



USB2530API User Manual

Table of Contents

1	Introduction.....	4
2	Legal Information	4
3	API	5
3.1	Device Open / Close APIs.....	5
3.1.1	MchpUsbOpen	5
3.1.2	MchpUsbGetDevice.....	6
3.1.3	MchpUsbOpenExt.....	7
3.1.4	MchpUsbClose.....	8
3.2	Flexconnect API.....	9
3.2.1	MchpUsbFlexConnect	9
3.3	GPIO Bridging APIs	11
3.3.1	MchpUsbGpioConfigurePulldown	12
3.3.2	MchpUsbGpioConfigurePullUp	13
3.3.3	MchpUsbGpioGet	14
3.3.4	MchpUsbGpioSet.....	15
3.4	XDATA Bridging APIs.....	16
3.4.1	MchpUsbXdataRead	16
3.4.2	MchpUsbXdataWrite	17
3.5	I2C Bridging APIs.....	18
3.5.1	MchpUsbI2CSetConfig.....	18
3.5.2	MchpUsbI2CGetConfig.....	19
3.5.3	MchpUsbI2CRead.....	20
3.5.4	MchpUsbI2CWrite.....	22
3.6	OTP Bridging APIs	24
3.6.1	MchpUsbOtpRead.....	24
3.6.2	MchpUsbOtpWrite.....	25
3.7	RAM Bridging APIs	26
3.7.1	MchpUsbRamRead	26
3.7.2	MchpUsbRamWrite	26
3.8	SPI Bridging APIs.....	26
3.8.1	MchpUsbSpiSetConfig	26
3.8.2	MchpUsbSpiWrite	27
3.8.3	MchpUsbSpiRead	28
3.9	UART Bridging APIs.....	30
3.9.1	MchpUsbUartSetConfig	30
3.9.2	MchpUsbUartWrite.....	31
3.9.3	MchpUsbUartRead	32
4	APPENDIX A.....	33
4.1	Classes & Structs.....	33
4.1.1	struct _DEVICE_INTERFACE_	33
4.2	Typedefs & Macros	33
4.2.1	Macros.....	33
4.2.2	Typedefs.....	33

4.2.3	MCHP_USB_ERROR	33
-------	----------------------	----

1 Introduction

Originally intended as a replacement for serial and parallel connections on a PC to connect mice, keyboards and printers, Universal Serial Bus has grown to become a common interface in many embedded industrial, medical, automotive, and consumer applications. Microchip's family of versatile, cost-effective, and power-efficient USB Hub delivers industry-leading data throughput in mixed-speed USB environments. The USB hub families provide a USB port expansion solution for USB 2.0 and USB 3.0 applications that demand ultra low power and a small footprint without compromising on performance.

Over the standard USB hub functionality, Microchip hubs support a wide array of functionalities like GPIO bridging, UART bridging, I²C bridging, etc. With its high level APIs, USB2530 API enables end customers to realize the full potential of Microchip USB hubs.

2 Legal Information

Software License Agreement

(c) 2004 - 2014 Microchip Technology Inc.

Microchip licenses this software to you solely for use with Microchip products. The software is owned by Microchip and its

licensors, and is protected under applicable copyright laws. All rights reserved.

SOFTWARE IS PROVIDED "AS IS" MICROCHIP EXPRESSLY DISCLAIMS ANY WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.

To the fullest extent allowed by law, Microchip and its licensors liability shall not exceed the amount of fees, if any, that you have paid directly to Microchip to use this software.

MICROCHIP PROVIDES THIS SOFTWARE CONDITIONALLY UPON YOUR ACCEPTANCE OF THESE TERMS.

Trademark Information

The Microchip name and logo, the Microchip logo, MPLAB, and PIC are registered trademarks of Microchip Technology

Incorporated in the U.S.A. and other countries.

PICDEM and PICtail are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

Microsoft, Windows, Windows Vista, and Authenticode are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

SD is a trademark of the SD Association in the U.S.A and other countries

3 API

Usage of these APIs requires that the WinUSB device driver is installed for the USB2530 internal USB device (5th port) for the USB2534 family of hub being used. For details on installing the WinUSB driver, please refer to [MchpUSB2534API Release notes.docx](#) Section 7.

3.1 Device Open / Close APIs

This section covers APIs which enable open / close of the handle to the device of interest. Before issuing any command to the device, the handle needs to be opened first.

3.1.1 MchpUsbOpen

Prototype:

HANDLE MchpUsbOpen (UINT16 VendorID, UINT16 ProductID)

Devices Supported: USB2534 Family

Description:

This API will return handle to the USB device with the specified USB Vendor ID and Product ID. If there is more than one USB device with the specified USB Vendor ID & product ID, handle to the first of such USB device will be returned.

Parameters:

↓↑	Parameter	Description
IN	VendorID	Vendor ID of the USB device of interest
IN	ProductID	Product ID of the USB device of interest

Returns:

A valid device Handle, which needs to be passed on to other APIs for interaction with the device

NULL if no device with the specified vendor ID & product ID is present in the system.

Sample:

```
HANDLE hDevice;  
  
hDevice = MchpUsbOpen (0x0424, 0x2530);  
  
if (hDevice != NULL)  
{
```

```
        // Interact further with the device
        ...
    }
```

3.1.2 MchpUsbGetDevice

Prototype:

MCHP_USB_ERROR MchpUsbGetDevice (UINT16 VendorID, UINT16 ProductID, OUT DEVIO * DevicePaths)

Devices Supported: USB2534 Family

Description:

This API will return a list of device paths for those USB devices specified in the USB Vendor ID and Product ID parameters.

Note: A DevicePath list will be created based on the number of USB devices present in the system with the specified VendorID & ProductID. Please refer DEVIO structure for details of the entries.

The caller is responsible to free the memory for the list of device paths allocated in this function. The memory is allocated using

Parameters:

↓↑	Parameter	Description
IN	VendorID	Vendor ID of the USB device of interest
IN	ProductID	Product ID of the USB device of interest
OUT	DevicePaths	List of device paths with the specified VendorID & ProductID dNoOfDevices member will contain the number of entries in the list DevicePath member contains the null terminated wide char list of device symbolic names. This member is internally used by the API library & needs to be treated as Reserved from the caller perspective

Returns:

Mchp_Error_Success – if the operation succeeded. The caller is responsible to call GlobalFree (DEVIO->DevicePath) after all interactions with the device is completed.

