

Protouch2 DLL User Guide

Version 1.8

Table of Contents

1 Introduction	1-1
2 Legal Information	2-2
3 HFC Driver Installation/Uninstallation	3-3
3.1 Driver Installation	3-4
3.2 Driver Uninstallation	3-5
4 LAN78xx Driver Installation	4-6
5 Logging	5-7
6 INI File Description	6-8
7 List of APIs	7-10
7.1 Common APIs	7-11
7.1.1 MchpUsbGetVersion Function	7-11
7.1.2 MchpEnableLogging Function	7-12
7.2 USB Hub APIs	7-14
7.2.1 Device Open / Close APIs	7-14
7.2.1.1 MchpUsbGetAllHubs Function	7-14
7.2.1.2 MchpUsbOpen Function	7-15
7.2.1.3 MchpUsbOpenID Function	7-17
7.2.1.4 MchpUsbHCEOpen Function	7-18
7.2.1.5 MchpUsbClose Function	7-19
7.2.2 GPIO Bridging APIs	7-21
7.2.2.1 MchpUsbConfigureGPIO Function	7-21
7.2.2.2 MchpUsbGpioGet Function	7-22
7.2.2.3 MchpUsbGpioSet Function	7-25
7.2.3 XDATA Bridging APIs	7-27
7.2.3.1 MchpUsbRegisterRead Function	7-27
7.2.3.2 MchpUsbRegisterWrite Function	7-29
7.2.4 I2C Bridging APIs	7-31
7.2.4.1 MchpUsbI2CSetConfig Function	7-31
7.2.4.2 MchpUsbI2CRead Function	7-33
7.2.4.3 MchpUsbI2CWrite Function	7-35

7.2.4.4 MchpUsbI2CTransfer Function	7-37
7.2.5 OTP Bridging APIs	7-39
7.2.5.1 MchpUsbOtpRead Function	7-39
7.2.5.2 MchpUsbOtpWrite Function	7-41
7.2.6 SPI Bridging APIs	7-43
7.2.6.1 MchpUsbSpiSetConfig Function	7-43
7.2.6.2 MchpUsbSpiFlashRead Function	7-45
7.2.6.3 MchpUsbSpiFlashWrite Function	7-47
7.2.6.4 MchpUsbSpiTransfer Function	7-50
7.2.7 UART Bridging APIs	7-52
7.2.7.1 MchpUsbEnableUARTBridging Function	7-53
7.2.7.2 MchpUsbSetUARTBaudrate Function	7-54
7.2.7.3 MchpUsbUartRead Function	7-56
7.2.7.4 MchpUsbUartWrite Function	7-58
7.2.7.5 MchpUsbUartReadTimeOut Function	7-61
7.2.8 Flexconnect API	7-63
7.2.8.1 MchpUsbFlexConnect Function	7-63
7.2.8.2 MchpUsbWriteFlexConfigArea Function	7-65
7.2.9 Programming APIs	7-66
7.2.9.1 MchpProgramFile Function	7-67
7.2.9.2 MchpProgramSPIFile Function	7-68
7.2.9.3 MchpProgramFileWithSerial Function	7-70
7.2.9.4 MchpUsbProgramFileWithUsb3Serial Function	7-72
7.2.9.5 MchpProgramSPIFirmwareWithConfig Function	7-74
7.2.9.6 MchpProgramSpiFwCfgFileWithSerial Function	7-75
7.2.9.7 MchpProgramSPIFirmwareWithConfigBuffer Function	7-77
7.2.10 Miscellaneous APIs	7-78
7.2.10.1 MchpUsbGetLastErr Function	7-79
7.2.10.2 MchpVerifyWidgetValues Function	7-80
7.2.10.3 MchpUsbDeviceReset Function	7-83
7.2.10.4 MchpUsbHCETransfer Function	7-84
7.2.10.5 MchpUsbVSMTransfer Function	7-87
7.2.10.6 MchpUsbGetAllHubsPortChainInfo Function	7-89
7.2.10.7 MchpGetHubPortChain Function	7-90
7.2.10.8 MchpGetHubIndex Function	7-92
7.3 LAN APIs	7-95
7.3.1 Device Open/Close	7-95
7.3.1.1 MchpLanGetDevices Function	7-95
7.3.1.2 MchpLanCloseAllHandles Function	7-95
7.3.1.3 MchpLanOpen Function	7-97
7.3.1.4 MchpLanOpenID Function	7-98
7.3.1.5 MchpLanClose Function	7-99

7.3.2 LAN Programming APIs	7-101
7.3.2.1 MchpLanProgramFile Function	7-101
7.3.2.2 MchpLanProgramFileWithSerial Function	7-102
7.3.3 Register Read/Write	7-104
7.3.3.1 MchpLanRegisterRead Function	7-104
7.3.3.2 MchpLanRegisterWrite Function	7-106
7.3.3.3 MchpLanPhyRegisterWrite Function	7-108
7.3.3.4 MchpLanPhyRegisterRead Function	7-109
7.3.4 Memory Read/Write APIs	7-111
7.3.4.1 MchpLanReadMemory Function	7-111
7.3.4.2 MchpLanReadEepromByte Function	7-113
7.3.4.3 MchpLanWriteEepromByte Function	7-114
7.3.5 Miscellaneous APIs	7-116
7.3.5.1 MchpLanVerifyParameters Function	7-116
7.3.5.2 MchpLanGetDriverVersion Function	7-118
7.3.5.3 MchpLanReadUsbPhyStat Function	7-119
7.3.5.4 MchpLanReadTxRxStat Function	7-121
7.3.5.5 MchpLanOtpConfigureCommand Function	7-122
7.3.5.6 MchpLanGetLastErr Function	7-124
7.3.5.7 MchpLanGetEepromSize Function	7-125
7.3.5.8 MchpLanEepromConfigureCommand Function	7-126
7.3.5.9 MchpLanEraseEeprom Function	7-128
7.3.5.10 MchpLanGetAdapterBits Function	7-129
7.3.5.11 MchpLanGetAdapterKey Function	7-131
7.3.5.12 MchpLanGetAdapterPDO Function	7-132
7.3.5.13 MchpLanGetAdapterPort Function	7-134
8 Error Codes	8-136
9 Frequently Asked Questions	9-140

1 Introduction

Microchip hubs support a wide array of configurable parameters like VID, PID and String Descriptors. These configuration parameters can be stored in a configuration file and programmed on to each devices for mass production. With its high level APIs,the Protouch2 DLL API enables end customers to program and realize the full potential of USB253x/USB4604, USB57x4 LAN78xx and other families of Microchip USB hub products.

2 Legal Information

Software License Agreement

(c) 2004 - 2016 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 HFC Driver Installation/Uninstallation

Usage of USB APIs requires that the WinUSB device driver to be installed for the Hub Feature Controller (HFC) for the Microchip hub with internal USB device being used. The WinUSB device is used for programming the OTP by communicating with the HFC device.

The Protouch2 command line programmer should be used to install and uninstall the WinUSB drivers. It can be found in the released package.(PT2_CLI/pt2main.exe)

3.1 Driver Installation

Refer **Protouch2 CLI User Manual.pdf** for more details about driver installation.

3.2 Driver Uninstallation

Refer **Protouch2 CLI User Manual.pdf** for more details about driver uninstallation.

4 LAN78xx Driver Installation

1. Run the “install.exe” application from the directory “\Drivers\LAN78xxDriver\”.
2. Accept EULA and proceed to the driver installation.
3. Once the driver installation is successful make sure the LAN78xx device is listed in the device manager.

5 Logging

By default,the log file will not be created.

[MchpEnableLogging](#) API must be called at the very beginning to create the log file(PT2.log).

The following levels can be set using [MchpEnableLogging](#) API.

1. No Debug message in log file.
2. Short description in log file.
3. Detailed description in log file.

Refer [MchpEnableLogging](#) API for more details.

6 INI File Description

INI File Name : Protouch2.ini

HUB_VID_LIST :

The application will populate list of hubs connected to the computer with unique index numbers. By default, microchip hubs will be moved to lower index's based on the Vendor ID (VID) during the initial process of application. To move other hubs to lower index position when listing the hubs, Vendor ID needs to be specified in HUB_VID_LIST section of INI file

Example:

```
[HUB_VID_LIST]
```

```
;Microchip VID
HUB_VID1=0x0424
```

Other VID's Section

Add as HUB_VID'n+1' = VID in "0x" format

```
"HUB_VID2=0x8085"
```

Maximum five VID's can be added here.

so the maximum is HUB_VID5= 0XXXXX;

RESTART_DELAY

RESTART_DELAY is the time delay for the device enumeration once the device is restarted. Device will be restarted in the application with RESTART_DELAY timeout for specific commands. Application will wait for the device enumeration up to value specified in the RESTART_DELAY variable.

Example:

```
RESTART_DELAY = 10000 ; In Milliseconds
```

VSM_DEVICE_SPECIFIC_INSTALL:

If VSM_DEVICE_SPECIFIC_INSTALL is TRUE, VSM driver will be installed on the selected hub, same will be uninstalled once the device operations are completed.

If VSM_DEVICE_SPECIFIC_INSTALL is FALSE, VSM driver will not be installed on the selected hub. But VSM HUB CLASS Filter should be installed on the computer.

Note : If the Hub Controller is enabled on the device, then the VSM_DEVICE_SPECIFIC_INSTALL can be made as FALSE and Hub class filter installation is optional.

Example:

```
VSM_DEVICE_SPECIFIC_INSTALL = TRUE
```

[HCE_DEV_INFO]

If the Product ID for Hub Controller needs to be changed, then add Hub Controller Product ID to the HCE_DEV_INFO section in the INI file.

To get the device handle from the Hub Controller device, Product ID needs to be added as shown below.

Maximum entry is 10. (Max is HCE_PID10 = 0x1234 and HCE_PID11 = 0x1234 is not valid one)

[HCE_DEV_INFO]

HCE_PID1 = 0x2530;

HCE_PID2 = 0x2740;

HCE_PID3 = 0x274E;

HCE_PID4 = 0x274F;

HCE_PID5 = 0x274C;



HCE_PID6 = 0x2840;

HCE_PID7 = 0x253C;

7 List of APIs

7.1 Common APIs

Functions

	Name	Description
	MchpUsbGetVersion	Get version number of the DLL.
	MchpEnableLogging	Enables PT2 (PT2.log) log file

7.1.1 MchpUsbGetVersion Function

C

```
MCHP_USB_API BOOL MchpUsbGetVersion(  
    PCHAR VersionNumber  
);
```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX,LAN78XX

Description

This API will get the version number of the DLL.

Preconditions

None.

Parameters

Parameters	Description
VersionNumber	Pointer to the buffer where the version number of the DLL will be stored.

Returns

None.

Remarks

None

Example

```
CHAR sztext[2048];  
typedef BOOL (*pfMchpUsbGetVersion) (PCHAR );  
pfMchpUsbGetVersion libMchpUsbGetVersion;  
  
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");  
if (LoadLib != NULL)  
{  
    printf("Loadlib library Loaded\n");  
}  
libMchpUsbGetVersion = (pfMchpUsbGetVersion) GetProcAddress ( LoadLib  
, "MchpUsbGetVersion");  
if (FALSE == libMchpUsbGetVersion(sztext))  
{  
    printf ("nPress any key to exit....");  
    _getch ();  
    exit (1);  
}  
//Print version number here  
cout << sztext << endl;
```

7.1.2 MchpEnableLogging Function

C

```
MCHP_USB_API void MchpEnableLogging(  
    INT LogLevel  
);
```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX,LAN78XX

Description

This API will set the level of log messages to be printed in the log file(PT2.log file) based on the LogLevel integer variable.

By default, the log messages will not be created. User must call this API with log level to create the log file.

This API will fail to update PT2.log file if this file does not have write access

Preconditions

None.

Parameters

Parameters	Description
LogLevel	0 - No Debug message in log file either if the value is 0 or if enum LOG_NO_MESSAGES is used. 1 - Short description in log file either if the value is 1 or if enum LOG_SHORT_DESCRIPTION is used. 2 - Detailed description in log file either if the value is 2 or if enum LOG_DETAILED_DESCRIPTION is used. All other values are reserved.

Returns

None.

Remarks

Enable only one mode for an operation

Example

```
CHAR sztext[2048];  
typedef BOOL (*pfMchpUsbGetVersion) (PCHAR );  
typedef void (*pfMchpEnableLogging) (INT);  
  
pfMchpUsbGetVersion libMchpUsbGetVersion;  
pfMchpEnableLogging libMchpEnableLogging;  
  
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");  
if (LoadLib != NULL)  
{  
    printf("Loadlib library Loaded\n");  
}  
libMchpUsbGetVersion = (pfMchpUsbGetVersion) GetProcAddress ( LoadLib  
, "MchpUsbGetVersion");  
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress ( LoadLib  
, "MchpEnableLogging");  
if (FALSE == libMchpUsbGetVersion(sztext))  
{  
    printf ("nPress any key to exit....");  
    _getch ();  
    exit (1);  
}  
//Print version number here  
cout << sztext << endl;
```



```
libMchpEnableLogging(LOG_DETAILED_DESCRIPTION);
```

7.2 USB Hub APIs

7.2.1 Device Open / Close APIs

Functions

	Name	Description
⇒	MchpUsbGetAllHubs	Get all programmable hubs.
⇒	MchpUsbOpen	Return handle to the first instance of a device with the specified hub_index.
⇒	MchpUsbOpenID	Return handle to the first instance of VendorID & ProductID matched device.
⇒	MchpUsbHCEOpen	This API will return handle to the Hub Controller Endpoint device.
⇒	MchpUsbClose	Close the device handle.

Description

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

7.2.1.1 MchpUsbGetAllHubs Function

C

```
MCHP_USB_API INT MchpUsbGetAllHubs(  
    PCHAR HubInfo  
);
```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will get all programmable usb hubs which are connected to the system. This API acts as a Global INIT function and hence should be called before any of the functions that access the hub_index. If there is any USB hub insertion or removal after calling this API , then the user can to take care of calling this function again.Also the user should check if the hub_index is correct if more than 1 hub is present in the system.

