standard
                                         function) except the pins which are configuration as
                                         dedicated function (alternate function 2).
                                         The GPIO direction and default values must be already
                                         configured as needed either as power-up defaults or with prior
                                         use of Mcp2210_SetGpioConfig().
                                       - If all CS[0..8] bit values are '0', current GPIO configuration
                                         is preserved.
                                       - If csmask[31] bit (GP8CE) is set ('1'), the GP8 errata is
                                         ignored, assuming that user application is not affected by the
                                         GP8 pin being reconfigured as GPIO "input" after the SPI
                                         transfer command.
```

```
Outputs:
    - array<Byte> ^pdataRx          - receive data buffer. Cannot be NULL.
                                      The user must ensure that buffer size matches the SPI transfer
                                      size setting. If ptxferSize parameter indicates a transfer size
                                      change and the array Length property is smaller than ptxferSize,
                                      negative error code is returned - M_E_ERR_INVALID_PARAMETER
    - unsigned int %pbaudRate       - If the function call returns M_E_SUCCESS, this parameter contains
                                      the current transfer speed value accepted by the device.
                                      The value may differ from the user supplied one - it may be
                                      derived from the user supplied value but adapted to the device
                                      capabilities.
    - unsigned int %ptxferSize      - If the function call returns M_E_SUCCESS, this parameter contains
                                      the current transfer size used by the device.
Returns:
    - 0 for success:        M_E_SUCCESS
    - negative error code:  M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                            M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO
                            M_E_ERR_UNKOWN_ERROR, M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
                            M_E_ERR_SPI_TIMEOUT, M_E_ERR_SPI_EXTERN_MASTER, M_E_ERR_SPI_RX_INCOMPLETE
                            M_E_ERR_SPI_CFG_ABORT
```

M_Mcp2210_xferSpiDataEx

```
int M_Mcp2210_xferSpiDataEx(IntPtr handle, array<Byte> ^pdataTx, array<Byte> ^pdataRx,
                            unsigned int %pbaudRate, unsigned int %ptxferSize, unsigned int csmask,
                            unsigned int %pidleCsVal, unsigned int %pactiveCsVal,
                            unsigned int %pCsToDataDly, unsigned int %pdataToCsDly,
                            unsigned int %pdataToDataDly, unsigned char %pspiMd)
====================================================================================================
Description: This function configures the SPI transfer settings (VM parameters only) and initiate the
SPI data transfer. Basically, this API combines the M_Mcp2210_SetSPIConfig() and M_Mcp2210_xferSpiData()
function calls. Also, this function allows the user to specify which GPIO pins to be configured as
SPI chip select, but it is assumed that the other GPIO pins configuration is already done, either as
power-up default settings or through the use of M_Mcp2210_SetGpioConfig() or the other GPIO related API.
If the function call succeeds (returns M_E_SUCCESS), the parameters listed below as "Inputs/outputs"
contain the configuration values used by device. In some cases, the values are derived from the user
supplied ones, being adapted to the device capabilities (e.g. the baudRate).
Please note that these changes are not reverted at the end of the SPI data transfer but
remain in effect until a new GPIO/SPI configuration change is initiated by the user.
Mind that, if there is already an SPI transfer in progress, the configuration change is not accepted
and the API returns M_E_ERR_SPI_CFG_ABORT error.
This API can also reverse the effect of the GP8 firmware error: when GP8 is set as GPIO "out",
the SPI transfer command is changing it to GPIO "in". The user has the option to skip this fix if the
application execution speed gain is relevant.

Parameters:
Inputs:
    - IntPtr handle                - The device handle. Cannot be NULL.