Errorcode as defined in MCHP_USB_ERROR or Windows System call error code - for failure.

Sample:

```

DEVIO stDEVIO;

// get the list of device paths for VID = 0x0424 & PID = 0x2530
MCHP_USB_ERROR result = MchpUsbGetDevice(0x0424, 0x2530, &stDEVIO);

if(result == MCHP_Error_Success){
    // stDEVIO.dNoOfDevices contains the number of list entries
    // ...

    // To free the memory, use the GlobalFree function.
    // dNoOfDevices contain the number of devices in the
    // list containing the symbolic names of the devices.

    unsigned int num_of_devices = stDEVIO.dNoOfDevices;
    for (int i=0; i < num_of_devices; i++)
    {
        // Free the individual WCHAR* entries in the list
        GlobalFree(stDEVIO.DevicePath[i]);
        stDEVIO.DevicePath[i] = NULL;
    }

    // Free the pointer to the list of WCHAR* entries
    GlobalFree(stDEVIO.DevicePath);
    stDEVIO.DevicePath = NULL;
}

```

3.1.3 MchpUsbOpenExt**Prototype:**

HANDLE MchpUsbOpenExt (INT DevPathIndex, UINT32 Timeout, DEVIO* DevicePaths)

Devices Supported: USB2534 Family

Description:

This API will return a handle to the device, specified at DevPathIndex, within the device path list of DevicePaths structure.

Parameters:

↓↑	Parameter	Description
IN	DevPathIndex	The index within the device path list in the DevicePaths structure
IN	Timeout	Reserved for Future use
IN	DevicePaths	List of device paths obtained by calling MchpUsbGetDevice function.

Returns:

On success returns a handle to the device pointed in the list

On failure the function returns a NULL.

Sample:

```
DEVID stDEVID;

// get the list of device paths for VID = 0x0424 & PID = 0x2530
MCHP_USB_ERROR result = MchpUsbGetDevice(0x0424, 0x2530, &stDEVID);

if (result == MCHP_Error_Success){
    //stDEVID is already populated from above call
    HANDLE hnd = MchpUsbOpenExt(0x02, 0, &stDEVID);
    If (NULL != hnd) {
        // Handle for the 2nd device entry in the list received
    }
}
```

3.1.4 MchpUsbClose

Prototype:

MCHP_USB_ERROR MchpUsbClose (HANDLE DevID)

Devices Supported: USB2534 Family

Description:

This API will close the handle for the device specified by DevID.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device which needs to be closed

Returns:

Mchp_Error_Success – if the operation succeeded

Errorcode as defined in MCHP_USB_ERROR or Windows System call error code - for failure.

The function returns MCHP_Error_Invalid_Device_Handle if the device handle did not match an entry of the open devices list maintained by the library.

Sample:


```
HANDLE hDevice;

hDevice = MchpUsbOpen (0x0424, 0x2530);

if (hDevice != INVALID_HANDLE_VALUE)
{
    // Interact further with the device
    // ...
    // Done with the device, close handle
    MCHP_USB_ERROR result = MchpUsbClose(hDevice);
    if(result == MCHP_Error_Success){
        // Close handle was successful
    }
}
```

3.2 Flexconnect API

Flexconnect refers to the feature in Microchip USB hubs, wherein the upstream port swaps its role with downstream port 1 and also vice versa at run time.

3.2.1 MchpUsbFlexConnect

Prototype:

MCHP_USB_ERROR MchpUsbFlexConnect (HANDLE DevID, UINT16 Config)

Devices Supported: USB2534 Family

Description:

This API will issue Flex Connect command to the device.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device to which flexconnect command will be issued
IN	Config	Passed as is in wValue field of the Flexconnect SETUP command. See below for the detailed description of different bits

Config bit field encoding:

Bit	7	6	5	4	3	2	1	0
Field	CHNG PIN FUNCT	FLEX_ STATE	DIS_P5	DIS_P4	DIS_P3	DIS_P2	DIS_P1	RFU
Bit	15	14	13	12	11	10	9	8
Field	Must be	DCP_E	VBUS_			HDD	HDD	HDD

	set to 1	N	RESET			TMR[2]	TMR[1]	TMR[0]
--	----------	---	-------	--	--	--------	--------	--------

DCP_EN :

1 – All disabled ports, while in Flexconnect mode, will output the Universal DCP handshake on DP and DM according to the Battery Charging configuration registers.

0 – In all disabled ports, the DP/DM lines will revert to the default disabled state, regardless of Flexconnect mode.

DIS_P1 - Disables downstream port 1 during the next enumeration

DIS_P2 - Disables downstream port 2 during the next enumeration

DIS_P3 - Disables downstream port 3 during the next enumeration

DIS_P4 - Disables downstream port 4 during the next enumeration

DIS_P5 - Disables downstream port 5 during the next enumeration. If port5 is disabled (UDC), then no more commands can be issued to the UDC until it is re-enabled via VSM.

Port disable/ enable feature assumes physical port only. The port remap register will be derived from the ROM defaults (will not consider modifications made in OTP/ SMBus)

FLEX_STATE

The required Flexconnect state of the hub upon the next attach.

1 – Turn on Flexconnect

0 – Turn off Flexconnect

CHNG PIN FUNCT This bit is valid only if FLEX_STATE is 1.

Case 1: CHNG PIN FUNCT = 0 -> EN_FLEX_MODE = 1

1. OCS1_N = Initiate Hub Enumeration, Low = Hub enters a low power state. High = Hub attempts to enumerate with upstream port (PORT1)
2. PRTCTL1 = Pulled high and has over current capabilities.
3. VBUS_DET = Don't care
4. SUSPEND = Downstream port (Port 0) port power control and OCS detection.

Case 2: CHNG PIN FUNCT = 1 -> EN_FLEX_MODE = 0

1. OCS1_N = Don't care (I am not sure this is the case if EN_FLEX_MODE = 0)
2. PRTCTL1 = Downstream port (Port 0) port power control and OCS detection
3. VBUS_DET = Initiate Hub Enumeration, Low = Hub enters a low power state. High = Hub attempts to enumerate with upstream port (PORT1)
4. SUSPEND/ PIO0 = Don't care.