Preconditions

By default,the log file will not be created.

so [MchpEnableLogging](#) API must be called to create the log file(PT2.log) before calling this API.

Parameters

Parameters	Description
HubInfo	Pointer to the buffer which has minimal information about usb hubs with hub_index.

Returns

No.of Hubs connected to system

Remarks

None

Example

```

typedef INT (*pfMchpUsbGetAllHubs) (PCHAR);
pfMchpUsbGetAllHubs libMchpUsbGetAllHubs;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
libMchpUsbGetAllHubs = (pfMchpUsbGetAllHubs) GetProcAddress ( LoadLib
,"MchpUsbGetAllHubs");

int hub_count = 0;
hub_count = libMchpUsbGetAllHubs(sztext);

if(hub_count)
{
    cout<<"Connected usb hubs are ..nn";
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< hubindex <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

7.2.1.2 MchpUsbOpen Function

C

```

MCHP_USB_API HANDLE MchpUsbOpen(
    CONST UINT HubIndex
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will return handle to the first instance of a device with the specified hub_index.

Preconditions

- [MchpUsbGetAllHubs](#) should be called before calling this API.
- RESTART_DELAY and VSM_DEVICE_SPECIFIC_INSTALL should be configured in the Protouch2.ini file.
- By default,the log file will not be created.
so [MchpEnableLogging](#) API must be called to create the log file(PT2.log) before calling this API.

Parameters

Parameters	Description
HubIndex	Index of the hub connected to the system - MchpUsbGetAllHubs

Returns

HANDLE of the selected HubIndex

INVALID_HANDLE_VALUE (Call GetMchpUsbLastError for more details) - for failure

Remarks

None

Example

```
typedef INT (*pfMchpUsbGetAllHubs) (PCHAR);
typedef HANDLE (*pfMchpUsbOpen) (CONST UINT);
pfMchpUsbGetAllHubs libMchpUsbGetAllHubs;
pfMchpUsbOpen libMchpUsbOpen;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
libMchpUsbGetAllHubs = (pfMchpUsbGetAllHubs) GetProcAddress ( LoadLib
, "MchpUsbGetAllHubs");
libMchpUsbOpen = (pfMchpUsbOpen) GetProcAddress ( LoadLib , "MchpUsbOpen");

int hub_count = 0;
hub_count = libMchpUsbGetAllHubs(sztext);

if(hub_count)
{
    cout<<"Connected usb hubs are ..nn";
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< hubindex <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError = Error_Success;

hDevice = libMchpUsbOpen(hubindex);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError((HANDLE)hubindex);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");
```

7.2.1.3 MchpUsbOpenID Function

C

```
MCHP_USB_API HANDLE MchpUsbOpenID(
    UINT16 VendorID,
    UINT16 ProductID
);
```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will return handle to the first instance of the Hub VendorID & ProductID matched device.

Preconditions

- [MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should not be called before calling this API.
- RESTART_DELAY and VSM_DEVICE_SPECIFIC_INSTALL flags should be configured in the Protouch2.ini file.
- By default,the log file will not be created.

so [MchpEnableLogging](#) API must be called to create the log file(PT2.log) before calling this API.

Parameters

Parameters	Description
VendorID	Vendor ID(VID) of the Hub.
ProductID	Product ID(PID) of the Hub.

Returns

HANDLE of the Vendor ID and Product ID matched hub - for success

INVALID_HANDLE_VALUE (Call GetMchpUsbLastError for more details) - for failure

Remarks

None

Example

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
pfMchpUsbOpenID libMchpUsbOpenID;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
, "MchpUsbOpenID");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
```

```

if(INVALID_HANDLE_VALUE == hDevice)
{
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfullyn");

```

7.2.1.4 MchpUsbHCEOpen Function

C

```

MCHP_USB_API HANDLE MchpUsbHCEOpen(
    UINT16 VendorID,
    UINT16 ProductID
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

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

Preconditions

- [MchpUsbGetAllHubs](#), [MchpUsbOpen](#) and [MchpUsbOpenID](#) should not be called before calling this API.
 - By default, the log file will not be created.
- so [MchpEnableLogging](#) API must be called to create the log file(PT2.log) before calling this API.

Parameters

Parameters	Description
VendorID	Vendor ID of the Hub Controller Endpoint device .
ProductID	Product ID of the Hub Controller Endpoint device .

Returns

HANDLE of the Vendor ID and Product ID matched Hub Controller - for success

INVALID_HANDLE_VALUE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Example

```

typedef HANDLE (*pfMchpUsbHCEOpen) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);

pfMchpUsbHCEOpen libMchpUsbHCEOpen;
pfMchpUsbClose libMchpUsbClose;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loadedn");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
}

```

```

        exit (1);
    }

    libMchpUsbHCEOpen = (pfMchpUsbHCEOpen) GetProcAddress ( LoadLib ,"MchpUsbHCEOpen");
    libMchpUsbClose    = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");

    HANDLE hDevice = INVALID_HANDLE_VALUE;

    UINT32 dwError;

    hDevice = libMchpUsbHCEOpen(0x424,0x2740);
    if(hDevice == INVALID_HANDLE_VALUE )
    {
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    printf("Device Opened successfulln");

    if (FALSE == libMchpUsbClose(hDevice))
    {
        dwError = libMchpUsbGetLastError(hDevice);
        printf ("Error,%04x",dwError);
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
}

```

7.2.1.5 MchpUsbClose Function

C

```

MCHP_USB_API BOOL MchpUsbClose(
    HANDLE DevID
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will close the handle for device specified in the call.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.(or)

[MchpUsbOpenID](#) should be called before calling this API

Parameters

Parameters	Description
DevID	Handle to the device - Return value of MchpUsbOpen or MchpUsbOpenID or MchpUsbHCEOpen .

Returns

TRUE - for Success;

FALSE - for Failure

Remarks

None

Example

```

typedef INT (*pfMchpUsbGetAllHubs) (PCHAR);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef UINT32 (*pfMchpUsbGetLastError) (HANDLE);

```

```

typedef HANDLE (*pfMchpUsbOpen) (CONST UINT);

pfMchpUsbGetAllHubs libMchpUsbGetAllHubs;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbGetLastErr libMchpUsbGetLastErr;
pfMchpUsbOpen libMchpUsbOpen;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
libMchpUsbGetAllHubs          = (pfMchpUsbGetAllHubs) GetProcAddress ( LoadLib
,"MchpUsbGetAllHubs");
libMchpUsbClose               = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");
libMchpUsbGetLastErr          = (pfMchpUsbGetLastErr) GetProcAddress ( LoadLib
,"MchpUsbGetLastErr");
libMchpUsbOpen                = (pfMchpUsbOpen) GetProcAddress ( LoadLib ,"MchpUsbOpen");

int hub_count = 0;
hub_count = libMchpUsbGetAllHubs(sztext);

if(hub_count)
{
    cout<<"Connected usb hubs are ..nn";
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< hubindex <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError = Error_Success;




hDevice = libMchpUsbOpen(hubindex);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastErr(HANDLE(hubindex));
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

if (FALSE == libMchpUsbClose(hDevice))
{
    dwError = libMchpUsbGetLastErr(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```


7.2.2 GPIO Bridging APIs

Functions

	Name	Description
	MchpUsbConfigureGPIO	This API configures the specified PIO line for general purpose input/output (GPIO)
	MchpUsbGpioGet	Get the state of the specified GPIO pin
	MchpUsbGpioSet	Set the state of the specified GPIO pin

Description

This APIs are used for 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.

7.2.2.1 MchpUsbConfigureGPIO Function

C

```
MCHP_USB_API BOOL MchpUsbConfigureGPIO(  
    HANDLE DevID,  
    INT PIONumber  
);
```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API configures the specified PIO line for general purpose input/output (GPIO).

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

PIN should be configured as GPIO mode before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
PIONumber	The GPIO pin number to be configured as GPIO mode.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Example

```

typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbConfigureGPIO) (HANDLE,INT);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbConfigureGPIO libMchpUsbConfigureGPIO;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");
libMchpUsbConfigureGPIO = (pfMchpUsbGpioGet) GetProcAddress ( LoadLib
,"MchpUsbConfigureGPIO");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");
//Configure pin number 11 as GPIO

if (FALSE ==libMchpUsbConfigureGPIO(hDevice,11))
{
    dwError = libMchpUsbGetLastError(hDevice);

    printf ("Failed to open the device Error,%04x",dwError);

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

```

7.2.2.2 MchpUsbGpioGet Function**C**

```

MCHP_USB_API BOOL MchpUsbGpioGet(
    HANDLE DevID,

```

```

    INT PIONumber,
    INT* Pinstate
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

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.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

PIN should be configured as GPIO mode before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
PIONumber	The GPIO pin number from which to read the pin state
Pinstate	1 = Pin state is High 0 = Pin state is Low

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

Example

```

typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbGpioGet) (HANDLE, INT, INT*);
typedef BOOL (*pfMchpUsbGpioSet) (HANDLE, INT, INT);
typedef BOOL (*pfMchpUsbConfigureGPIO) (HANDLE,INT);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbGpioGet libMchpUsbGpioGet;
pfMchpUsbGpioSet libMchpUsbGpioSet;
pfMchpUsbConfigureGPIO libMchpUsbConfigureGPIO;
CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
}

```

```

    exit (1);
}

libMchpUsbOpenID          = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose           = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");
libMchpUsbGpioGet         = (pfMchpUsbGpioGet) GetProcAddress ( LoadLib ,"MchpUsbGpioGet");
libMchpUsbGpioSet         = (pfMchpUsbGpioSet) GetProcAddress ( LoadLib
,"MchpUsbGpioSet");
libMchpUsbConfigureGPIO   = (pfMchpUsbConfigureGPIO) GetProcAddress ( LoadLib
,"MchpUsbConfigureGPIO");

HANDLE hDevice =  INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfullyn");
//Get status of pin number gpio 11
byData = 0x00;

//Configure pin number 11 as GPIO
if (FALSE ==libMchpUsbConfigureGPIO(hDevice,11))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to open the device Error,%04x",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

//GPIO set
int PIONumber=11;

int PINState=0;
if (FALSE == libMchpUsbGpioGet (hDevice,PIONumber,&PINState))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

7.2.2.3 MchpUsbGpioSet Function

C

```
MCHP_USB_API BOOL MchpUsbGpioSet(  
    HANDLE DevID,  
    INT PIONumber,  
    INT Pinstate  
);
```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

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.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

PIN should be configured as GPIO mode before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
PIONumber	The GPIO pin number from which to write the pin state
Pinstate	1 = Pin state is High 0 = Pin state is Low

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Example

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);  
typedef BOOL (*pfMchpUsbClose) (HANDLE);  
typedef BOOL (*pfMchpUsbGpioGet) (HANDLE, INT, INT*);  
typedef BOOL (*pfMchpUsbGpioSet) (HANDLE, INT, INT);  
typedef BOOL (*pfMchpUsbConfigureGPIO) (HANDLE,INT);  
  
pfMchpUsbOpenID libMchpUsbOpenID;  
pfMchpUsbClose libMchpUsbClose;  
pfMchpUsbGpioGet libMchpUsbGpioGet;  
pfMchpUsbGpioSet libMchpUsbGpioSet;  
pfMchpUsbConfigureGPIO libMchpUsbConfigureGPIO;  
  
CHAR sztext[2048];  
  
//Load library
```

```

HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID          = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
, "MchpUsbOpenID");
libMchpUsbClose           = (pfMchpUsbClose) GetProcAddress ( LoadLib , "MchpUsbClose");
libMchpUsbGpioGet         = (pfMchpUsbGpioGet) GetProcAddress ( LoadLib , "MchpUsbGpioGet");
libMchpUsbGpioSet         = (pfMchpUsbGpioSet) GetProcAddress ( LoadLib
, "MchpUsbGpioSet");
libMchpUsbConfigureGPIO   = (pfMchpUsbConfigureGPIO) GetProcAddress ( LoadLib
, "MchpUsbConfigureGPIO");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");
//Toggle PIONumber 11

//Configure pin number 11 as GPIO

if (FALSE == libMchpUsbConfigureGPIO(hDevice,11))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to open the device Error,%04x",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

//GPIO set

int PIONumber=11;

int PINState=1;

if (FALSE == libMchpUsbGpioSet (hDevice,PIONumber,PINState))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}



```

```
}

```

7.2.3 XDATA Bridging APIs

Functions

	Name	Description
	MchpUsbRegisterRead	Read the XDATA register(s) in the XDATA space of internal registers
	MchpUsbRegisterWrite	Write to the XDATA register (s) in the XDATA space of internal registers.

Description

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

7.2.3.1 MchpUsbRegisterRead Function

C

```
MCHP_USB_API BOOL MchpUsbRegisterRead(
    HANDLE DevID,
    UINT16 RegisterAddress,
    UINT16 BytesToRead,
    UINT8* InputData
);
```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API for Read the XDATA register(s) in the XDATA space of internal registers. This API also does the following: 1) Checks for the correct device handle before reading the registers 2) Checks for the proper buffer length

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device. Before calling this API ,the caller must acquire the device handle by calling appropriate API.
RegisterAddress	Start Address(in the XDATA space) from where Read operation starts
BytesToRead	Number of bytes to be read
InputData	Pointer to the buffer where data from XDATA registers will be stored.Caller must allocate memory for the buffer to accomodate the number of bytes to be read.

Returns

TRUE - for Success; FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

None

Example