    - array<Byte> ^pdataTx         - data buffer to be transmitted. Cannot be NULL.
                                     The user must ensure that buffer size matches the SPI transfer
                                     size setting. If the array Length property is smaller
                                     than ptxferSize, negative error code is returned
                                     - M_E_ERR_INVALID_PARAMETER

    - unsigned int csmask          - bit mask indicating which GPIO pins must be active chip select
                                     (CS) pins for the SPI transfer.
                                     bit31  -  ...  -  -  bit8                                    bit0
                                     GP8CE  x  ...  x  x   CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0
                                     - A bit value of '1' configures the GPIO pins as CS
                                       (alternate function 1) during the SPI data transfer.
                                       A bit value of 0 configures the pins as GPIO (standard function)
                                       except the pins which are configuration as
                                       dedicated function (alternate function 2).
                                       The GPIO direction and default values must be already configured
                                       as needed either as power-up defaults or with prior use of
                                       Mcp2210_SetGpioConfig().
                                     - If all CS[0..8] bit values are '0', current GPIO configuration
                                       is preserved.
                                     - If csmask[31] bit (GP8CE) is set ('1'), the GP8 errata is
                                       ignored, assuming that user application is not affected by the
                                       GP8 pin being reconfigured as GPIO "input" after the SPI
                                       transfer command.
Output:
    - array<Byte> ^pdataRx         - receive data buffer. Cannot be NULL.
                                     The user must ensure that buffer size matches the SPI transfer
                                     size setting. If the array Length property is smaller
                                     than ptxferSize, negative error code is returned
                                     - M_E_ERR_INVALID_PARAMETER

Inputs / Outputs:
    - unsigned int %pbaudRate      - transfer speed.
    - unsigned int %ptxferSize      - Bytes to Transfer per SPI Transaction.
                                      - 16-bit value => max value is (2^16)-1.

    - unsigned int %pidleCsVal      - IDLE chip select value.
                                     bit31  -  -  -  -  bit8                                    bit0
                                        x  x  ...  x  x  CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0

    - unsigned int %pactiveCsVal    - Active Chip Select Value.
                                    bit31  -  -  -  -  bit8                                     bit0
                                        x  x  ...  x  x  CS8  CS7  CS6  CS5  CS4  CS3  CS2  CS1  CS0

    - unsigned int %pCsToDataDly    - Chip Select to Data Delay (quanta of 100 us)
                                      - 16-bit value => max value is (2^16-1).

    - unsigned int %pdataToCsDly    - Last Data Byte to CS (de-asserted) Delay (quanta of 100 us)
                                      - 16-bit value => max value is (2^16)-1.

    - unsigned int %pdataToDataDly  - Delay Between Subsequent Data Bytes (quanta of 100 us)
                                      - 16-bit value => max value is (2^16)-1.

    - unsigned char %pspiMd         - SPI Mode.
                                    Valid values are:
                                    - M_MCP2210_SPI_MODE0
                                    - M_MCP2210_SPI_MODE1
                                    - M_MCP2210_SPI_MODE2
                                    - M_MCP2210_SPI_MODE3

Returns:
    - 0 for success:        M_E_SUCCESS
    - negative error code:  M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE, M_E_ERR_INVALID_PARAMETER
                            M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                            M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
                            M_E_ERR_SPI_CFG_ABORT, M_E_ERR_SPI_TIMEOUT, M_E_ERR_SPI_EXTERN_MASTER,
                            M_E_ERR_SPI_RX_INCOMPLETE
```

M_Mcp2210_CancelSpiTxfer

```
int M_Mcp2210_CancelSpiTxfer(IntPtr handle, unsigned char %pspiExtReqStat, unsigned char %pspiOwner)
====================================================================================================
Description: Cancel the current SPI transfer and provides information about:
- SPI Bus release external request status,
- SPI Bus ownership prior the command (who is the Master).
The command also releases the bus ownership.

Parameters:
Inputs:
    - IntPtr handle                 - The device handle. Cannot be NULL.