HDD TMR – The Host Disconnect Detect Timer, this is the time the firmware waits at most for the host to arrive after entering FLEX state, after which un-flex automatically.

000 = No auto unflex on timeout

001 = 10ms

010 = 100ms
011 = 500ms
100 = 1s
101 = 5s
110 = 10s
111 = 20s

Returns:

Mchp_Error_Success – if the operation succeeded

Errorcode as defined in MCHP_USB_ERROR or Windows System call error code - for failure.

Sample:

```
HANDLE hDevice;

hDevice = MchpUsbOpen (0x0424, 0x2530);

if (hDevice != INVALID_HANDLE_VALUE)
{
    /*
     * Prepare the bit field for flexconnect configuration
     * Bit 15 to be set to 1
     * Bit 6 is set for enabling FLEX_STATE
     * This denotes the required Flexconnect state of the
     * hub upon the next attach.
     * 1 - Turn on Flexconnect
     * 0 - Turn off Flexconnect
     */
    UINT16 Config = 0x8040;

    // Issue a fleconnect command to the opened device
    if(hDevice!=NULL){
        result = MchpUsbFlexConnect(hDevice, Config);
        if(result==MCHP_Error_Success){
            // Flexconnect was successful
        }
    }
}
```

3.3 GPIO Bridging APIs

This section enumerates the APIs that facilitate low level control of GPIO pins in Microchip USB hubs. User can configure the direction, pull up / down, read data & write data to any GPIO. Note that some of the GPIO may be already assigned to other functions, which may need to be reassigned by the user by calling MchpUsbXdataRead & MchpUsbXdataWrite API (to write different registers) before calling these APIs.

3.3.1 MchpUsbGpioConfigurePulldown

Prototype:

MCHP_USB_ERROR MchpUsbGpioConfigurePulldown (HANDLE DevID, DWORD dwGPIOMaskHi, DWORD dwGPIOMaskLo, BOOL bEnablePulldown)

Devices Supported: USB2534 Family

Description:

This API enables or disables the pull down for the specified GPIO lines. The GPIO mask is a 64 bit value with the higher order bits (63-32) mentioned in the dwGPIOMaskHi and the lower order bits (31-0) mentioned in dwGPIOMaskLo.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device for which the GPIO lines will be pulled down
IN	dwGPIOMaskHi	Mask of bits in the order PIO63:PIO32. The bits which are set select the specific PIOs for this function to operate
IN	dwGPIOMaskLo	Mask of bits in the order PIO31:PIO0. The bits which are set select the specific PIOs for this function to operate
IN	bEnablePulldown	TRUE : Enables pull down FALSE : Disables pull down

Returns:

Mchp_Error_Success – if the operation succeeded

Errorcode as defined in MCHP_USB_ERROR or Windows System call error code - for failure.

The function returns MCHP_Error_Invalid_Device_Handle if the device handle did not match an entry of the open devices list maintained by the library.

Sample:

```

HANDLE hDevice;
INT nGPIOState;

hDevice = MchpUsbOpen (0x0424, 0x2530);

if (hDevice != INVALID_HANDLE_VALUE)
{
    // Put PIO7, PIO10 & PIO41 in input mode
    MchpUsbGpioGet (hDevice, 7, &nGPIOState); // nGPIOState is ignored
    MchpUsbGpioGet (hDevice, 10, &nGPIOState);
}

```

```

MchpUsbGpioGet (hDevice, 41, &nGPIOState);

// Enable Pulldown for PIO7, PIO10 & PIO41
MchpUsbGpioConfigurePulldown (hDevice, 0x00000200, 0x00000480);
}

```

3.3.2 MchpUsbGpioConfigurePullUp

Prototype:

MCHP_USB_ERROR MchpUsbGpioConfigurePullUp (HANDLE DevID, DWORD dwGPIOMaskHi, DWORD dwGPIOMaskLo, BOOL bEnablePullup)

Devices Supported: USB2534 Family

Description:

This API enables or disables the pull up for the specified GPIO lines. The GPIO mask is a 64 bit value with the higher order bits (63-32) mentioned in the dwGPIOMaskHi and the lower order bits (31-0) mentioned in dwGPIOMaskLo.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device for which the GPIO lines will be pulled down
IN	dwGPIOMaskHi	Mask of bits in the order PIO63:PIO32. The bits which are set select the specific PIOs for this function to operate
IN	dwGPIOMaskLo	Mask of bits in the order PIO31:PIO0. The bits which are set select the specific PIOs for this function to operate
IN	bEnablePullup	TRUE : Enables pull up FALSE : Disables pull up

Returns:

Mchp_Error_Success – if the operation succeeded

Errorcode as defined in MCHP_USB_ERROR or Windows System call error code - for failure.

The function returns MCHP_Error_Invalid_Device_Handle if the device handle did not match an entry of the open devices list maintained by the library.

Sample:

```

HANDLE hDevice;

hDevice = MchpUsbOpen (0x0424, 0x2530);

if (hDevice != INVALID_HANDLE_VALUE)

```

```

{
    // Programmer knows that PI03 & PI08 are already in input mode

    // Enable Pulldown for PI03 & PI08
    MchpUsbGpioConfigurePullUp (hDevice, 0x00000000, 0x00000108);
}

```

3.3.3 MchpUsbGpioGet

Prototype:

MCHP_USB_ERROR MchpUsbGpioGet (HANDLE DevID, INT PIONumber, INT* Pinstate)

Devices Supported: USB2534 Family

Description:

This API gets the state of the specified GPIO pin. The direction of the GPIO pin referred in PIONumber is set to IN in this function.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	PIONumber	The GPIO pin number from which to read the pin state
OUT	Pinstate	0 = Pin state is High 1 = Pin state is Low

Returns:

Mchp_Error_Success – if the operation succeeded

Errorcode as defined in MCHP_USB_ERROR or Windows System call error code - for failure.

The function returns MCHP_Error_Invalid_Device_Handle if the device handle did not match an entry of the open devices list maintained by the library.