```

typedef INT (*pfMchpUsbGetAllHubs) (PCHAR);
typedef HANDLE (*pfMchpUsbOpen) (CONST UINT);
typedef UINT32 (*pfMchpUsbGetLastError) (HANDLE);
typedef BOOL (*pfMchpUsbRegisterRead) (HANDLE, UINT16, UINT16, UINT8* );
typedef BOOL (*pfMchpUsbRegisterWrite) (HANDLE,UINT16,UINT16, UINT8* );

pfMchpUsbGetAllHubs libMchpUsbGetAllHubs;
pfMchpUsbOpen libMchpUsbOpen;
pfMchpUsbGetLastError libMchpUsbGetLastError;
pfMchpUsbRegisterRead libMchpUsbRegisterRead;
pfMchpUsbRegisterWrite libMchpUsbRegisterWrite;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
libMchpUsbGetAllHubs = (pfMchpUsbGetAllHubs) GetProcAddress ( LoadLib
,"MchpUsbGetAllHubs");
libMchpUsbOpen = (pfMchpUsbOpen) GetProcAddress ( LoadLib ,"MchpUsbOpen");
libMchpUsbGetLastError = (pfMchpUsbGetLastError) GetProcAddress ( LoadLib
,"MchpUsbGetLastError");
libMchpUsbRegisterRead = (pfMchpUsbRegisterRead) GetProcAddress ( LoadLib
,"MchpUsbRegisterRead");
libMchpUsbRegisterWrite = (pfMchpUsbRegisterWrite) GetProcAddress ( LoadLib
,"MchpUsbRegisterWrite");

int hub_count = 0;
hub_count = libMchpUsbGetAllHubs(sztext);

if(hub_count)
{
    cout<<"Connected usb hubs are ..nn";
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< hubindex <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError = Error_Success;

hDevice = libMchpUsbOpen(hubindex);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(HANDLE(hubindex));
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");

```



```

UINT8 byData = 0x55;
UINT16 wAddr = 0x4800;
printf("Xdata Write operation, Write 0x%02x in 0x%04xn",byData,wAddr);
if (FALSE ==libMchpUsbRegisterWrite(hDevice,wAddr,1,&byData))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to open the device Error,%04x",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Success :Xdata Write operationn";
byData = 0x00;

cout << "Xdata Read operationn";
if (FALSE ==libMchpUsbRegisterRead(hDevice,wAddr,1,&byData))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Success : Xdata Read operationn";
printf("Xdata Read value is %02x from 0x%04x n",byData,wAddr);

```

7.2.3.2 MchpUsbRegisterWrite Function

C

```

MCHP_USB_API BOOL MchpUsbRegisterWrite(
    HANDLE DevID,
    UINT16 RegisterAddress,
    UINT16 BytesToWrite,
    UINT8* OutputData
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API for Write the XDATA register(s) in the XDATA space of internal registers. This API also does the following: 1) Checks for the correct device handle before reading the registers 2) Checks for the proper buffer length

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device. Before calling this API ,the caller must acquire the device handle by calling appropriate API.
RegisterAddress	Start Address(in the XDATA space) from where Write operation starts
BytesToWrite	Number of bytes to be write
OutputData	Pointer to the buffer containing data to write to XDATA registers.

Returns

TRUE - for Success; FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

None

Example

```

typedef INT (*pfMchpUsbGetAllHubs) (PCHAR);
typedef HANDLE (*pfMchpUsbOpen) (CONST UINT);
typedef UINT32 (*pfMchpUsbGetLastError) (HANDLE);
typedef BOOL (*pfMchpUsbRegisterRead) (HANDLE, UINT16, UINT16, UINT8* );
typedef BOOL (*pfMchpUsbRegisterWrite) (HANDLE,UINT16, UINT16, UINT8* );

pfMchpUsbGetAllHubs libMchpUsbGetAllHubs;
pfMchpUsbOpen libMchpUsbOpen;
pfMchpUsbGetLastError libMchpUsbGetLastError;
pfMchpUsbRegisterRead libMchpUsbRegisterRead;
pfMchpUsbRegisterWrite libMchpUsbRegisterWrite;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
libMchpUsbGetAllHubs = (pfMchpUsbGetAllHubs) GetProcAddress ( LoadLib
,"MchpUsbGetAllHubs");
libMchpUsbOpen = (pfMchpUsbOpen) GetProcAddress ( LoadLib ,"MchpUsbOpen");
libMchpUsbGetLastError = (pfMchpUsbGetLastError) GetProcAddress ( LoadLib
,"MchpUsbGetLastError");
libMchpUsbRegisterRead = (pfMchpUsbRegisterRead) GetProcAddress ( LoadLib
,"MchpUsbRegisterRead");
libMchpUsbRegisterWrite = (pfMchpUsbRegisterWrite) GetProcAddress ( LoadLib
,"MchpUsbRegisterWrite");

int hub_count = 0;
hub_count = libMchpUsbGetAllHubs(sztext);

if(hub_count)
{
    cout<<"Connected usb hubs are ..nn";
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< hubindex <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError = Error_Success;

hDevice = libMchpUsbOpen(hubindex);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(HANDLE(hubindex));
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

```

}
printf("Device Opened successfully\n");

UINT8 byData = 0x55;
UINT16 wAddr = 0x4800;
printf("Xdata Write operation, Write 0x%02x in 0x%04xn",byData,wAddr);
if (FALSE ==libMchpUsbRegisterWrite(hDevice,wAddr,1,&byData))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to open the device Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Success :Xdata Write operationn";
byData = 0x00;

cout << "Xdata Read operationn";
if (FALSE ==libMchpUsbRegisterRead(hDevice,wAddr,1,&byData))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Success : Xdata Read operationn";
printf("Xdata Read value is %02x from 0x%04x n",byData,wAddr);

```

7.2.4 I2C Bridging APIs

Functions

	Name	Description
◆	MchpUsbI2CSetConfig	Set I2C config parameters
◆	MchpUsbI2CRead	I2C read through the I2C pass-through interface of USB device
◆	MchpUsbI2CWrite	I2C write through the I2C pass-through interface of USB device
◆	MchpUsbI2CTransfer	I2C read and write through the I2C pass-through interface of USB device

Description

Microchip USB hubs facilitate USB-I2C bridging through USB control point of the embedded USB device (5th port).

7.2.4.1 MchpUsbI2CSetConfig Function

C

```

MCHP_USB_API BOOL MchpUsbI2CSetConfig(
    HANDLE DevID,
    INT CRValue,
    INT nValue
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

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

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

MchpUsbOpenID should be called before calling this API

(or)

MchpUsbHCEOpen should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
CRValue	Clock Rate value of the I2C clock if nValue is zero.
nValue	For USB57x4 Family: 1 = 62.5Khz 2 = 235KHz 3 = 268KHz 4 = 312kHz 5 = 375KHz For USB253x/4604 Family: 1 = 62.5KHz Other values are Reserved. CRValue is dont care if nValue is nonzero.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

Example

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbI2CSetConfig) (HANDLE,INT, INT);
typedef BOOL (*pfMchpUsbI2CRead) (HANDLE, INT, UINT8*, UINT8);
typedef BOOL (*pfMchpUsbI2CWrite) (HANDLE, INT, UINT8*, UINT8);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbI2CSetConfig libMchpUsbI2CSetConfig;
pfMchpUsbI2CRead libMchpUsbI2CRead;
pfMchpUsbI2CWrite libMchpUsbI2CWrite;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");

libMchpUsbI2CSetConfig = (pfMchpUsbI2CSetConfig) GetProcAddress ( LoadLib
,"MchpUsbI2CSetConfig");
libMchpUsbI2CRead = (pfMchpUsbI2CRead) GetProcAddress ( LoadLib ,"MchpUsbI2CRead");
libMchpUsbI2CWrite =(pfMchpUsbI2CWrite) GetProcAddress ( LoadLib ,"MchpUsbI2CWrite");

HANDLE hDevice = INVALID_HANDLE_VALUE;
```

```

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfullyn");

//To Read EEPROM AT24C04
//Set desired value in clock
if (FALSE == libMchpUsbI2CSetConfig (hDevice, 30, 0))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to program SPI,%04x",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

7.2.4.2 MchpUsbI2CRead Function

C

```

MCHP_USB_API BOOL MchpUsbI2CRead(
    HANDLE DevID,
    INT BytesToRead,
    UINT8* InputData,
    UINT8 SlaveAddress
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

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

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
BytesToRead	Number of bytes to be read. Maximum value can be 512.
InputData	Pointer to the Buffer containing I2C read data
SlaveAddress	I2C Slave address

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

Example

```

typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbI2CSetConfig) (HANDLE,INT, INT);
typedef BOOL (*pfMchpUsbI2CRead) (HANDLE, INT, UINT8*, UINT8);
typedef BOOL (*pfMchpUsbI2CWrite) (HANDLE, INT, UINT8*, UINT8);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbI2CSetConfig libMchpUsbI2CSetConfig;
pfMchpUsbI2CRead libMchpUsbI2CRead;
pfMchpUsbI2CWrite libMchpUsbI2CWrite;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");

libMchpUsbI2CSetConfig = (pfMchpUsbI2CSetConfig) GetProcAddress ( LoadLib
,"MchpUsbI2CSetConfig");
libMchpUsbI2CRead = (pfMchpUsbI2CRead) GetProcAddress ( LoadLib ,"MchpUsbI2CRead");
libMchpUsbI2CWrite =(pfMchpUsbI2CWrite) GetProcAddress ( LoadLib ,"MchpUsbI2CWrite");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

//To Read EEPROM AT24C04
//Set desired value in clock
if (FALSE == libMchpUsbI2CSetConfig (hDevice, 30, 0))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to program SPI,%04x",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
//Write start address
UINT8 byAddr = 00;
if (FALSE == libMchpUsbI2CWrite (hDevice,1,&byAddr,0x50))
{

```

```

    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to program SPI,%04x",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
//Read 512 bytes
UINT8 byReadData[512];
if (FALSE == libMchpUsbI2CRead (hDevice,512,&byReadData[0],0x50))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to program SPI,%04x",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

7.2.4.3 MchpUsbI2CWrite Function

C

```

MCHP_USB_API BOOL MchpUsbI2CWrite(
    HANDLE DevID,
    INT BytesToWrite,
    UINT8* OutputData,
    UINT8 SlaveAddress
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

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

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
BytesToWrite	Number of bytes to be write. Maximum value can be 512.
OutputData	Pointer to the Buffer containing I2C data to be written
SlaveAddress	I2C Slave address

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

Example

```

typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);

```

```

typedef BOOL (*pfMchpUsbI2CSetConfig) (HANDLE,INT, INT);
typedef BOOL (*pfMchpUsbI2CRead) (HANDLE, INT, UINT8*, UINT8);
typedef BOOL (*pfMchpUsbI2CWrite) (HANDLE, INT, UINT8*, UINT8);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbI2CSetConfig libMchpUsbI2CSetConfig;
pfMchpUsbI2CRead libMchpUsbI2CRead;
pfMchpUsbI2CWrite libMchpUsbI2CWrite;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
, "MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib , "MchpUsbClose");

libMchpUsbI2CSetConfig = (pfMchpUsbI2CSetConfig) GetProcAddress ( LoadLib
, "MchpUsbI2CSetConfig");
libMchpUsbI2CRead = (pfMchpUsbI2CRead) GetProcAddress ( LoadLib , "MchpUsbI2CRead");
libMchpUsbI2CWrite = (pfMchpUsbI2CWrite) GetProcAddress ( LoadLib , "MchpUsbI2CWrite");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfully\n");

//To Write EEPROM AT24C04
//Set desired value in clock
if (FALSE == libMchpUsbI2CSetConfig (hDevice, 30, 0))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to program SPI,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
//Write 8 bytes
UINT8 bytobeWritten[9];

memset (bytobeWritten,0xaa,sizeof(bytobeWritten));

//Set start address
bytobeWritten[0] = 0x00;

if (FALSE == libMchpUsbI2CWrite (hDevice,9,&bytobeWritten,0x50))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to program SPI,%04xn",dwError);

```



```

    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

7.2.4.4 MchpUsbI2CTransfer Function

C

```

MCHP_USB_API BOOL MchpUsbI2CTransfer(
    HANDLE DevID,
    BOOL bDirection,
    UINT8* pbyBuffer,
    UINT16 wDataLen,
    UINT8 bySlaveAddress,
    BOOL bStart,
    BOOL bStop,
    BOOL bNack
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX,USB58XX/USB59XX

Description

This API performs an I2C read and write through the I2C pass-through interface of USB device.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
bDirection	0 : I2C Write 1 : I2C Read
pbyBuffer	I2C Write - Pointer to the buffer which contains the data to be sent over I2C I2C Read - Pointer to the buffer to which the data read from I2C will be stored.
DataLength	I2C Write - Number of bytes to write I2C Read - Number of bytes to read Maximum value can be 512 .
bySlaveAddress	Slave address of the I2C device
bStart	Indicates whether the start condition needs to be generated for this transfer, useful when combining single transfer in multiple API calls to handle large data. TRUE (Generates Start condition) FALSE(Does not generate Start condition)
bStop	Indicates whether the stop condition needs to be generated for this transfer, useful when combining single transfer in multiple API calls to handle large data. TRUE (Generates Stop condition) FALSE(Does not generate Stop condition)
bNack	Indicates whether the last byte should be NACK'ed for this transfer. TRUE (Generates NACK condition for the last byte of the transfer) FALSE(Does not generate NACK condition)

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks**Example**

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbI2CSetConfig) (HANDLE,INT, INT);
typedef BOOL (*pfMchpUsbI2CTransfer) (HANDLE, BOOL , UINT8*, UINT16, UINT8,BOOL,BOOL,BOOL);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbI2CSetConfig libMchpUsbI2CSetConfig;
pfMchpUsbI2CTransfer libMchpUsbI2CTransfer;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");

libMchpUsbI2CSetConfig = (pfMchpUsbI2CSetConfig) GetProcAddress ( LoadLib
,"MchpUsbI2CSetConfig");
libMchpUsbI2CTransfer = (pfMchpUsbI2CTransfer) GetProcAddress ( LoadLib
,"libMchpUsbI2CTransfer");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

//To Write EEPROM AT24C04
//Set desired value in clock
if (FALSE == libMchpUsbI2CSetConfig (hDevice, 30, 0))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to program SPI,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
```



```

}
//For i2c eeprom at24c04 ,read 10 bytes
UINT8 byData[512];
UINT8 byByteToWrite = 0x00; //Write address first
if(FALSE == libMchpUsbI2CTransfer(hDevice,0,&byByteToWrite,1,0x50,1,1,0))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to program SPI,%04x",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
//Read 10 bytes
if(FALSE == libMchpUsbI2CTransfer(hDevice,1,&byData[0],10,0x50,1,1,1))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Failed to program SPI,%04x",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

7.2.5 OTP Bridging APIs

Functions

	Name	Description
	MchpUsbOtpRead	This API performs read operation from OTP memory.
	MchpUsbOtpWrite	This API performs Write operation into OTP memory.

Description

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

7.2.5.1 MchpUsbOtpRead Function

C