Outputs:
    - unsigned char %pspiExtReqStat - SPI Bus release external request status:
                                        - 0x01 - No External Request for SPI Bus Release
                                        - 0x00 - Pending External Request for SPI Bus Release
    - unsigned char %pspiOwner      - SPI Bus current owner:
                                        - 0x00 - No owner
                                        - 0x01 - USB Bridge
                                        - 0x02 - External Master.
Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_RequestSpiBusRel

```
int M_Mcp2210_RequestSpiBusRel(IntPtr handle, unsigned char ackPinVal)
====================================================================================================
Description: Request the release of the SPI bus and set the value of the "SPI Bus Release ACK" pin
if GP7 is assigned to this dedicated function and if the bus can be released.
If SPI transfer is in progress, the bus is not released, the command returns M_E_ERR_SPI_XFER_ONGOING
errorcode and the host application should either wait the transfer completion or cancel it
(by using M_Mcp2210_CancelSpiTxfer) and request again the SPI bus release.

Parameters:
Inputs:
    - IntPtr handle                  - The device handle. Cannot be NULL.
    - unsigned char ackPinVal        - the value of the "SPI Bus Release ACK" pin if the
                                       command succeeds and the GP7 is assigned to alternate function 2.
Outputs:
    - none

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE, M_E_ERR_INVALID_PARAMETER
                           M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
                           M_E_ERR_SPI_XFER_ONGOING
```

M_Mcp2210_GetSpiStatus

```
int M_Mcp2210_GetSpiStatus(IntPtr handle, unsigned char %pspiExtReqStat,
                           unsigned char %pspiOwner, unsigned char %pspiTxferStat)
====================================================================================================
Description: If there is no device communication error, this command returns 0 (M_E_SUCCESS) and
updates the parameters with information about SPI transfer status. In case of error, the function
returns negative error code.

Parameters:
Inputs:
    - IntPtr handle                  - The USB device handle. Cannot be NULL.

Outputs:
    - unsigned char %pspiExtReqStat  - SPI Bus release external request status:
                                        - 0x01 - No External Request for SPI Bus Release
                                        - 0x00 - Pending External Request for SPI Bus Release
    - unsigned char %pspiOwner       - SPI Bus current owner:
                                        - 0x00 - No owner
                                        - 0x01 - USB Bridge
                                        - 0x02 - External Master.
    - unsigned char %pspiTxferStat   - SPI transfer status:
                                        - 0x00 - No SPI transfer in progress
                                        - 0x01 - SPI transfer is in progress
Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_ReadEEProm

```
int M_Mcp2210_ReadEEProm(IntPtr handle, unsigned char address, unsigned char %pcontent)
====================================================================================================
Description: Read the byte at the EEPROM memory address provided as parameter.
The function returns negative error code in case of failure.
The EEPROM size is 256 bytes.

Parameters:
Inputs:
    - IntPtr handle                - The USB device handle. Cannot be NULL.
    - unsigned char address        - the EEPROM location to be read.
Outputs:
    - unsigned char %pcontent      - the value stored at the EEPROM address, if the function returns
                                     M_E_SUCCESS. In case of failure, pcontent is not changed.
Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_WriteEEProm

```
int M_Mcp2210_WriteEEProm(IntPtr handle, unsigned char address, unsigned char content)
====================================================================================================
Description: Write one byte at the EEPROM memory address provided as parameter.
The EEPROM size is 256 bytes.
The function returns negative error code in case of failure.
If the error code is "M_E_ERR_BLOCKED_ACCESS" the write failure is due to the device being either
password protected or permanently locked. M_Mcp2210_GetAccessStatus() API provides the details about
device protection status. If the device is not permanently locked, the password protection can be
unlocked or disabled by using the access control API.

Parameters:
Inputs:
    - IntPtr handle             - The USB device handle. Cannot be NULL.
    - unsigned char address     - the EEPROM location to write to.
    - unsigned char content     - the value to store at the EEPROM address.

Outputs:
    - none

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
                           M_E_ERR_BLOCKED_ACCESS
```