Sample:

```

HANDLE hDevice;
INT nPI03State, nPI08State;

hDevice = MchpUsbOpen (0x0424, 0x2530);

if (hDevice != INVALID_HANDLE_VALUE)
{
    // Read the pin state of PI03 & PI08
    MchpUsbGpioGet (hDevice, 3, &nPI03State);
    MchpUsbGpioGet (hDevice, 8, &nPI08State);
}

```

3.3.4 MchpUsbGpioSet

Prototype:

MCHP_USB_ERROR MchpUsbGpioSet (HANDLE DevID, INT PIONumber, INT Pinstate)

Devices Supported: USB2534 Family

Description:

This API sets the state of the specified GPIO pin with the state mentioned in Pinstate. The GPIO pin direction is set to OUT in this function.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	PIONumber	The GPIO pin number from which to read the pin state
IN	Pinstate	0 = Set pin state to High 1 = Set pin state to Low

Returns:

Mchp_Error_Success – if the operation succeeded

Errorcode as defined in MCHP_USB_ERROR or Windows System call error code - for failure.

The function returns MCHP_Error_Invalid_Device_Handle if the device handle did not match an entry of the open devices list maintained by the library.

Sample:

```
HANDLE hDevice;  
  
hDevice = MchpUsbOpen (0x0424, 0x2530);  
  
if (hDevice != INVALID_HANDLE_VALUE)  
{  
    // Set PIO3=1 & PIO8=0  
    MchpUsbGpioSet (hDevice, 3, 1);  
    MchpUsbGpioSet (hDevice, 8, 0);  
}
```

3.4 XDATA Bridging APIs

This section lists the APIs that enable read / write of register space in Microchip USB hubs.

3.4.1 MchpUsbXdataRead

Prototype:

MCHP_USB_ERROR MchpUsbXdataRead (HANDLE DevID, WORD StartAddress, INT BytesToRead, BYTE* InputData, INT* ActualBytesRead)

Devices Supported: USB2534 Family

Description:

This function reads a specified number of bytes mentioned in BytesToRead starting from the StartAddress.

Note: The caller function should allocate enough memory mentioned by the BytesToRead field before passing the InputData pointer to the function.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	StartAddress	Start address to read from (in the XDATA space)
IN	BytesToRead	Number of bytes to read
OUT	InputData	Pointer to the buffer where data from XDATA registers will be stored. Caller must allocate memory for the buffer to accommodate the number of bytes to be read.
OUT	ActualBytesRead	Holds the number of bytes actually read.

Returns:

Mchp_Error_Success – if the operation succeeded

Errorcode as defined in MCHP_USB_ERROR or Windows System call error code - for failure.

Sample:

```
/*  
* Obtain a valid hDevice USB2530 device handle  
*  
*/
```



```

unsigned char buff[512]          = {0};
unsigned int ActualBytesRead      = 0;

MCHP_USB_ERROR result = MchpUsbXdataRead(hDevice,
                                          0x3000, // Read configuration registers from 3000-31FF
                                          512,
                                          buff,
                                          &ActualBytesRead);

if(resut==MCHP_Error_Success){
    // XData read was successful
}

```

3.4.2 MchpUsbXdataWrite

Prototype:

MCHP_USB_ERROR MchpUsbXdataWrite (HANDLE DevID, WORD StartAddress, INT BytesToWrite, BYTE* OutputData, INT* ActualBytesWritten)

Devices Supported: USB2534 Family

Description:

This function writes a specified number of bytes mentioned in the BytesToWrite parameter starting from the StartAddress.

Note: The caller function should allocate enough memory mentioned by the BytesToWrite field before passing the InputData pointer to the function.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	StartAddress	Start address to read from (in the XDATA space)
IN	BytesToWrite	Number of bytes to read
IN	OutputData	Pointer to the buffer containing data to write to XDATA registers.
OUT	ActualBytesWritten	Holds the number of bytes actually written.

Returns:

Mchp_Error_Success – if the operation succeeded

Errorcode as defined in MCHP_USB_ERROR or Windows System call error code - for failure.

Sample:

```

/*
* Obtain a valid hDevice USB2530 device handle

```

```
*  
*/  
  
unsigned char byBCConfigPort1;  
unsigned int ActualBytesWritten = 0;  
  
// Enable downstream Battery charging for Port 1  
byBCConfigPort1 = 0xD7;  
  
MCHP_USB_ERROR result = MchpUsbXdataWrite(hDevice,  
                                           0x413C,  
                                           1,  
                                           &byBCConfigPort1,  
                                           &ActualBytesWritten);  
  
if(resut==MCHP_Error_Success){  
    // XData write was successful  
}
```

3.5 I2C Bridging APIs

Microchip USB hubs facilitate USB-I2C bridging through USB control point of the embedded USB device (5th port). This section explains the relevant high level APIs.

3.5.1 MchpUsbI2CSetConfig

Prototype:

MCHP_USB2530API_API MCHP_USB_ERROR MchpUsbI2CSetConfig (
HANDLE DevID, BOOL Enable, INT ClockRate)

Devices Supported: USB2534 Family

Description:

This function enables or disables I2C pass-through and the clock rate of the I2C Master device.

Note: The clock rate parameter is not used and is always 62.5 KHz.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	Enable	TRUE : Enable I2C pass-through FALSE : Disable I2C pass-through
IN	ClockRate	Reserved for Future use

Returns:

Mchp_Error_Success – if the operation succeeded

Errorcode as defined in MCHP_USB_ERROR or Windows System call error code - for failure.

Sample:

```
/*
 * Obtain a valid hDevice USB2530 device handle
 * Enable I2C Pass Thru
 */
MCHP_USB_ERROR error = MchpUsbI2CSetConfig(hDevice, TRUE, NULL);

if(error != MCHP_Error_Success) {

    // I2C Pass Thru enable succeeded

}
```

3.5.2 MchpUsbI2CGetConfig

Prototype:

MCHP_USB2530API_API MCHP_USB_ERROR MchpUsbI2CGetConfig (HANDLE DevID, BOOL* Enable, INT* ClockRate)

Devices Supported: USB2534 Family

Description:

This function gets the configuration state like whether I2C pass-through is enabled or not and the clock rate of the I2C Master device.