```

MCHP_USB_API BOOL MchpUsbOtpRead(
    HANDLE DevID,
    UINT MemoryLocation,
    UINT BytesToRead,
    UINT8* OTPBuffer
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API reads bytes of data mentioned in the BytesToRead parameter from the OTP memory region of the device starting at address mentioned in the MemoryLocation parameter. Before Reading from OTP ,it will check for correct device Handle and Proper buffer length.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

MchpUsbHCEOpen should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
MemoryLocation	Start Address of the OTP from where read operation starts.
BytesToRead	Number of Bytes to be read. Maximum Bytes: USB253x/USB4604 - 2KB USB57X4 - 8KB
OTPCBuffer	Pointer to the Buffer which contains the read data to be stored.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

Example

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbOtpWrite) ( HANDLE , UINT, UINT8*);
typedef BOOL (*pfMchpUsbOtpRead) ( HANDLE , UINT, UINT , UINT8*);
```

```
pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbOtpWrite libMchpUsbOtpWrite;
pfMchpUsbOtpRead libMchpUsbOtpRead;
```

```
CHAR sztext[2048];
```

```
//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
```

```
libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");
libMchpUsbOtpWrite = (pfMchpUsbOtpWrite) GetProcAddress ( LoadLib ,"MchpUsbOtpWrite");
libMchpUsbOtpRead = (pfMchpUsbOtpRead) GetProcAddress ( LoadLib ,"MchpUsbOtpRead");
```

```
HANDLE hDevice = INVALID_HANDLE_VALUE;
```

```
UINT32 dwError;
```

```
hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
}
```

```

        exit (1);
    }
    printf("Device Opened successfully\n");

    UINT8 *byReadData = (UINT8 *) malloc(8*1024);

    if (byReadData == NULL)
    {
        printf ("Failed to open the device Error,%04x",dwError);

        printf ("nPress any key to exit....");

        _getch ();

        exit (1);
    }
    if (FALSE == libMchpUsbOtpRead(hDevice,0x0000,8*1024,(UINT8 *)byReadData))
    {
        dwError = libMchpUsbGetLastError(hDevice);

        printf ("Failed to Read the OTP,%04x",dwError);

        printf ("nPress any key to exit....");

        _getch ();

        exit (1);
    }
    if (byReadData)
    {
        free(byReadData);
        byReadData = NULL;
    }
}

```

7.2.5.2 MchpUsbOtpWrite Function

C

```

MCHP_USB_API BOOL MchpUsbOtpWrite(
    HANDLE DevID,
    UINT BytesToWrite,
    UINT8* ConfigData
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API writes bytes of data as mentioned in the BytesToWrite parameter to the OTP memory region . Before Writing to OTP ,it will check for correct device Handle and Proper buffer length.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device

BytesToWrite	Number of Bytes to be write. Maximum Bytes: USB253x/USB4604 - 2KB USB57X4 - 8KB
ConfigData	Pointer to the Buffer that contains the data to be written into OTP

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks**Example**

```

typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbOtpWrite) ( HANDLE , UINT, UINT8*);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbOtpWrite libMchpUsbOtpWrite;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");
libMchpUsbOtpWrite = (pfMchpUsbOtpWrite) GetProcAddress ( LoadLib ,"MchpUsbOtpWrite");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

//Config Data to change USB2 DID
UINT8 byConfigData[7] = {0x80,0x30,0x04,0x02,0x34,0x12,0xff};

if (FALSE == libMchpUsbOtpWrite(hDevice, sizeof (byConfigData),byConfigData[0]))
{
    dwError = libMchpUsbGetLastError(hDevice);

```

```

    printf ("Failed to Write the OTP Error,%04x",dwError);

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

```

7.2.6 SPI Bridging APIs

Functions

	Name	Description
≡	MchpUsbSpiSetConfig	This API enables/disables the SPI interface.
≡	MchpUsbSpiFlashRead	This API performs read operation from SPI Flash.
≡	MchpUsbSpiFlashWrite	This API performs write operation to SPI Flash memory.
≡	MchpUsbSpiTransfer	This API performs read/write operation to the SPI Interface.

Description

This section lists all the USB-SPI bridging APIs.

7.2.6.1 MchpUsbSpiSetConfig Function

C

```

MCHP_USB_API BOOL MchpUsbSpiSetConfig(
    HANDLE DevID,
    INT EnterExit
);

```

Devices Supported

USB4604,USB57X4,USB58XX/USB59XX

Description

This API enables/disables the SPI interface. 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 dependant totally on the strap options.

A INT variable EnterExit is used to identify if it is pass thru enter or exit.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
EnterExit	Pass thru Enter or exit option 1 : Pass thru Enter; 0 : Pass thru Exit;

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks**Example**

```

typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbSpiSetConfig) (HANDLE, INT);
typedef BOOL (*pfMchpUsbSpiFlashWrite)(HANDLE,UINT32 ,UINT8*, UINT32 );
typedef BOOL (*pfMchpUsbSpiFlashRead)(HANDLE,UINT32 ,UINT8*, UINT32 );
typedef BOOL (*pfMchpUsbSpiTransfer)(HANDLE,INT,UINT8*, UINT16,UINT32);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbSpiSetConfig libMchpUsbSpiSetConfig;
pfMchpUsbSpiFlashWrite libMchpUsbSpiFlashWrite;
pfMchpUsbSpiFlashRead libMchpUsbSpiFlashRead;
pfMchpUsbSpiTransfer libMchpUsbSpiTransfer;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");

libMchpUsbSpiFlashWrite =(pfMchpUsbSpiFlashWrite) GetProcAddress ( LoadLib
,"MchpUsbSpiFlashWrite");

libMchpUsbSpiFlashRead =(pfMchpUsbSpiFlashRead) GetProcAddress ( LoadLib
,"MchpUsbSpiFlashRead");
libMchpUsbSpiSetConfig = (pfMchpUsbSpiSetConfig) GetProcAddress ( LoadLib
,"MchpUsbSpiSetConfig");

libMchpUsbSpiTransfer = (pfMchpUsbSpiTransfer) GetProcAddress ( LoadLib
,"MchpUsbSpiTransfer");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
}

```



```

        exit (1);
    }
    printf("Device Opened successfull\n");
    //Enter into SPI Pass thru
    if (FALSE == libMchpUsbSpiSetConfig(hDevice,1))
    {
        printf ("MchpUsbSpiSetConfig failed");

        printf ("nPress any key to exit....");

        _getch ();

        exit (1);
    }

    //Exit into SPI Pass thru
    if (FALSE == libMchpUsbSpiSetConfig(hDevice,0))
    {
        printf ("MchpUsbSpiSetConfig failed");

        printf ("nPress any key to exit....");

        _getch ();

        exit (1);
    }
}

```

7.2.6.2 MchpUsbSpiFlashRead Function

C

```

MCHP_USB_API BOOL MchpUsbSpiFlashRead(
    HANDLE DevID,
    UINT32 StartAddr,
    UINT8* InputData,
    UINT32 BytesToRead
);

```

Devices Supported

USB4604,USB57X4,USB58XX/USB59XX

Description

This API reads bytes of data mentioned in the BytesToRead parameter from the SPI Flash memory region of the device starting at address mentioned in the StartAddr parameter. Before reading from SPI Flash, it will check for correct device Handle and Proper buffer length.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
StartAddr	Start Address of the SPI Flash from where read operation starts.
InputData	Pointer to the Buffer which contains the data to be read.
BytesToRead	No of Bytes to be read.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks**Example**

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbSpiSetConfig) (HANDLE, INT);
typedef BOOL (*pfMchpUsbSpiFlashWrite)(HANDLE,UINT32 ,UINT8*, UINT32 );
typedef BOOL (*pfMchpUsbSpiFlashRead)(HANDLE,UINT32 ,UINT8*, UINT32 );
typedef BOOL (*pfMchpUsbSpiTransfer)(HANDLE,INT,UINT8*, UINT16,UINT32);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbSpiSetConfig libMchpUsbSpiSetConfig;
pfMchpUsbSpiFlashWrite libMchpUsbSpiFlashWrite;
pfMchpUsbSpiFlashRead libMchpUsbSpiFlashRead;
pfMchpUsbSpiTransfer libMchpUsbSpiTransfer;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");

libMchpUsbSpiFlashWrite =(pfMchpUsbSpiFlashWrite) GetProcAddress ( LoadLib
,"MchpUsbSpiFlashWrite");

libMchpUsbSpiFlashRead =(pfMchpUsbSpiFlashRead) GetProcAddress ( LoadLib
,"MchpUsbSpiFlashRead");
libMchpUsbSpiSetConfig = (pfMchpUsbSpiSetConfig) GetProcAddress ( LoadLib
,"MchpUsbSpiSetConfig");

libMchpUsbSpiTransfer = (pfMchpUsbSpiTransfer) GetProcAddress ( LoadLib
,"MchpUsbSpiTransfer");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
}
```

```

        exit (1);
    }
    printf("Device Opened successfully\n");

    //Write SPI Flash
    HANDLE hBinFile;
    UINT8 byFirmwareData[64*1024];
    UINT8 byReadFirmwareData[64*1024];
    UINT32 dwActualSize = 0;

    hBinFile = CreateFile ("spi_firmware.bin", GENERIC_READ, FILE_SHARE_READ,
        NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

    if (INVALID_HANDLE_VALUE == hBinFile)
    {
        exit(1);
    }

    // Allow maxm_fw_size or dwActualsize whichever is small
    if (FALSE == ReadFile (hBinFile, &byFirmwareData[0], sizeof(byFirmwareData) ,(LPDWORD)
        &dwActualSize, NULL))
    {
        CloseHandle (hBinFile);
    }

    CloseHandle (hBinFile);
    if (FALSE == libMchpUsbSpiFlashWrite(hDevice,0x0,&byFirmwareData[0],dwActualSize))
    {
        dwError = libMchpUsbGetLastError(hDevice);

        printf ("Failed to Write SPI Error,%04x",dwError);

        printf ("nPress any key to exit....");

        _getch ();

        exit (1);
    }
    if (FALSE == libMchpUsbSpiFlashRead(hDevice,0x0,&byReadFirmwareData[0],dwActualSize))
    {
        dwError = libMchpUsbGetLastError(hDevice);

        printf ("Failed to Write SPI Error,%04x",dwError);

        printf ("nPress any key to exit....");

        _getch ();

        exit (1);
    }
    if (memcmp(byReadFirmwareData,byFirmwareData,10))
    {
        printf ("Read and write comparison failed");

        printf ("nPress any key to exit....");

        _getch ();

        exit (1);
    }
}

```

7.2.6.3 MchpUsbSpiFlashWrite Function

C

```

MCHP_USB_API BOOL MchpUsbSpiFlashWrite(
    HANDLE DevID,
    UINT32 StartAddr,

```

```

    UINT8* OutputData,
    UINT32 BytesToWrite
);

```

Devices Supported

USB4604,USB57X4,USB58XX/USB59XX

Description

This API writes bytes of data as mentioned in the BytesToWrite parameter to the SPI Flash memory region from memory location as specified in StartAddr. Before Writing to SPI Flash,it will check for correct device Handle and Proper buffer length.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
StartAddr	Start Address of the SPI Flash from where write operation starts.
OutputData	Pointer to the Buffer which contains the data to be written.
BytesToWrite	No of Bytes to be written.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

Example

```

typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbSpiSetConfig) (HANDLE, INT);
typedef BOOL (*pfMchpUsbSpiFlashWrite)(HANDLE,UINT32 ,UINT8*, UINT32 );
typedef BOOL (*pfMchpUsbSpiFlashRead)(HANDLE,UINT32 ,UINT8*, UINT32 );
typedef BOOL (*pfMchpUsbSpiTransfer)(HANDLE,INT,UINT8*, UINT16,UINT32);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbSpiSetConfig libMchpUsbSpiSetConfig;
pfMchpUsbSpiFlashWrite libMchpUsbSpiFlashWrite;
pfMchpUsbSpiFlashRead libMchpUsbSpiFlashRead;
pfMchpUsbSpiTransfer libMchpUsbSpiTransfer;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loadedn");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
}

```

```

        exit (1);
    }

    libMchpUsbOpenID          = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
, "MchpUsbOpenID");
    libMchpUsbClose           = (pfMchpUsbClose) GetProcAddress ( LoadLib , "MchpUsbClose");

    libMchpUsbSpiFlashWrite = (pfMchpUsbSpiFlashWrite) GetProcAddress ( LoadLib
, "MchpUsbSpiFlashWrite");

    libMchpUsbSpiFlashRead = (pfMchpUsbSpiFlashRead) GetProcAddress ( LoadLib
, "MchpUsbSpiFlashRead");
    libMchpUsbSpiSetConfig = (pfMchpUsbSpiSetConfig) GetProcAddress ( LoadLib
, "MchpUsbSpiSetConfig");

    libMchpUsbSpiTransfer = (pfMchpUsbSpiTransfer) GetProcAddress ( LoadLib
, "MchpUsbSpiTransfer");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");
//Write SPI Flash
HANDLE hBinFile;
UINT8 byFirmwareData[64*1024];
UINT8 byReadFirmwareData[64*1024];
UINT32 dwActualSize = 0;

hBinFile = CreateFile ("spi_firmware.bin", GENERIC_READ, FILE_SHARE_READ,
NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

if (INVALID_HANDLE_VALUE == hBinFile)
{
    exit(1);
}

// Allow maxm_fw_size or dwActualsize whichever is small
if (FALSE == ReadFile (hBinFile, &byFirmwareData[0], sizeof(byFirmwareData) ,(LPDWORD)
&dwActualSize, NULL))
{
    CloseHandle (hBinFile);
}