M_Mcp2210_GetAccessCtrlStatus

```
int M_Mcp2210_GetAccessCtrlStatus(IntPtr handle, unsigned char %pAccessCtrl,
                                  unsigned char %pPasswdAttemptCnt, unsigned char %pPasswdAccepted)
====================================================================================================
Description: This command provides information about the NVARM access control status.
In case of successful execution, the function returns 0 (M_E_SUCCESS) and updates the pointer
parameters with the status codes. In case of failure, negative error code is returned.

Parameters:
Inputs:
    - IntPtr handle                     - The USB device handle. Cannot be NULL.

Outputs:
    - unsigned char %pAccessCtrl        - NVRAM chip settings access control:
                                            - 0x00 - settings not protected
                                            - 0x40 - settings protected by Password
                                            - 0x80 - settings permanently locked

    - unsigned char %pPasswdAttemptCnt  - counts how many times the NVRAM password was tried.
                                          The value will saturate to 5 even if the number of wrong
                                          attempts exceeds 5. After the password is accepted, the
                                          counter is reset to 0.

    - unsigned char %pPasswdAccepted    - If the NVRAM access is password protected
                                          (indicated by pAccessCtrl == 0x40), this flag indicates
                                          whether the password has been accepted or not after using the
                                          M_Mcp2210_EnterAccessPasswd() command:
                                            - 0x00 - Password NOT accepted
                                            - 0x01 - Password accepted.
                                        - If the NVRAM access is not password protected
                                          (indicated by pAccessCtrl == 0x00), this flag returns 0x00.

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_EnterPassword

```
int M_Mcp2210_EnterPassword(IntPtr handle, String ^passwd)
====================================================================================================
Description: Sends the NVRAM unlock password to the device.
The function returns 0 (M_E_SUCCESS) either if the password is accepted or if the device is not
password protected. In both cases, the NVRAM changes are permitted.
The function returns negative error code in case of device communication failure or if the password
is not accepted by the device.
Rejected password error codes:
    - M_E_ERR_NVRAM_LOCKED        - NVRAM is permanently locked, no password is accepted
    - M_E_ERR_WRONG_PASSWD        - password mismatch, but number of attempts is less than 5
    - M_E_ERR_ACCESS_DENIED       - password mismatch, but the number of attempts exceeded 5,
                                    so the NVRAM writes are blocked until the next device reset

Parameters:
Inputs:
    - IntPtr handle         - The USB device handle. Cannot be NULL.
    - String ^passwd        - string of 8 characters. Cannot be NULL.
                            !!! Important !!!
                            The password must have exactly 8 characters. If there is a different
                            number of characters, -2 (M_E_ERR_INVALID_PARAMETER) error code is returned.
                            The String UNICODE characters are converted to ANSI string before being sent
                            to the device for verification.
Outputs:
    - none