Note: The clock rate is always 62.5 KHz for USB2534 family of devices

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
OUT	Enable	A BOOL pointer: TRUE : I2C Pass-through is Enabled FALSE : I2C pass-through is Disabled
OUT	ClockRate	An INT pointer that contains clock rate of the I2C Master. Note: This value is 62.5 KHz for USB2534 family of devices

Returns:

Mchp_Error_Success – if the operation succeeded

Errorcode as defined in MCHP_USB_ERROR or Windows System call error code - for failure.

Sample:

```
BOOL PassThruEnable = FALSE;
INT ClockRate = 0;

MCHP_USB_ERROR error = MchpUsbI2CGetConfig( hDevice, // Valid device handle
                                             &PassThruEnable,
                                             &ClockRate);

if(error == MCHP_Error_Success) {
    // I2CGetConfig succeeded
    // if the PassThruEnable is TRUE I2C Pass thru is enabled
}
```

3.5.3 MchpUsbI2CRead

Prototype:

```
MCHP_USB_ERROR MchpUsbI2CRead ( HANDLE DevID, INT BytesToRead,
                                BYTE* InputData, BYTE bySlaveAddress )
```

Devices Supported: USB2534 Family

Description:

This API performs an I2C read through the I2C pass-through interface of USB2530 device.

Note: A successful I2C EnablePassthrough should be done using MchpUsbI2CSetConfig API before using this function.

The memory for InputData should be allocated to the size of BytesToRead bytes by the calling function.

As the firmware acts as a complete pass through for the I2C read / write requests, the caller should provide the slave address parameter in I2C compliant format as described in the table below (7 MSBs for address and a value of 1 in bit 0 for a read operation).

	Slave Address							R = 1
Bits	7	6	5	4	3	2	1	0
Example	1	0	1	0	0	0	0	1

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device

IN	BytesToRead	Length of bytes to read from I2C slave
IN	InputData	Pointer to input data buffer to retrieve the bytes read
IN	bySlaveAddress	Slave address from which the read operation is to be performed

Returns:

Error_Success - Successfully performed the I2C transfer.

Error_I2C_Transfer_Cmd_Failed - I2C Transfer failed.

Sample:

```

/*
 * Enable I2C Pass thru using MchpUsb2530 API;
 * Program EEPROM through I2C from starting addr 0
 * Disable I2C Pass thru once done
 */

// EEPROM
#define EEPROM_ADDR          0xA0

MCHP_USB_ERROR error = MCHP_Error_Success;
BYTE gbyEEPROMBuffer[9];
BYTE byAddr = EEPROM_ADDR;

WORD wOffset = 0;
unsigned char pbyBuffer[1024] = {0};

// Enable pass thru
error = MchpUsbI2CSetConfig(hDevice, TRUE, NULL);
if(error != MCHP_Error_Success) {
    // return error;
}

if (wOffset < 256) {
    gbyEEPROMBuffer[0] = wOffset;
} else {
    gbyEEPROMBuffer[0] = (wOffset - 256);
    byAddr                |= 0x02;
}

// Write the address
error = MchpUsbI2CWrite(hDevice, 1, gbyEEPROMBuffer, byAddr);

if (error == MCHP_Error_Success) {
    // Read the value to pbyBuffer
    // Append READ bit (0x1) to the slave address
    error = MchpUsbI2CRead(hDevice, wDataLen, pbyBuffer, (byAddr | 0x01));
}

// Disable pass thru
error = MchpUsbI2CSetConfig(hDevice, FALSE, NULL);

```

3.5.4 MchpUsbI2CWrite

Prototype:

MCHP_USB_ERROR MchpUsbI2CWrite(HANDLE DevID, INT BytesToWrite, BYTE* OutputData, BYTE bySlaveAddress)

Devices Supported: USB2534 Family

Description:

This API performs an I2C write through the I2C pass-through interface of USB2530 device.

Note: A successful I2C EnablePassthrough should be done using MchpUsbI2CSetConfig API before using this function.

As the firmware acts as a complete pass through for the I2C read / write requests, the caller should provide the slave address parameter in I2C compliant format as described in the table below (7 MSBs for address and a value of 0 in bit 0 for a write operation).

	Slave Address							R = 0
Bits	7	6	5	4	3	2	1	0
Example	1	0	1	0	0	0	0	0

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	BytesToRead	Length of bytes to write to the I2C slave
IN	OutputData	Pointer to data buffer which contains the data to write to the slave
IN	bySlaveAddress	Slave address to write the data buffer to.

Returns:

Error_Success - Successfully performed the I2C transfer.
 Error_I2C_Transfer_Cmd_Failed - I2C Transfer failed.

Sample:

```
/*
 * Enable I2C Pass thru using MchpUsb2530 API;
 * Program EEPROM through I2C
 * Disable I2C Pass thru once done
 */
```

```
// EEPROM
#define EEPROM_ADDR          0xA0

MCHP_USB_ERROR error = MCHP_Error_Success;
WORD wBytesSent = 0;
BYTE byNumBytes = 0;
BYTE byAddr      = EEPROM_ADDR;
BYTE gbyEEPROMBuffer[9];

// Assumed that this buffer will be filled with data by other calls
unsigned char pbyBuffer[1024] = {0};

error = MchpUsbI2CSetConfig(hDevice, TRUE, NULL);

if(error != MCHP_Error_Success)
    return error;

//Program 8 bytes at a time
do
{
    //Prepare buffer
    byNumBytes = 1;

    if((wOffset + wBytesSent) < 256)
    {
        gbyEEPROMBuffer[0] = (wOffset + wBytesSent);
    }
    else
    {
        gbyEEPROMBuffer[0] = (wOffset + wBytesSent) - 256;
        byAddr              |= 0x02;
    }

    for(BYTE i = 0; i < 8; i++)
    {
        if((wBytesSent + i) < wDataLen)
        {
            gbyEEPROMBuffer[i + 1] = pbyBuffer[wBytesSent + i];
            byNumBytes++;
        }
    }

    error = MchpUsbI2CWrite(hDevice, byNumBytes, gbyEEPROMBuffer, byAddr);
    if(MCHP_Error_Success == error)
    {
        wBytesSent += (byNumBytes - 1);
    }
    else
    {
        break;
    }

    //Allow 10ms worstcase for EEPROM to do the actual write
    Sleep(10);
}
```

```
}while(wBytesSent < wDataLen);  
  
// Disable pass thru  
error = MchpUsbI2CSetConfig(hDevice, FALSE, NULL);
```

3.6 OTP Bridging APIs

Microchip hubs support configuration of the hub through OTP memory. This section enumerates the APIs relevant for OTP memory read / program.