CloseHandle (hBinFile);
if (FALSE == libMchpUsbSpiFlashWrite(hDevice,0x0,&byFirmwareData[0],dwActualSize))
{
    dwError = libMchpUsbGetLastError(hDevice);

    printf ("Failed to Write SPI Error,%04x",dwError);

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
if (FALSE == libMchpUsbSpiFlashRead(hDevice,0x0,&byReadFirmwareData[0],dwActualSize))
{

```

```
dwError = libMchpUsbGetLastError(hDevice);  
printf ("Failed to Write SPI Error,%04x",dwError);  
printf ("nPress any key to exit....");  
_getch ();  
exit (1);  
}  
if (memcmp(byReadFirmwareData,byFirmwareData,10))  
{  
    printf ("Read and write comparison failed");  
    printf ("nPress any key to exit....");  
    _getch ();  
    exit (1);  
}
```

7.2.6.4 MchpUsbSpiTransfer Function

C

```
MCHP_USB_API BOOL MchpUsbSpiTransfer(  
    HANDLE DevID,  
    INT Direction,  
    UINT8* Buffer,  
    UINT16 DataLength,  
    UINT32 TotalLength  
);
```

Devices Supported

USB4604,USB57X4,USB58XX/USB59XX

Description

This API is the low level SPI pass thru command read/write. All commands to the SPI interface are directed as SPI Pass thru write, SPI pass thru read is nothing but a XDATA read from a specified offset where the response is stored.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
Direction	This bit will indicate if it is a Pass thru read or write. Read = 1; Write = 0.
Buffer	Buffer containing the command/ data to be sent to the device in case of SPI pass thru write. In case of pass thru read this buffer is used to store the data recieved from the device.
DataLength	This field is the size of USB command OUT packet being sent to the firmware.
wTotalLength	The wTotalLength is utilized to mention the number of bytes the SPI flash will return for the pass thru command.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

SPI pass thru enter should be done before executing this API.

Example

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbSpiSetConfig) (HANDLE, INT);
typedef BOOL (*pfMchpUsbSpiFlashWrite)(HANDLE,UINT32 ,UINT8*, UINT32 );
typedef BOOL (*pfMchpUsbSpiFlashRead)(HANDLE,UINT32 ,UINT8*, UINT32 );
typedef BOOL (*pfMchpUsbSpiTransfer)(HANDLE,INT,UINT8*, UINT16,UINT32);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbSpiSetConfig libMchpUsbSpiSetConfig;
pfMchpUsbSpiFlashWrite libMchpUsbSpiFlashWrite;
pfMchpUsbSpiFlashRead libMchpUsbSpiFlashRead;
pfMchpUsbSpiTransfer libMchpUsbSpiTransfer;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");

libMchpUsbSpiFlashWrite =(pfMchpUsbSpiFlashWrite) GetProcAddress ( LoadLib
,"MchpUsbSpiFlashWrite");

libMchpUsbSpiFlashRead =(pfMchpUsbSpiFlashRead) GetProcAddress ( LoadLib
,"MchpUsbSpiFlashRead");
libMchpUsbSpiSetConfig = (pfMchpUsbSpiSetConfig) GetProcAddress ( LoadLib
,"MchpUsbSpiSetConfig");

libMchpUsbSpiTransfer = (pfMchpUsbSpiTransfer) GetProcAddress ( LoadLib
,"MchpUsbSpiTransfer");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");
//Enter into SPI Pass thru
```

```

if (FALSE == libMchpUsbSpiSetConfig(hDevice,1))
{
    printf ("MchpUsbSpiSetConfig failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
UINT8 bySPIBuffer[4];
UINT8 byOpcodeGetJEDECID = 0x9f;
//Write 0x9f to get JEDEC ID, Datalen is 1
//Totally 4 bytes will be retrived as jedec id, give total length as 4
if(FALSE == libMchpUsbSpiTransfer(hDevice,0,&byOpcodeGetJEDECID,1,4))
{
    printf ("MchpUsbSpiTransfer failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
//Read 4 bytes of JEDEC ID
if(FALSE == libMchpUsbSpiTransfer(hDevice,1,&bySPIBuffer[0],4,4))
{
    printf ("MchpUsbSpiTransfer failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
//exit from spi config
if (FALSE == libMchpUsbSpiSetConfig(hDevice,0))
{
    printf ("MchpUsbSpiSetConfig failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

```

7.2.7 UART Bridging APIs

Functions

	Name	Description
≡	MchpUsbEnableUARTBridging	This API enables the UART device for serial communication.
≡	MchpUsbSetUARTBaudrate	This API configures the baud rate for serial communication.
≡	MchpUsbUartRead	This API synchronously receives data through serial port from the connected serial peripheral.
≡	MchpUsbUartWrite	This API transfers data through serial port to the connected serial peripheral.
≡	MchpUsbUartReadTimeOut	This API synchronously receives data through serial port from the connected serial peripheral for a given Time period.

Description

7.2.7.1 MchpUsbEnableUARTBridging Function

C

```
MCHP_USB_API BOOL MchpUsbEnableUARTBridging(
    HANDLE DevID,
    BOOL bEnable
);
```

Devices Supported

USB253X/USB4604

Description

This API enables the UART device for serial communication.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
bEnable	TRUE - Enable UART bridging, FALSE - Disable UART bridging

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Example

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbUartRead) (HANDLE, UINT32,UINT8* );
typedef BOOL (*pfMchpUsbUartWrite) (HANDLE ,UINT32, UINT8 *);
typedef BOOL (*pfMchpUsbSetUARTBaudrate)(HANDLE, UINT32 );
typedef BOOL (*pfMchpUsbEnableUARTBridging) (HANDLE , BOOL );

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbUartRead libMchpUsbUartRead;
pfMchpUsbUartWrite libMchpUsbUartWrite;
pfMchpUsbSetUARTBaudrate libMchpUsbSetUARTBaudrate;
pfMchpUsbEnableUARTBridging libMchpUsbEnableUARTBridging;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
```

```

    {
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }

    libMchpUsbOpenID          = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
, "MchpUsbOpenID");
    libMchpUsbClose           = (pfMchpUsbClose) GetProcAddress ( LoadLib , "MchpUsbClose");

    libMchpUsbUartRead = (pfMchpUsbUartRead) GetProcAddress ( LoadLib , "MchpUsbUartRead");
    libMchpUsbUartWrite = (pfMchpUsbUartWrite ) GetProcAddress ( LoadLib
, "MchpUsbUartWrite");
    libMchpUsbSetUARTBaudrate = (pfMchpUsbSetUARTBaudrate) GetProcAddress ( LoadLib
, "MchpUsbSetUARTBaudrate");
    libMchpUsbEnableUARTBridging = (pfMchpUsbEnableUARTBridging) GetProcAddress ( LoadLib
, "MchpUsbEnableUARTBridging");

    HANDLE hDevice =  INVALID_HANDLE_VALUE;

    UINT32 dwError;

    hDevice = libMchpUsbOpenID(0x424, 0x1234);
    if(INVALID_HANDLE_VALUE == hDevice)
    {
        dwError = libMchpUsbGetLastError(hDevice);
        printf ("Error,%04xn",dwError);
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    printf("Device Opened successfull\n");

    if (FALSE == libMchpUsbEnableUARTBridging(hDevice,TRUE))
    {
        printf ("Uart Bridge failed");

        printf ("nPress any key to exit....");

        _getch ();

        exit (1);
    }
}

```

7.2.7.2 MchpUsbSetUARTBaudrate Function

C

```

MCHP_USB_API BOOL MchpUsbSetUARTBaudrate(
    HANDLE DevID,
    UINT32 BaudRate
);

```

Devices Supported

USB253X/USB4604

Description

This API configures the baud rate for serial communication.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

MchpUsbHCEOpen should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
BaudRate	Baud rate to be set. Suggested standard values are 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

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.

Example

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbUartRead) (HANDLE, UINT32,UINT8* );
typedef BOOL (*pfMchpUsbUartWrite) (HANDLE ,UINT32, UINT8 *);
typedef BOOL (*pfMchpUsbSetUARTBaudrate)(HANDLE, UINT32 );
typedef BOOL (*pfMchpUsbEnableUARTBridging) (HANDLE , BOOL );

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbUartRead libMchpUsbUartRead;
pfMchpUsbUartWrite libMchpUsbUartWrite;
pfMchpUsbSetUARTBaudrate libMchpUsbSetUARTBaudrate;
pfMchpUsbEnableUARTBridging libMchpUsbEnableUARTBridging;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");

libMchpUsbUartRead = (pfMchpUsbUartRead) GetProcAddress ( LoadLib ,"MchpUsbUartRead");
libMchpUsbUartWrite = (pfMchpUsbUartWrite ) GetProcAddress ( LoadLib
,"MchpUsbUartWrite");
libMchpUsbSetUARTBaudrate = (pfMchpUsbSetUARTBaudrate) GetProcAddress ( LoadLib
,"MchpUsbSetUARTBaudrate");
libMchpUsbEnableUARTBridging = (pfMchpUsbEnableUARTBridging) GetProcAddress ( LoadLib
,"MchpUsbEnableUARTBridging");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
```

```

{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

if (FALSE == libMchpUsbEnableUARTBridging(hDevice,TRUE))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
if (FALSE == libMchpUsbSetUARTBaudrate(hDevice,9600))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
if (FALSE == libMchpUsbEnableUARTBridging(hDevice,FALSE))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
}

```

7.2.7.3 MchpUsbUartRead Function

C

```

MCHP_USB_API BOOL MchpUsbUartRead(
    HANDLE DevID,
    UINT32 BytesToRead,
    UINT8 * InputData
);

```

Devices Supported

USB253X/USB4604

Description

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

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
InputData	Pointer to input data buffer which contains the data to transfer
BytesToRead	Length of bytes to transfer to the serial port

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

Set Baud rate using [MchpUsbSetUARTBaudrate](#) 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 ReceiveData buffer as mentioned in the dwReceiveLength parameter

Example

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbUartRead) (HANDLE, UINT32,UINT8* );
typedef BOOL (*pfMchpUsbUartWrite) (HANDLE ,UINT32, UINT8 *);
typedef BOOL (*pfMchpUsbSetUARTBaudrate)(HANDLE, UINT32 );
typedef BOOL (*pfMchpUsbEnableUARTBridging) (HANDLE , BOOL );
```

```
pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbUartRead libMchpUsbUartRead;
pfMchpUsbUartWrite libMchpUsbUartWrite;
pfMchpUsbSetUARTBaudrate libMchpUsbSetUARTBaudrate;
pfMchpUsbEnableUARTBridging libMchpUsbEnableUARTBridging;
```

```
CHAR sztext[2048];
```

```
//Load library
```

```
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
```

```
if (LoadLib != NULL)
```

```
{
    printf("Loadlib library Loaded\n");
}
```

```
else
```

```
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
```

```
libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");

libMchpUsbUartRead = (pfMchpUsbUartRead) GetProcAddress ( LoadLib ,"MchpUsbUartRead");
libMchpUsbUartWrite = (pfMchpUsbUartWrite ) GetProcAddress ( LoadLib
,"MchpUsbUartWrite");
libMchpUsbSetUARTBaudrate = (pfMchpUsbSetUARTBaudrate) GetProcAddress ( LoadLib
,"MchpUsbSetUARTBaudrate");
libMchpUsbEnableUARTBridging = (pfMchpUsbEnableUARTBridging) GetProcAddress ( LoadLib
,"MchpUsbEnableUARTBridging");
```

```
HANDLE hDevice = INVALID_HANDLE_VALUE;
```

```
UINT32 dwError;
```

```

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

if (FALSE == libMchpUsbEnableUARTBridging(hDevice,TRUE))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
if (FALSE == libMchpUsbSetUARTBaudrate(hDevice,9600))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
UINT8 byDataUART [4] = {0x60,0x61,0x62,0x64};
if (FALSE == libMchpUsbUartWrite(hDevice,4,&byDataUART[0]))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

if (FALSE == libMchpUsbUartRead(hDevice,4,&byDataUART[0]))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

if (FALSE == libMchpUsbEnableUARTBridging(hDevice,FALSE))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

```

7.2.7.4 MchpUsbUartWrite Function

C

MCHP_USB_API BOOL **MchpUsbUartWrite**(

```

HANDLE DevID,
UINT32 BytesToWrite,
UINT8 * OutputData
);

```

Devices Supported

USB253X/USB4604

Description

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

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
OutputData	Pointer to output data buffer which contains the data to transfer
BytesToWrite	Length of bytes to transfer to the serial port

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Set Baud rate using [MchpUsbSetUARTBaudrate](#) API before calling this API.

Example

```

typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbUartRead) (HANDLE, UINT32,UINT8* );
typedef BOOL (*pfMchpUsbUartWrite) (HANDLE ,UINT32, UINT8 *);
typedef BOOL (*pfMchpUsbSetUARTBaudrate)(HANDLE, UINT32 );
typedef BOOL (*pfMchpUsbEnableUARTBridging) (HANDLE , BOOL );

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbUartRead libMchpUsbUartRead;
pfMchpUsbUartWrite libMchpUsbUartWrite;
pfMchpUsbSetUARTBaudrate libMchpUsbSetUARTBaudrate;
pfMchpUsbEnableUARTBridging libMchpUsbEnableUARTBridging;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loadedn");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

```

    libMchpUsbOpenID          = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
, "MchpUsbOpenID");
    libMchpUsbClose           = (pfMchpUsbClose) GetProcAddress ( LoadLib , "MchpUsbClose");

    libMchpUsbUartRead = (pfMchpUsbUartRead) GetProcAddress ( LoadLib , "MchpUsbUartRead");
    libMchpUsbUartWrite = (pfMchpUsbUartWrite ) GetProcAddress ( LoadLib
, "MchpUsbUartWrite");
    libMchpUsbSetUARTBaudrate = (pfMchpUsbSetUARTBaudrate) GetProcAddress ( LoadLib
, "MchpUsbSetUARTBaudrate");
    libMchpUsbEnableUARTBridging = (pfMchpUsbEnableUARTBridging) GetProcAddress ( LoadLib
, "MchpUsbEnableUARTBridging");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");

if (FALSE == libMchpUsbEnableUARTBridging(hDevice,TRUE))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
if (FALSE == libMchpUsbSetUARTBaudrate(hDevice,9600))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
UINT8 byDataUART [4] = {0x60,0x61,0x62,0x64};
if (FALSE == libMchpUsbUartWrite(hDevice,4,&byDataUART[0]))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

if (FALSE == libMchpUsbUartRead(hDevice,4,&byDataUART[0]))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