Returns:
    - 0 for success:        M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE, M_E_ERR_INVALID_PARAMETER
                            M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                            M_E_ERR_NVRAM_LOCKED, M_E_ERR_WRONG_PASSWD, M_E_ERR_ACCESS_DENIED
                            M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_SetAccessControl

```
int M_Mcp2210_SetAccessControl(IntPtr handle, unsigned char accessConfig,
                               String ^currentPasswd, String ^newPasswd)
====================================================================================================
Description: Configures the NVRAM password protection.
The function returns 0 (M_E_SUCCESS) if the new access control configuration is accepted
or negative error code in case of device communication failure or if the access control configuration
change is not accepted by the device.
Error codes in case of rejected configuration change:
    - M_E_ERR_NVRAM_LOCKED        - NVRAM is permanently locked, no access control change is accepted
    - M_E_ERR_WRONG_PASSWD        - old password mismatch, but the number of attempts is less than 5
    - M_E_ERR_ACCESS_DENIED       - old password mismatch, but the number of attempts exceeded 5,
                                    so the NVRAM writes are blocked until the next device reset

!!! Important notes !!!
1. The password strings must have exactly 8 characters.
If they have a different number of characters, -2 (M_E_ERR_INVALID_PARAMETER) error code is returned.

2. The String UNICODE characters are converted to ANSI string before being send to the device.

3. The new password string becomes effective after the next chip reset or power cycle. Therefore,
if maximum protection is required, a chip reset is recommended after calling
M_Mcp2210_SetAccessControl()
with accessConfig parameter set to M_MCP2210_NVRAM_PROTECTED or M_MCP2210_NVRAM_PASSWD_CHANGE.
The chip reset can be achieved using M_Mcp2210_Reset() API.

Parameters:
Inputs:
    - IntPtr handle        - The USB device handle. Cannot be NULL.
    - unsigned char accessConfig
                           - indicates which type of access control change is performed by the API call.
                                - 0x0 (M_MCP2210_NVRAM_NO_PROTECTION) - disable the password protection:
                                  If the password protection is enabled, the currentPassword must be
                                  valid, otherwise the function returns negative error code:
                                  M_E_ERR_WRONG_PASSWD or M_E_ERR_ACCESS_DENIED
                                  If the password protection is already disabled, the function
                                  ignores the currentPassword and returns M_E_SUCCESS.

                                - 0x40 (M_MCP2210_NVRAM_PROTECTED) - enable the password protection:
                                  If the password protection is not enabled, newPassword must be a valid
                                  password string.
                                  If the password protection is already enabled, newPassword is ignored
                                  and the function returns negative error code: M_E_ERR_NVRAM_PROTECTED.

                                - 0xA5 (M_MCP2210_NVRAM_PASSWD_CHANGE) - change the current password
                                  with the new one if the password protection is already enabled.
                                  Both currentPasswd and newPasswd must be valid password strings.
                                  If the password protection is not enabled, the function returns
                                  negative error code: M_E_ERR_PASSWD_CHANGE

    - String ^currentPasswd  - string of 8 characters

    - String ^newPasswd      - string of 8 characters

Outputs:
    - none

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE, M_E_ERR_INVALID_PARAMETER
```

```
                            M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                            M_E_ERR_NVRAM_LOCKED, M_E_ERR_WRONG_PASSWD, M_E_ERR_ACCESS_DENIED
                            M_E_ERR_NVRAM_PROTECTED, M_E_ERR_PASSWD_CHANGE
                            M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```

M_Mcp2210_SetPermanentLock

```
int M_Mcp2210_SetPermanentLock(IntPtr handle)
====================================================================================================
Description: Permanently locks the device NVRAM settings.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!    WARNING    !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
USE THIS FUNCTION WITH GREAT CAUTION. THERE IS NO WAY TO REMOVE THE PERMANENT LOCK
THE CHIP POWER-UP DEFAULT SETTINGS CANNOT BE CHANGED AFTER THIS FUNCTION CALL
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

The function returns 0 (M_E_SUCCESS) if the lock configuration is accepted
or negative error code in case of device communication failure or if the access control configuration
change is not accepted by the device.

Error codes in case of rejected configuration change:
    - M_E_ERR_NVRAM_LOCKED      - NVRAM is already permanently locked
    - M_E_ERR_BLOCKED_ACCESS    - NVRAM is password protected. NVRAM change protection must be
                                  deactivated first, using M_Mcp2210_EnterPassword() or
                                  M_Mcp2210_SetAccessControl()

Parameters:
Inputs:
    - IntPtr handle             - The USB device handle. Cannot be NULL.

Outputs:
    - none

Returns:
    - 0 for success:       M_E_SUCCESS
    - negative error code: M_E_ERR_NULL, M_E_ERR_INVALID_HANDLE_VALUE, M_E_ERR_INVALID_PARAMETER
                           M_E_ERR_HID_TIMEOUT, M_E_ERR_HID_RW_FILEIO, M_E_ERR_UNKOWN_ERROR
                           M_E_ERR_NVRAM_LOCKED, M_E_ERR_BLOCKED_ACCESS
                           M_E_ERR_CMD_ECHO, M_E_ERR_CMD_FAILED
```