3.6.1 MchpUsbOtpRead

Prototype:

MCHP_USB_ERROR MchpUsbOtpRead (HANDLE DevID, INT MemoryLocation, INT BytesToRead, BYTE* OTPBuffer, INT * ActualBytesRead)

Devices Supported: USB2534 Family

Description:

This API reads bytes of data mentioned in the BytesToRead parameter from the OTP memory region of the USB2530 device starting at address mentioned in the MemoryLocation parameter.

Note: The memory for OTPBuffer should be allocated to the size of BytesToRead bytes by the calling function.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	MemoryLocation	Start address in OTP from where the read operation starts
IN	BytesToRead	Pointer to input data buffer to retrieve the bytes read
OUT	OTPBuffer	Pointer to the buffer where the data read from OTP memory region will be stored.
OUT	ActualBytesRead	Pointer where the number of bytes actually read from OTP Space is returned.

Returns:

Mchp_Error_Success - for operation success

Errorcode (defined in MCHP_USB_ERROR or Windows System call error code) - for failure

Sample:


```

unsigned char buff[1024] = {0};
int ActualBytesRead = 0;

// Read 1KB of OTP starting from offset 0
MCHP_USB_ERROR error = MchpUsbOtpRead( hDevice, // Valid device handle
                                       0x0,    // Start address 0
                                       1024,   // Bytes to read (<= buffer size)
                                       buff,    // OTPBuffer
                                       &ActualBytesRead); // Actual bytes read

if(resut==MCHP_Error_Success){
    // OtpRead Succeeded
}

```

3.6.2 MchpUsbOtpWrite

Prototype:

MCHP_USB_ERROR MchpUsbOtpWrite (HANDLE DevID, INT MemoryLocation, INT BytesToWrite, BYTE* ConfigData, INT * ActualBytesWritten)

Devices Supported: USB2534 Family

Description:

This API writes bytes of data as mentioned in the BytesToWrite parameter to the OTP memory region of the USB2530 device beginning from the memory location contained in MemoryLocation parameter.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	MemoryLocation	Start address in OTP from where the read operation starts
IN	BytesToWrite	Pointer to input data buffer to retrieve the bytes read
IN	ConfigData	Pointer to the buffer where the data read from OTP memory region will be stored.
OUT	ActualBytesWritten	Pointer where the number of bytes actually read from OTP Space is returned.

Returns:

Mchp_Error_Success - for operation success
 Errorcode (defined in MCHP_USB_ERROR or Windows System call error code) - for failure

Sample:

```
unsigned char buff[1024] = {0};
int ActualBytesRead = 0;

// Program 520 bytes starting at offset 2
MCHP_USB_ERROR error = MchpUsbOtpWrite( hDevice, // Valid device handle
    0x2,    // Start address 2
    520,    // Bytes to write (<= buffer size)
    buff,   // OTPBuffer contains the data to be written
    &ActualBytesRead ); // Actual bytes read

if(resut==MCHP_Error_Success){
    // OtpWrite Succeeded
}
```

3.7 RAM Bridging APIs

3.7.1 MchpUsbRamRead

This API is a placeholder reserved for future use and currently just duplicates the functionality of MchpUsbXdataRead API

3.7.2 MchpUsbRamWrite

This API is a placeholder reserved for future use and currently just duplicates the functionality of MchpUsbXdataWrite API

3.8 SPI Bridging APIs

This section lists all the USB-SPI bridging APIs.

3.8.1 MchpUsbSpiSetConfig

Prototype:

```
MCHP_USB_ERROR MchpUsbSpiSetConfig ( HANDLE DevID, UINT32
EnterExit, UCHAR* IDorMCHP_USB_ERROR )
```

Devices Supported: USB2534 Family

Description:

This API enables/disables the SPI interface with USB2534. If SPI control register is not edited by the user then this function would put SPI in default mode i.e, mode0 and dual_out_en = 0. Speed is dependent totally on the strap options.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	EnterExit	1 - Pass thru Enter 0 - Pass thru Exit
OUT	IDorMCHP_USB_ERROR	Reserved for Future use

Returns:

Mchp_Error_Success - for operation success

Errorcode (defined in MCHP_USB_ERROR or Windows System call error code) - for failure

Sample:

```

/*
 * Enter SPI Pass thru using MchpUsb2530 API; hDevice is a valid USB2530 device
 handle
 */
MCHP_USB_ERROR error = MchpUsbSpiSetConfig( hDevice, 1, NULL);
if (MCHP_Error_Success != error)
{
    // SPI Pass thru succeeded
}

```

3.8.2 MchpUsbSpiWrite**Prototype:**

MCHP_USB_ERROR MchpUsbSpiWrite(HANDLE DevID, UINT BytesToWrite, UCHAR* OutputData, UINT wTotalLength)

Devices Supported: USB2534 Family

Description:

This API is a low level SPI pass thru command Write. All commands to the SPI interface are directed as SPI Pass thru write.

Note: SPI pass thru enter must have been done before executing this API.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	BytesToWrite	The BytesToWrite is utilized to mention the number of bytes the SPI flash will return.
IN	OutputData	Buffer containing the command/ data to be sent to the device.
IN	wTotalLength	This field is the size of USB command OUT packet being sent to the firmware.