```



```

if (FALSE == libMchpUsbEnableUARTBridging(hDevice,FALSE))
{
    printf ("Uart Bridge failed");

    printf ("\nPress any key to exit....");

    _getch ();

    exit (1);
}

```

7.2.7.5 MchpUsbUartReadTimeOut Function

C

```

MCHP_USB_API BOOL MchpUsbUartReadTimeOut(
    HANDLE DevID,
    UINT32 BytesToRead,
    UINT8 * InputData,
    UINT32 Timeout
);

```

Devices Supported

USB253X/USB4604

Description

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

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
InputData	Pointer to input data buffer which contains the data to transfer
BytesToRead	Length of bytes to transfer to the serial port
TimeOut	Timeout interval, in milliseconds. Function waits until data is received or the time period elapses 0 -does not wait if the data is not received INFINITE - wait until data is received

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Set Baud rate using [MchpUsbSetUARTBaudrate](#) 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 ReceiveData buffer as mentioned in the dwReceiveLength parameter

Example

```

typedef HANDLE (*pMchpUsbOpenID) (UINT16,UINT16);

```

```

typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*MchpUsbUartReadTimeOut) (HANDLE, UINT32,UINT8*,UINT32 );
typedef BOOL (*pfMchpUsbUartWrite) (HANDLE ,UINT32, UINT8 *);
typedef BOOL (*pfMchpUsbSetUARTBaudrate)(HANDLE, UINT32 );
typedef BOOL (*pfMchpUsbEnableUARTBridging) (HANDLE , BOOL );

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbUartReadTimeOut libMchpUsbUartReadTimeOut;
pfMchpUsbUartWrite libMchpUsbUartWrite;
pfMchpUsbSetUARTBaudrate libMchpUsbSetUARTBaudrate;
pfMchpUsbEnableUARTBridging libMchpUsbEnableUARTBridging;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");

libMchpUsbUartReadTimeOut = (pfMchpUsbUartReadTimeOut) GetProcAddress ( LoadLib
,"MchpUsbUartReadTimeOut");
libMchpUsbUartWrite = (pfMchpUsbUartWrite ) GetProcAddress ( LoadLib
,"MchpUsbUartWrite");
libMchpUsbSetUARTBaudrate = (pfMchpUsbSetUARTBaudrate) GetProcAddress ( LoadLib
,"MchpUsbSetUARTBaudrate");
libMchpUsbEnableUARTBridging = (pfMchpUsbEnableUARTBridging) GetProcAddress ( LoadLib
,"MchpUsbEnableUARTBridging");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfully\n");

if (FALSE == libMchpUsbEnableUARTBridging(hDevice,TRUE))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
if (FALSE == libMchpUsbSetUARTBaudrate(hDevice,9600))
{
    printf ("Uart Bridge failed");
}

```

```

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
UINT8 byDataUART [4] = {0x60,0x61,0x62,0x64};
if (FALSE == libMchpUsbUartWrite(hDevice,4,&byDataUART[0]))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
//waits for 5000 milliseconds
if (FALSE == libMchpUsbUartReadTimeOut(hDevice,4,byDataUART[0],5000))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

if (FALSE == libMchpUsbEnableUARTBridging(hDevice,FALSE))
{
    printf ("Uart Bridge failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

```

7.2.8 Flexconnect API

Functions

	Name	Description
◆	MchpUsbFlexConnect	This API is used to send the Flexconnect command to device.
◆	MchpUsbWriteFlexConfigArea	This API will get the list of LAN devices connected to the system

Description

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

7.2.8.1 MchpUsbFlexConnect Function

C

```

MCHP_USB_API BOOL MchpUsbFlexConnect(
    HANDLE DevID,
    UINT16 Config
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API is used to send the Flexconnect Command to device with config data as specified in the Config variable.

This Config value is based on the Product, please refer Product Specification for more details.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
Config	Passed as wValue field of the Flexconnect SETUP Command.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks**Example**

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbFlexConnect) (HANDLE, UINT16);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;

pfMchpUsbFlexConnect libMchpUsbFlexConnect;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
, "MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib , "MchpUsbClose");

libMchpUsbFlexConnect = (pfMchpUsbFlexConnect) GetProcAddress ( LoadLib
, "MchpUsbFlexConnect");

HANDLE hDevice = INVALID_HANDLE_VALUE;
```

```

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");
//To turn on Flexconnect with Port 2 & 4 disabled
//wValue 0x8454 DIS_P2 = 1 ; Disable Port 2
//DIS_P4= 1 : Disable Port 4
//FLEX_STATE = 1 : Enable Flexconnect
//HDD TMR = 100b : Timer 1 second
//Bit 15 = 1
if (FALSE == libMchpUsbFlexConnect(hDevice,0x8454))
{
    printf ("MchpUsbFlexConnect failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

```

7.2.8.2 MchpUsbWriteFlexConfigArea Function

C

```

MCHP_USB_API BOOL MchpUsbWriteFlexConfigArea(
    BYTE DevID,
    UINT8* FlexConfigData,
    INT Configdatalen,
    BOOL SetFlexReg
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will return number of LAN devices connected to the system. This API acts as a Global INIT function and hence should be called before any of the functions that access the hub_index. If there is any LAN devices insertion or removal after calling this API, then the user have to take care of calling this function again. Also the user should check if the LAN Adapter index is correct if more than 1 LAN devices are present in the system.

Preconditions

By default, the log file will not be created. so [MchpEnableLogging](#) API must be called to create the log file(PT2.log) before calling this API.

Parameters

Parameters	Description
LanInfo	Pointer to the buffer which has minimal information about LAN devices with index.

Returns

No. of LAN devices connected to system

Remarks

None.

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef void (*pfMchpEnableLogging) (INT);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loadedn");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices          = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpEnableLogging          = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

7.2.9 Programming APIs

Functions

	Name	Description
≡	MchpProgramFile	Program configuration file to the selected device ID
≡	MchpProgramSPIFile	Program SPI firmware
≡	MchpProgramFileWithSerial	Program configuration file with serial number to the selected device ID
≡	MchpUsbProgramFileWithUsb3Serial	This API will program configuration file along with serial numbers.
≡	MchpProgramSPIFirmwareWithConfig	This API will program firmware file along with configuration file.
≡	MchpProgramSpiFwCfgFileWithSerial	This API will program firmware file as specified in FirmwareFile argument along with the configuration file as specified in ConfigFile argument and Serial Number specified in SerialNumber argument
≡	MchpProgramSPIFirmwareWithConfigBuffer	This API will program SPI firmware bytes along with configuration bytes.

Description

This section lists all high level APIs which can be used for programming.

7.2.9.1 MchpProgramFile Function

C

```
MCHP_USB_API BOOL MchpProgramFile(  
    HANDLE DevID,  
    PCHAR InputFileName  
);
```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will program the configuration file given as argument to the selected device ID.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
InputFileName	Input configuration file to be programmed into the device

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

None

Example

```
typedef INT (*pfMchpUsbGetAllHubs) (PCHAR);  
typedef HANDLE (*pfMchpUsbOpen) (CONST UINT);  
typedef UINT32 (*pfMchpUsbGetLastError) (HANDLE);  
typedef BOOL (*pfMchpProgramFile) (HANDLE, PCHAR);
```

```
pfMchpUsbGetAllHubs libMchpUsbGetAllHubs;  
pfMchpUsbOpen libMchpUsbOpen;  
pfMchpUsbGetLastError libMchpUsbGetLastError;  
pfMchpProgramFile libMchpProgramFile;
```

```
CHAR sztext[2048];
```

```
//Load library  
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");  
if (LoadLib != NULL)  
{  
    printf("Loadlib library Loadedn");  
}  
else  
{
```

```

    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
libMchpUsbGetAllHubs          = (pfMchpUsbGetAllHubs) GetProcAddress ( LoadLib
,"MchpUsbGetAllHubs");
libMchpUsbOpen                = (pfMchpUsbOpen) GetProcAddress ( LoadLib ,"MchpUsbOpen");
libMchpUsbGetLastError        = (pfMchpUsbGetLastError) GetProcAddress ( LoadLib
,"MchpUsbGetLastError");
libMchpProgramFile            = (pfMchpProgramFile)  GetProcAddress ( LoadLib
,"MchpProgramFile");

int hub_count = 0;
hub_count = libMchpUsbGetAllHubs(sztext);

if(hub_count)
{
    cout<<"Connected usb hubs are ..nn";
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< hubindex <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError = Error_Success;

hDevice = libMchpUsbOpen(hubindex);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(HANDLE(hubindex));
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");

if (FALSE == libMchpProgramFile(hDevice,"CONFIG_FILE.cfg"))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Device Program Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Device Programmed Successfulln";

```

7.2.9.2 MchpProgramSPIFile Function

C

```

MCHP_USB_API BOOL MchpProgramSPIFile(
    HANDLE DevID,
    PCHAR InputFileName,
    BOOL EraseConfig
);

```

Devices Supported

USB4604,USB57X4,USB58XX/USB59XX

Description

This API will program firmware to spi memory on the selected device ID.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
InputFileName	Input bin file to be programmed into the device
EraseConfig	TRUE - Erase all programmed configuration data in spi memory FALSE - Do not erase programmed configuration data in spi memory

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

None

Example

```
typedef INT (*pfMchpUsbGetAllHubs) (PCHAR);
typedef HANDLE (*pfMchpUsbOpen) (CONST UINT);
typedef UINT32 (*pfMchpUsbGetLastError) (HANDLE);
typedef BOOL (*pfMchpProgramSPIFile) (HANDLE,PCHAR,BOOL);

pfMchpUsbGetAllHubs libMchpUsbGetAllHubs;
pfMchpUsbOpen libMchpUsbOpen;
pfMchpUsbGetLastError libMchpUsbGetLastError;
pfMchpProgramSPIFile libMchpProgramSPIFile;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
libMchpUsbGetAllHubs = (pfMchpUsbGetAllHubs) GetProcAddress ( LoadLib
,"MchpUsbGetAllHubs");
libMchpUsbOpen = (pfMchpUsbOpen) GetProcAddress ( LoadLib ,"MchpUsbOpen");
libMchpUsbGetLastError = (pfMchpUsbGetLastError) GetProcAddress ( LoadLib
,"MchpUsbGetLastError");
libMchpProgramSPIFile = (pfMchpProgramSPIFile) GetProcAddress ( LoadLib
,"MchpProgramSPIFile");

int hub_count = 0;
hub_count = libMchpUsbGetAllHubs(sztext);

if(hub_count)
{
    cout<<"Connected usb hubs are ..nn";
}
```

```

        cout<<sztext<<endl;
        cout<<"Selected hub_index is "<< hubindex <<endl;
    }
    else
    {
        printf ("zero hubs foundn");
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    HANDLE hDevice =  INVALID_HANDLE_VALUE;

    UINT32 dwError = Error_Success;

    hDevice = libMchpUsbOpen(hubindex);
    if(INVALID_HANDLE_VALUE == hDevice)
    {
        dwError = libMchpUsbGetLastError(HANDLE(hubindex));
        printf ("Error,%04xn",dwError);
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    printf("Device Opened successfulln");

    if (FALSE ==libMchpProgramSPIFile(hDevice,"spi_firmware.bin",TRUE))
    {
        dwError= libMchpUsbGetLastError(hDevice);
        printf ("Failed to program SPI,%04x",dwError);
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    Sleep(5000);
    printf("SPI programming passedn");

```

7.2.9.3 MchpProgramFileWithSerial Function

C

```

MCHP_USB_API BOOL MchpProgramFileWithSerial(
    HANDLE DevID,
    PCHAR InputFileName,
    PCHAR SerialNumber
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will program configuration file along with the serial number given as argument to the selected device ID.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device

InputFileName	Input configuration file to be programmed into the device. This argument is optional. NULL can be passed to this argument if input file is not given.
SerialNumber	Serial Number to be programmed

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

For USB57X4 device , same serial number will be programmed for USB 2.0 and USB 3.1 Gen1.