Returns:

Mchp_Error_Success - for operation success

Errorcode (defined in MCHP_USB_ERROR or Windows System call error code) - for failure

Sample:

```
/*
 * Enter SPI Pass thru using MchpUsb2530 API;
 * hDevice is a valid USB2530 device handle;
 * Send a read JEDEC ID command through a pass thru write
 * Exit SPI pass thru once done
 */

// JEDEC ID read op-code is 0x9F; Refer SPI Opcodes in PAS for further details
BYTE abyjedecid_cmd[] = {0x9f};

/*
 * Issue Read jedec id command via pass thru write
 */
MCHP_USB_ERROR error = MchpUsbSpiWrite(hDevice, 1, &abyjedecid_cmd[0], 4);
if(MCHP_Error_Success == error)
{
    // Pass thru write succeeded
    // ...
    MchpUsbSpiSetConfig( hDevice, 0, NULL); // Exit pass thru
}
```

3.8.3 MchpUsbSpiRead

Prototype:

MCHP_USB_ERROR MchpUsbSpiRead(HANDLE DevID, UINT BytesToRead, UCHAR* InputData, UINT wTotalLength)

Devices Supported: USB2534 Family

Description:

This API is the low level SPI pass thru command read.

Note: SPI pass thru enter must been done before executing this API.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	BytesToRead	The BytesToRead is utilized to mention the number of bytes the SPI flash will return for the pass thru command

IN	InputData	Buffer used to store the data received from the device.
IN	wTotalLength	This field is the size of USB command OUT packet being sent to the firmware

Returns:

Mchp_Error_Success - for operation success

Errorcode (defined in MCHP_USB_ERROR or Windows System call error code) - for failure

Sample:

```

/*
 * Enter SPI Pass thru using MchpUsb2530 API;
 * hDevice is a valid USB2530 device handle;
 * Send a read JEDEC ID command through a pass thru write
 * Read the jedec id from the response buffer using pass thru read
 * Exit SPI pass thru once done
 */

typedef struct tagJEDECID
{
    BYTE byManufacturer;
    BYTE byMemoryType;
    BYTE byDeviceId;
    char szPartNumber [20];

    BYTE byValidationMask;
} JEDECID;

// JEDEC ID read op-code is 0x9F; Refer SPI Opcodes in PAS for further details
BYTE abyjedecid_cmd[] = {0x9f};
BYTE abyjedecid_rsp[10] = {0};

/*
 * Issue Read jedec id command via pass thru write
 */
MCHP_USB_ERROR error = MchpUsbSpiWrite(hDevice, 1, &abyjedecid_cmd[0], 4);

// Read the ID
error = MchpUsbSpiRead(hDevice, 4, &abyjedecid_rsp[0], 4);

If (MCHP_Error_Success == error)
{
    // SPI read succeeded
    // Byte 0 in the response buffer is the response for opcode 0x9F. So
    // ignore.
    // byte 1 is the manufacturer, byte2 memory and is based on the JEDECID
    // struct provided above
    // ...
    MchpUsbSpiSetConfig( hDevice, 0, NULL); // Exit pass thru
}

```

3.9 UART Bridging APIs

This section lists all the USB-UART bridging APIs.

3.9.1 MchpUsbUartSetConfig

Prototype:

MCHP_USB_ERROR MchpUsbUartSetConfig(HANDLE DevID, UINT32 BaudRate, BOOL StopBits, BOOL ParityEn, BOOL ParityOdd)

Devices Supported: USB2534 Family

Description:

This API enables the UART device and sets the baud rate for serial communication.

Note:

Non-standard baud rates different from the ones specified here are also possible. Make sure that the other paired sender/receiver also uses the same baud rate.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	BaudRate	Baud rate to be set. Suggested standard values are 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200.
IN	StopBits	Not Applicable. Included for Compatibility. Set value to NULL.
IN	ParityEn	Not Applicable. Included for Compatibility. Set value to NULL.
IN	ParityOdd	Not Applicable. Included for Compatibility. Set value to NULL.

Returns:

Mchp_Error_Success - for operation success

Errorcode (defined in MCHP_USB_ERROR or Windows System call error code) - for failure

Sample:

```
/* Obtain a valid device, hDevice, through a call to one of device handle open APIs. Then set baud rate to 9600 */
```

```
MCHP_USB_ERROR error = MchpUsbUartSetConfig(hDevice, 9600, 0, 0, 0);
```

```
if (error == MCHP_Error_Success) {  
    // Baud rate set to 9600  
}
```

3.9.2 MchpUsbUartWrite

Prototype:

MCHP_USB_ERROR MchpUsbUartWrite (HANDLE DevID, UINT32 BytesToWrite, UCHAR* OutputData)

Devices Supported: USB2534 Family

Description:

This API transfers data through serial port to the connected serial peripheral.

Note:

Set Baud rate using MchpUsbUartSetConfig API before calling this API.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	BytesToWrite	Length of bytes to transfer from serial port
IN	OutputData	Pointer to output data buffer which contains the data to transfer

Returns:

Mchp_Error_Success - for operation success

Errorcode (defined in MCHP_USB_ERROR or Windows System call error code) - for failure

Sample:

```
/* 1. Obtain a valid device, hDevice, through a call to one of device handle
open APIs.
2. Then set baud rate to 9600
3. Transmit some data*/
unsigned char transmit_data[512] = {0};           // OutputData

// Some function that would fill random data
RandomFillData(transmit_data);

unsigned int bytes_to_write = 512;

if (ERROR_SUCCESS == MchpUsbUartWrite(hDevice, bytes_to_writeto_write,
transmit_data)) {
    // UART write success
}
```

3.9.3 MchpUsbUartRead

Prototype:

MCHP_USB_ERROR MchpUsbUartRead(HANDLE DevID, UINT32 BytesToRead, UCHAR* InputData)

Devices Supported: USB2534 Family

Description:

This API synchronously receives data through serial port from the connected serial peripheral

Note:

Set Baud rate using MchpUsbUartSetConfig API before calling this API.

This API call is a blocking one and will not return until it receives the specified number of bytes.

The calling function should allocate memory for the InputData buffer as mentioned in the BytesToRead parameter.

Parameters:

↓↑	Parameter	Description
IN	DevID	Handle to the USB2530 device
IN	BytesToRead	Length of bytes to transfer to the serial port
OUT	InputData	Pointer to input data buffer which contains the data to transfer

Returns:

Mchp_Error_Success - for operation success

Errorcode (defined in MCHP_USB_ERROR or Windows System call error code) - for failure

Sample:

```
/* 1. Obtain a valid device, hDevice, through a call to one of device handle  
open APIs.  
2. Then set baud rate to 9600  
3. Receive some data*/  
unsigned char ReadBuffer [120] = {0};
```



```
MCHP_USB_ERROR error = MCHP_Error_Success;
int receiveSize = 100; // <= sizeof ucInput

if (ERROR_SUCCESS == MchpUsbUartRead(hDevice, receiveSize, ReadBuffer)) {
    // UART read success
}
```

4 APPENDIX A

4.1 Classes & Structs

4.1.1 struct _DEVICE_INTERFACE_

```
typedef struct _DEVICE_INTERFACE_{
    DWORD    dNoOfDevices;    /*!< Number of devices */
    WCHAR    **DevicePath;    /*!< List of device instance paths */
} DEVIO, *PDEVIO;
```

Devices Supported: USB2534 Family

Description:

This structure holds the number of devices and their device paths

4.2 Typedefs & Macros

4.2.1 Macros

```
#define MCHP_USB2530API_API __declspec(dllimport)
```

4.2.2 Typedefs

```
typedef struct _DEVICE_INTERFACE_ DEVIO
```

```
typedef struct _DEVICE_INTERFACE_* PDEVIO
```

```
typedef enum tagMCHP_USB_ERROR MCHP_USB_ERROR
```

4.2.3 MCHP_USB_ERROR

Description:

This list of errors is defined in the `enum tagMCHP_USB_ERROR` which can be found in the `McpUSBInterface.h` header file distributed along with the library.

The following are the errors, their values and descriptions.

#	MCHP_USB_ERROR	Value	Description
1	MCHP_Error_Success	0x0000	Operation Success
2	MCHP_Error_Device_Not_Found	0x0001	The specific device was not found
3	MCHP_Error_Invalid_Argument	0x0002	Argument passed to the API is invalid
4	MCHP_Error_Invalid_Device_Handle	0x0003	Device handle passed to the API is not valid
5	MCHP_Error_WinUSBAPIL_Fail	0x0004	API of the winusb library failed
6	MCHP_Error_ApiNotSupported	0x0005	This particular API is not supported for this hub family
7	MCHP_Loadbinfail	0x1000	Could not load the binary file
8	MCHP_ReadBackFailed	0x1001	Reading from SPI flash failed
9	MCHP_WrongFileSize	0x1002	File size did not match
10	MCHP_SpiPassThruWriteFailed	0x1003	SPI pass through write command failed
11	MCHP_SpiPassThruEnterFailed	0x1004	SPI Pass thru Enter command failed
12	MCHP_SpiNoDevice	0x1005	SPI flash could not be detected or not present
13	MCHP_SpiCancelDl	0x1006	SPI Cancel Download
14	MCHP_SpiProgrammingFailed	0x1007	SPI flash programming failed
15	MCHP_SpiPassThruExitFailed	0x1008	SPI Pass thru Enter command failed
16	MCHP_SpiPassThruReadFailed	0x1009	SPI pass through read command failed
17	MCHP_SpiFlashWrongDeviceID	0x100A	Unsupported SPI flash detected
18	MCHP_SpiFWCompareFailed	0x100B	SPI flash read back and compare failed with programmed binary
19	MCHP_SpiOpenEraseSigFileFailed	0x100C	Open Erase signature bin file failed
20	MCHP_SpiReadEraseSigFileFailed	0x100D	Read Erase signature bin file failed
21	MCHP_SpiSRAMProgFailed	0x100E	SRAM programming failed
22	MCHP_SpiEraseSignatureFailed	0x100F	SPI_ERASE_4KBSECTOR command failed. Applicable for USB5734 only
23	MCHP_SpiChipEraseFailed	0x1010	Chip Erase command failed

24	MCHP_Error_I2C_Passthrough_Enter_Cmd_Failed	0x2000	Cannot enable I2C Pass thru interface
25	MCHP_Error_I2C_Passthrough_Exit_Cmd_Failed	0x2001	Cannot disable I2C Pass thru interface
26	MCHP_Error_I2C_Transfer_Cmd_Failed	0x2002	I2C Transfer failed
27	MCHP_Error_I2C_Max_Size_Error	0x2003	I2C MAX Size Error
28	MCHP_SMBPortOpenErr	0x3000	SMBus port open error
29	MCHP_SMBBitRateErr	0x3001	Bitrate error
30	MCHP_SMBDataLenErr	0x3002	Data length error
31	MCHP_SMBWriteErr	0x3003	SMBus write access failed
32	MCHP_SMBReadErr	0x3004	SMBus read access failed
33	MCHP_SMBCloseErr	0x3005	SMBus close error
34	MCHP_Error_OTP_Invalid_BufferSize	0x3000	Invalid buffer size
35	MCHP_Error_UART_BaudrateErrorPercentTooHigh	0x4000	Communication at the specified baud rate will be error prone
36	MCHP_Error_UART_Set_Regs	0x4001	Cannot set USB2534 UART registers, probably command failure
37	MCHP_Error_UART_PC_To_Device_Sent0Bytes	0x4002	Transmit failed without transmitting any data
38	MCHP_Error_UART_PC_To_Device_Failed	0x4003	Transmit failed after transmitting some data
39	MCHP_Error_UART_Rx_Buffer_Overflow	0x4004	Receive failed due to buffer overrun, reduce baud rate
40	MCHP_Error_UART_RxFIFO_Status_Unexpected	0x4005	Receive failed due to unexpected Rx FIFO status
41	MCHP_Error_UART_Rx_Thread_Creation_Failed	0x4006	Receive failed since worker thread creation failed
42	MCHP_Error_UART_Rx_Pending	0x4007	UART Rx is pending due to asynchronous mode
43	MCHP_Error_UART_Rx_UserAbort	0x4008	UART receive aborted as per user request
44	MCHP_Error_UART_Rx_Command_Failed	0x4009	UART Receive command failed by the firmware
45	MCHP_Error_UART_Rx_0_Bytes	0x400A	Receive failed without receiving any data
46	MCHP_Error_OTP_Check_Fail	0x5000	Check OTP command failed
47	MCHP_Error_Undefined	0xFFFF	Unknown error occurred