Example

```
typedef INT (*pfMchpUsbGetAllHubs) (PCHAR);
typedef HANDLE (*pfMchpUsbOpen) (CONST UINT);
typedef UINT32 (*pfMchpUsbGetLastError) (HANDLE);
typedef BOOL (*pfMchpProgramFileWithSerial) (HANDLE, PCHAR, PCHAR);

pfMchpUsbGetAllHubs libMchpUsbGetAllHubs;
pfMchpUsbOpen libMchpUsbOpen;
pfMchpUsbGetLastError libMchpUsbGetLastError;
pfMchpProgramFileWithSerial libMchpProgramFileWithSerial;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
libMchpUsbGetAllHubs = (pfMchpUsbGetAllHubs) GetProcAddress ( LoadLib
, "MchpUsbGetAllHubs");
libMchpUsbOpen = (pfMchpUsbOpen) GetProcAddress ( LoadLib , "MchpUsbOpen");
libMchpUsbGetLastError = (pfMchpUsbGetLastError) GetProcAddress ( LoadLib
, "MchpUsbGetLastError");
libMchpProgramFileWithSerial = (pfMchpProgramFileWithSerial) GetProcAddress ( LoadLib
, "MchpProgramFileWithSerial");

int hub_count = 0;
hub_count = libMchpUsbGetAllHubs(sztext);

if(hub_count)
{
    cout<<"Connected usb hubs are ..nn";
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< hubindex <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError = Error_Success;

hDevice = libMchpUsbOpen(hubindex);
if(INVALID_HANDLE_VALUE == hDevice)
```

```

{
    dwError = libMchpUsbGetLastError(HANDLE(hubindex));
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfullyn");

if (FALSE == libMchpProgramFileWithSerial(hDevice,"CONFIG_FILE.cfg","123456"))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Device Program Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Device Programmed Successfullyn";

```

7.2.9.4 MchpUsbProgramFileWithUsb3Serial Function

C

```

MCHP_USB_API BOOL MchpUsbProgramFileWithUsb3Serial(
    HANDLE DevID,
    PCHAR InputFileName,
    PCHAR Usb2SerialNumber,
    PCHAR Usb3SerialNumber
);

```

Devices Supported

USB57X4

Description

This API will program configuration file as specified in pchInputFileName along with the pchUsb2SerialNo serial number and pchusb3SerialNo serial num to the selected device ID.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
InputFileName	Input configuration file to be programmed into the device.This argument is optional.NULL can be passed to this argument if input bin file is not given.
Usb2SerialNumber	USB2 Serial Number to be programmed
Usb3SerialNumber	USB3.1 Gen1 Serial Number to be programmed

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

Example

```

typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbProgramFileWithUsb3Serial) (HANDLE,PCHAR, PCHAR,PCHAR);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpUsbProgramFileWithUsb3Serial libMchpUsbProgramFileWithUsb3Serial;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");

libMchpUsbProgramFileWithUsb3Serial = (pfMchpUsbProgramFileWithUsb3Serial)
GetProcAddress ( LoadLib ,"MchpUsbProgramFileWithUsb3Serial");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

if (FALSE ==
libMchpUsbProgramFileWithUsb3Serial(hDevice,"otp_image.bin","MCHP1234","MCHP4321"))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Device Program Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Device Programmed Successfull\n";

if (FALSE == libMchpUsbClose(hDevice))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04x",dwError);
    printf ("nPress any key to exit....");
    _getch ();
}

```

```
        exit (1);
    }
```

7.2.9.5 MchpProgramSPIFirmwareWithConfig Function

C

```
MCHP_USB_API BOOL MchpProgramSPIFirmwareWithConfig(
    HANDLE DevID,
    BOOL EraseConfig,
    PCHAR FirmwareFile,
    PCHAR ConfigFile
);
```

Devices Supported

USB253X/USB4604

Description

This API will program firmware file as specified in FirmwareFile argument along with the configuration file as specified in ConfigFile argument.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
EraseConfig	TRUE - Erase all programmed configuration data in spi memory FALSE - Do not erase programmed configuration data in spi memory
FirmwareFile	Input firmware bin file to be programmed into the device
ConfigFile	Input configuration file to be programmed into the device

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Example

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpProgramSPIFirmwareWithConfig) (HANDLE,BOOL,PCHAR,PCHAR);
CHAR sztext[2048];

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpProgramSPIFirmwareWithConfig libMchpProgramSPIFirmwareWithConfig;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
}
```

```

        _getch ();
        exit (1);
    }

    libMchpUsbOpenID          = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
, "MchpUsbOpenID");
    libMchpUsbClose           = (pfMchpUsbClose) GetProcAddress ( LoadLib , "MchpUsbClose");
    libMchpProgramSPIFirmwareWithConfig =
(pfMchpProgramSPIFirmwareWithConfig)GetProcAddress ( LoadLib
, "MchpProgramSPIFirmwareWithConfig");

    HANDLE hDevice =  INVALID_HANDLE_VALUE;

    UINT32 dwError;

    hDevice = libMchpUsbOpenID(0x424, 0x1234);
    if(INVALID_HANDLE_VALUE == hDevice)
    {
        dwError = libMchpUsbGetLastError(hDevice);
        printf ("Error,%04xn",dwError);
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    printf("Device Opened successfulln");
    if (FALSE ==
libMchpProgramSPIFirmwareWithConfig(hDevice,TRUE,"spifirmware.bin", "myconfig.cfg"))
    {
        dwError = libMchpUsbGetLastError(hDevice);
        printf ("Device Program Failed, Error %04xn",dwError);
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
}

```

7.2.9.6 MchpProgramSpiFwCfgFileWithSerial Function

C

```

MCHP_USB_API BOOL MchpProgramSpiFwCfgFileWithSerial(
    HANDLE DevID,
    BOOL EraseConfig,
    PCHAR FirmwareFile,
    PCHAR ConfigFile,
    PCHAR SerialNumber
);

```

Devices Supported

USB4604,USB57x4,USB58XX/USB59XX

Description

This API will program firmware file as specified in FirmwareFile argument along with the configuration file as specified in ConfigFile argument and Serial Number specified in SerialNumber argument

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
EraseConfig	TRUE - Erase all programmed configuration data in spi memory FALSE - Do not erase programmed configuration data in spi memory
FirmwareFile	Input firmware bin file to be programmed into the device
ConfigFile	Input configuration file to be programmed into the device
SerialNumber	Serial Number to be programmed

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks

None.

Example

```

typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpProgramSpiFwCfgFileWithSerial) (HANDLE,BOOL,PCHAR,PCHAR);
CHAR sztext[2048];

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpProgramSpiFwCfgFileWithSerial libMchpProgramSpiFwCfgFileWithSerial;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");
libMchpProgramSpiFwCfgFileWithSerial =
(pfMchpProgramSpiFwCfgFileWithSerial)GetProcAddress ( LoadLib
,"MchpProgramSpiFwCfgFileWithSerial");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");
if (FALSE ==
libMchpProgramSpiFwCfgFileWithSerial(hDevice,TRUE,"spifirmware.bin","myconfig.cfg","1234"))
{

```



```

        dwError = libMchpUsbGetLastError(hDevice);
        printf ("Device Program Failed, Error %04xn",dwError);
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }

```

7.2.9.7 MchpProgramSPIFirmwareWithConfigBuffer Function

C

```

MCHP_USB_API BOOL MchpProgramSPIFirmwareWithConfigBuffer(
    HANDLE DevID,
    BOOL EraseConfig,
    UINT8* FirmwareData,
    UINT32 SpiBytesToWrite,
    UINT8* ConfigData,
    UINT32 CfgBytesToWrite
);

```

Devices Supported

USB4604,USB57X4,USB58XX/USB59XX

Description

This API will program firmware data as stored in pointer FirmwareData argument along with the configuration data as stored in pointer ConfigData argument.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device
EraseConfig	TRUE - Erase all programmed configuration data in spi memory FALSE - Do not erase programmed configuration data in spi memory
FirmwareData	Pointer to the buffer contains SPI firmware data to be written
SpiBytesToWrite	Number of Firmware bytes to be written
ConfigData	Pointer to the buffer contains configuration data to be written
CfgBytesToWrite	Number of configuration bytes to be written

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Example

```

typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpProgramSPIFirmwareWithConfigBuffer) (HANDLE,BOOL,UINT8*,
UINT32,UINT8*, UINT32);
CHAR sztext[2048];

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpProgramSPIFirmwareWithConfigBuffer libMchpProgramSPIFirmwareWithConfigBuffer;

```

```

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID          = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose           = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");
libMchpProgramSPIFirmwareWithConfigBuffer = (pfMchpProgramSPIFirmwareWithConfigBuffer)
GetProcAddress (LoadLib,"MchpProgramSPIFirmwareWithConfigBuffer");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

FILE      *ffwfile;
FILE      *fcfgfile;
UINT8 FirmwareData[65536];
UINT8 ConfigData[2048];

fopen_s(&ffwfile, "USB2534_SPI_CARPLAY.bin", "rb");
fread(&FirmwareData, 1, sizeof(FirmwareData), ffwfile);
fclose(ffwfile);

fopen_s(&fcfgfile, "did_5678.cfg", "rb");
fread(&ConfigData, 1, sizeof(ConfigData), fcfgfile);
fclose(fcfgfile);






if (FALSE ==
libMchpProgramSPIFirmwareWithConfigBuffer(hDevice,false,FirmwareData,65536,ConfigData,7))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Device Program Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

7.2.10 Miscellaneous APIs

Functions

	Name	Description
≡	MchpUsbGetLastError	Get last error for the specific hub instance.
≡	MchpVerifyWidgetValues	Verify hub Configuration parameters
≡	MchpUsbDeviceReset	This API for will issue soft reset to the device.

	MchpUsbHCETransfer	This API will issue the command to Hub Controller Endpoint(HCE).
	MchpUsbVSMTransfer	This API will issue the VSM command to the hub.
	MchpUsbGetAllHubsPortChainInfo	This API will get the port chain information about the hubs which are connected to the computer.
	MchpGetHubPortChain	This API will get the port chain of the hub from the specified index of the hub.
	MchpGetHubIndex	This API will get the hub index from the specified port chain of the hub.

Description

This section lists all miscellaneous APIs which contains various additional features.

7.2.10.1 MchpUsbGetLastError Function

C

```
MCHP_USB_API UINT32 MchpUsbGetLastError(
    HANDLE DevID
);
```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will get last error occurred when handling other API's in this library.

Preconditions

None.

Parameters

Parameters	Description
DevID	Handle to the device - Return value of MchpUsbOpen or MchpUsbOpenID or MchpUsbHCEOpen .

Returns

APISTATUS Error codes.

Remarks

None

Example

```
CHAR sztext[2048];
typedef UINT32 (*pfMchpUsbGetLastError) (HANDLE);
pfMchpUsbGetLastError libMchpUsbGetLastError;
CONST UINT hubindex=0;

HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
libMchpUsbGetLastError = (pfMchpUsbGetLastError) GetProcAddress ( LoadLib
,"MchpUsbGetLastError");
dwError = libMchpUsbGetLastError(hDevice);

//Print error here
cout << dwError << endl;
```

7.2.10.2 MchpVerifyWidgetValues Function

C

```
MCHP_USB_API BOOL MchpVerifyWidgetValues(  
    HANDLE DevID,  
    PCHAR Widgets,  
    PCHAR Values  
);
```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will Verify hub Configuration parameters on the selected device ID.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device - Return value of MchpUsbOpen or MchpUsbOpenID or MchpUsbHCEOpen .
Widgets	Pointer to the buffer containing hub configurations parameters
Values	Pointer to the buffer where hub configurations parameters read from the hub.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Supported Hub Configuration Items For USB57X4 and USB2534:

1. vid

This is a 16-bit value that uniquely identifies the Vendor of the user device (idVendor: assigned by USB-Interface Forum).

2. pid

This is a 16-bit value that the Vendor can assign that uniquely identifies this particular product (idProduct)

3. did

This is a 16-bit device release number in BCD format(bcdDevice)

4. usbvcd

USB2 Specification Release Number in BCD format(bcdUSB)

5. languageid

USB2 LANGUAGE ID

6. manufacturer

Manufacturer String of the USB2 HUB.

7. product

Product String of the USB2 HUB.

8. serial

Serial String of the USB2 HUB.

Additional Supported Hub Configuration Items For USB57X4 :**9. usb3vid**

This is a 16-bit value that uniquely identifies the Vendor of the USB3.1 Gen1 user device (idVendor: assigned by USB-Interface)

10. usb3pid

This is a 16-bit value that the Vendor can assign that uniquely identifies this particular product for USB 3.1 Gen1 user device (idProduct)

11. usb3did

This is a 16-bit device release number for USB 3.1 Gen1 user device in BCD format (bcdDevice)

12. usb3vcd

USB 3.1 Gen1 Specification Release Number in BCD format (bcdUSB)

13. usb3languageid

USB 3.1 Gen1 LANGUAGE ID

14. usb3manufacturer

Manufacturer String of the USB 3.1 Gen1 HUB.

15. usb3product

Product String of the USB 3.1 Gen1 HUB.

16. usb3serial

Serial String of the USB 3.1 Gen1 HUB.

Only above listed items are supported in this API and item names are case sensitive.

Example

```
typedef INT (*pfMchpUsbGetAllHubs) (PCHAR);
typedef HANDLE (*pfMchpUsbOpen) (CONST UINT);
typedef UINT32 (*pfMchpUsbGetLastErr) (HANDLE);
typedef BOOL (*pfMchpVerifyWidgetValues) (HANDLE, PCHAR, PCHAR);

pfMchpUsbGetAllHubs libMchpUsbGetAllHubs;
pfMchpUsbOpen libMchpUsbOpen;
pfMchpUsbGetLastErr libMchpUsbGetLastErr;
pfMchpVerifyWidgetValues libMchpVerifyWidgetValues;

CHAR sztext[2048];
```

```

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
libMchpUsbGetAllHubs          = (pfMchpUsbGetAllHubs) GetProcAddress ( LoadLib
,"MchpUsbGetAllHubs");
libMchpUsbOpen                = (pfMchpUsbOpen) GetProcAddress ( LoadLib ,"MchpUsbOpen");
libMchpUsbGetLastError        = (pfMchpUsbGetLastError) GetProcAddress ( LoadLib
,"MchpUsbGetLastError");
libMchpVerifyWidgetValues     = (pfMchpVerifyWidgetValues) GetProcAddress ( LoadLib
,"MchpVerifyWidgetValues");

int hub_count = 0;
hub_count = libMchpUsbGetAllHubs(sztext);

if(hub_count)
{
    cout<<"Connected usb hubs are ..nn";
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< hubindex <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError = Error_Success;

hDevice = libMchpUsbOpen(hubindex);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(HANDLE(hubindex));
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfullyn");

char widget_buffer_names[4098];
char widget_buffer_values[4098];
memset(widget_buffer_names,0x00,sizeof(widget_buffer_names));
string verifyParams [] = {"vid",
                           "pid",
                           "did",
                           "usbvcd",
                           "languageid",
                           "manufacturer",
                           "product",
                           "serial",
                           "END"};
for (int i=0;verifyParams[i] != "END"; i++)
{
    strcat(widget_buffer_names,verifyParams[i].c_str());
    strcat(widget_buffer_names,",");
}

// Pass comma seperated parameters to the DLL API

```

```

libMchpVerifyWidgetValues(hDevice,widget_buffer_names,widget_buffer_values);

//Parse the comma sepearted value from DLL API
string ParseValue(widget_buffer_values);

//Small Parser code
size_t pos = 0, position;
std::string token;
int j = 0;
while ((pos = ParseValue.find(",")) != std::string::npos) {

    position = ParseValue.find(",");
    token = ParseValue.substr(0, position);
    //Compare each values and print
    printf("WidgetName : %s , Value : %sn",verifyParams[j++].c_str(),token.c_str());

    ParseValue.erase(0, position + 1);

}

```

7.2.10.3 MchpUsbDeviceReset Function

C

```

MCHP_USB_API BOOL MchpUsbDeviceReset(
    HANDLE DevID
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API for will issue soft reset to the device.

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
DevID	Handle to the device

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Example

```

typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpUsbDeviceReset) (HANDLE);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;

```

```

pfMchpUsbDeviceReset libMchpUsbDeviceReset;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID          = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose           = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");
libMchpUsbDeviceReset     =(pfMchpUsbDeviceReset) GetProcAddress ( LoadLib
,"MchpUsbDeviceReset");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");
//Reset the device
if (FALSE == libMchpUsbDeviceReset(hDevice))
{
    printf ("MchpUsbDeviceReset failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

```

7.2.10.4 MchpUsbHCETTransfer Function

C

```

MCHP_USB_API BOOL MchpUsbHCETransfer(
    HANDLE DevID,
    UINT8 bmRequestType,
    UINT8 * pbyBuffer,
    UINT8 bRequest,
    UINT16 wValue,
    UINT16 wIndex,
    UINT16 wLength
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will issue the command to Hub Controller Endpoint(HCE).

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
hub_index	Index of the Hub.hub_index found from the MchpUsbGetAllHubs API.
bmRequestType	bmRequestType field in the Setup Packet.Device to Host or Host to Device based on the Setup Command.
pbyBuffer	Pointer to the Buffer.
bRequest	bmRequestType field in the Setup Packet.
wValue	wValue field in the Setup Packet.
wIndex	wIndex field in the Setup Packet.
wLength	wLength field in the Setup Packet.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastError for more details) - for failure

Remarks**Example**

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);

typedef BOOL (*pfMchpUsbHCETransfer) (HANDLE,UINT8,UINT8*,UINT8,UINT16,UINT16,UINT16);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;

pfMchpUsbHCETransfer libMchpUsbHCETransfer;

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loadedn");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
```

```

libMchpUsbOpenID          = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose           = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");

libMchpUsbHCETransfer = (pfMchpUsbHCETransfer) GetProcAddress ( LoadLib
,"MchpUsbHCETransfer");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");
UINT8 byBufferData[2],bRequest;
UINT16 wValue , wIndex , wLength,bmRequestType;

bmRequestType = 0xc1;
bRequest      = 0x04;      //CMD_XDATA_READ
wValue        = 0x3000;    //XDATA
wIndex        = 0x00; // xdata address
wLength       = 2 ;       // No.of Bytes to be Read
if (FALSE ==
libMchpUsbHCETransfer(0,bmRequestType,&byBufferData[0],bRequest,wValue,wIndex,wLength))
{
    printf ("MchpUsbHCETransfer failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
//Write Xdata
bmRequestType = 0x41;
bRequest      = 0x03;      //VSM_WRITE_MEM
wValue        = 0x3000;    //XDATA
wIndex        = 0x00; // xdata address
wLength       = 2 ;       // No.of Bytes to be Read
byBufferData[0] = 0xAA;
byBufferData[1] = 0xBB;
if (FALSE ==
libMchpUsbHCETransfer(0,bmRequestType,&byBufferData[0],bRequest,wValue,wIndex,wLength))
{
    printf ("MchpUsbHCETransfer failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}

bmRequestType = 0xc1;
bRequest      = 0x04;      //CMD_XDATA_READ
wValue        = 0x3000;    //XDATA
wIndex        = 0x00; // xdata address
wLength       = 2 ;       // No.of Bytes to be Read
if (FALSE ==
libMchpUsbHCETransfer(0,bmRequestType,&byBufferData[0],bRequest,wValue,wIndex,wLength))

```

```

{
    printf ("MchpUsbHCETransfer failed");

    printf ("\nPress any key to exit....");

    _getch ();

    exit (1);
}

```

7.2.10.5 MchpUsbVSMTransfer Function

C

```

MCHP_USB_API BOOL MchpUsbVSMTransfer(
    CONST UINT hub_index,
    UINT8 bmRequestType,
    UINT8 * pbyBuffer,
    UINT8 bRequest,
    UINT16 wValue,
    UINT16 wIndex,
    UINT16 wLength
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will issue the Vendor Specific Messaging (VSM) command to the hub. This feature will allow a host driver to read and write registers in the device.

Preconditions

[MchpUsbGetAllHubs](#) should be called before calling this API.

The Hub class VSM filter driver should be installed on the computer before calling this API.

Parameters

Parameters	Description
hub_index	Index of the Hub.hub_index found from the MchpUsbGetAllHubs API.
bmRequestType	bmRequestType field in the VSM Setup Packet.Device to Host or Host to Device based on the VSM Command.
pbyBuffer	Pointer to the Buffer which contains the data.
bRequest	bmRequestType field in the VSM Setup Packet.
wValue	wValue field in the VSM Setup Packet.
wIndex	wIndex field in the VSM Setup Packet.
wLength	wLength field in the VSM Setup Packet.

Returns

TRUE - for Success;

FALSE - (Call GetMchpUsbLastErr for more details) - for failure

Remarks

Example

```

typedef INT (*pfMchpUsbGetAllHubs) (PCHAR);
typedef BOOL (*pfMchpUsbVSMTransfer) (CONST
UINT,UINT8,UINT8*,UINT8,UINT16,UINT16,UINT16);

pfMchpUsbGetAllHubs libMchpUsbGetAllHubs;
pfMchpUsbVSMTransfer libMchpUsbVSMTransfer;

```

```

CHAR sztext[2048];

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
libMchpUsbGetAllHubs          = (pfMchpUsbGetAllHubs) GetProcAddress ( LoadLib
,"MchpUsbGetAllHubs");
libMchpUsbVSMTransfer = (pfMchpUsbVSMTransfer) GetProcAddress ( LoadLib
,"MchpUsbVSMTransfer");

int hub_count = 0;
hub_count = libMchpUsbGetAllHubs(sztext);

if(hub_count)
{
    cout<<"Connected usb hubs are ..nn";
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< hubindex <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
UINT8 byBufferData[3],bRequest,bmRequestType;

UINT16 wValue , wIndex , wLength;
bmRequestType = 0xc0;
bRequest      = 0x01;      //VSM_READ_TARGET_MEMORY
wValue        = 0x00;      //XDATA
wIndex        = 0x3000;    // xdata address
wLength       = 2 ;        // No.of Bytes to be Read
if (FALSE ==
libMchpUsbVSMTransfer(0,bmRequestType,&byBufferData[0],bRequest,wValue,wIndex,wLength))
{
    printf ("MchpUsbVSMTransfer failed");

    printf ("nPress any key to exit....");

    _getch ();

    exit (1);
}
//Write Xdata
bmRequestType = 0x40;
bRequest      = 0x02;      //VSM_WRITE_MEM
wValue        = 0x00;      //XDATA
wIndex        = 0x3000;    // xdata address
wLength       = 3 ;        // No.of Bytes to be Read
byBufferData[0] = 0x03;
byBufferData[1] = 0xAA;
byBufferData[2] = 0xBB;
if (FALSE ==
libMchpUsbVSMTransfer(0,bmRequestType,&byBufferData[0],bRequest,wValue,wIndex,wLength))
{
    printf ("MchpUsbVSMTransfer failed");

    printf ("nPress any key to exit....");

    _getch ();
}

```

```

        exit (1);
    }
    bmRequestType = 0xc0;
    bRequest      = 0x01;      //VSM_READ_TARGET_MEMORY
    wValue        = 0x00;      //XDATA
    wIndex        = 0x3000;    // xdata address
    wLength       = 2 ;        // No.of Bytes to be Read
    if (FALSE ==
    libMchpUsbVSMTransfer(0,bmRequestType,&byBufferData[0],bRequest,wValue,wIndex,wLength))
    {
        printf ("MchpUsbVSMTransfer failed");

        printf ("nPress any key to exit....");

        _getch ();

        exit (1);
    }

```

7.2.10.6 MchpUsbGetAllHubsPortChainInfo Function

C

```

MCHP_USB_API INT MchpUsbGetAllHubsPortChainInfo(
    PCHAR HubPortChainInfo
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will get the port chain information about the hubs which are connected to the computer.

Port chain information of the hubs will be stored in the given argument HubPortChainInfo.

Preconditions

None

Parameters

Parameters	Description
HubPortChainInfo	Port chain information of all hubs connected to the computer will be stored in HubPortChainInfo argument. The user must allocate minimum of 2048 characters for HubPortChainInfo.

Returns

Number of connected hubs will be returned.

Remarks

None.

Example

```

typedef INT (*pfMchpUsbGetAllHubsPortChainInfo) (PCHAR);

pfMchpUsbGetAllHubsPortChainInfo libMchpUsbGetAllHubsPortChainInfo;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
}

```

```

    _getch ();
    exit (1);
}

```

```
libMchpUsbGetAllHubsPortChainInfo = (pFMchpUsbGetAllHubsPortChainInfo) GetProcAddress (
LoadLib , "MchpUsbGetAllHubsPortChainInfo");
```

[illegible]

7.2.10.7 MchpGetHubPortChain Function

C

```
MCHP_USB_API BOOL MchpGetHubPortChain(
    UINT HubIndex,
    PCHAR chPortChain
);
```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will get the port chain of the hub in specified `chPortChain` argument from the given index of the hub in `HubIndex` argument.

Preconditions

MchpUsbGetAllHubs and **MchpUsbOpen** should be called before calling this API.

(or)

MchpUsbOpenID should be called before calling this API

(or)

MchpUsbHCEOpen should be called before calling this API.

Parameters

Parameters	Description
HubIndex	Index of the hub found from MchpUsbGetAllHubs API (or) HANDLE of the hub found from MchpUsbOpenID , MchpUsbOpen and MchpUsbHCEOpen
chPortChain	Port chain of the hub will be copied to chPortChain buffer if the return value is TRUE. User must allocate minimum of 50 characters(Based on the port chain length).

Returns

TRUE - for Success;

FALSE - for failure

Remarks

None.

Example

```

typedef HANDLE (*pfMchpUsbGetAllHubs) (PCHAR);
typedef HANDLE (*pfMchpUsbOpen) (CONST UINT);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpGetHubIndex) (PCHAR);
typedef BOOL (*pfMchpGetHubPortChain) (UINT, PCHAR);
typedef BOOL (*pfMchpProgramFile) (HANDLE, PCHAR);
typedef BOOL (*pfMchpVerifyWidgetValues) (HANDLE, PCHAR, PCHAR)

pfMchpUsbGetAllHubs libMchpUsbGetAllHubs;
pfMchpUsbOpen libMchpUsbOpen;
pfMchpUsbClose libMchpUsbClose;
pfMchpGetHubIndex libMchpGetHubIndex;
pfMchpGetHubPortChain libMchpGetHubPortChain;
pfMchpProgramFile libMchpProgramFile;
pfMchpVerifyWidgetValues libMchpVerifyWidgetValues;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
libMchpUsbGetAllHubs = (pfMchpUsbGetAllHubs) GetProcAddress ( LoadLib
, "MchpUsbGetAllHubs");
libMchpUsbOpen = (pfMchpUsbOpen) GetProcAddress ( LoadLib , "MchpUsbOpen");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib , "MchpUsbClose");
libMchpGetHubPortChain = (pfMchpGetHubPortChain) GetProcAddress ( LoadLib
, "MchpGetHubPortChain");
libMchpGetHubIndex = (pfMchpGetHubIndex) GetProcAddress ( LoadLib , "MchpGetHubIndex");
libMchpProgramFile = (pfMchpProgramFile) GetProcAddress ( LoadLib
, "MchpProgramFile");
libMchpVerifyWidgetValues = (pfMchpVerifyWidgetValues) GetProcAddress ( LoadLib
, "MchpVerifyWidgetValues");

HANDLE hDevice = INVALID_HANDLE_VALUE;
UINT32 dwError;
CHAR sztxt[2048];
CHAR chUSBPortChainInf[50];
UINT hubIndex = 0; //For example

//Get all usb hubs
libMchpUsbGetAllHubs(sztxt);

//Store port chain of the hub
libMchpGetHubPortChain(hubIndex, chUSBPortChainInf);

printf ("%s", chUSBPortChainInf);

// Get handle to the hub
hDevice = libMchpUsbOpen(0x424, 0x2734);
if (INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
}

```

```

    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

//Change VendorID by programming OTP using CONFIG_FILE.cfg (Generate from PT2 tool)
if (FALSE == libMchpProgramFile(hDevice,"CONFIG_FILE.cfg"))
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Device Program Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
//Close the hub
libMchpUsbClose(hDevice);

//After programming hub will be moved to other index's since the VID is changed in this
example.
//To access the same hub which was programmed before
//There would be the changes in the usb tree, so Get all usb hubs again to retrieve
libMchpUsbGetAllHubs(sztxt);

//Get the hub index using portchain, so that we can verify the parameters
UINT uHubIndex = libMchpGetHubIndex(chUSBPortChainInf);
if (INVALID_HANDLE_VALUE == uHubIndex)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Device Program Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
else
{
    hDevice = libMchpUsbOpen(0x424,0x2734);
    if(INVALID_HANDLE_VALUE == hDevice)
    {
        //Error
    }
    char widgetName[256] = {"vid"};
    char widgetValue[256];

    //Verify programmed VID here
    libMchpVerifyWidgetValues(hDevice,widgetName,widgetValue);
}
}

```

7.2.10.8 MchpGetHubIndex Function

C

```

MCHP_USB_API UINT MchpGetHubIndex(
    PCHAR Portchain
);

```

Devices Supported

USB253X/USB4604/USB3X13,USB57X4,USB58XX/USB59XX

Description

This API will return the hub index from the specified port chain of the hub in the Portchain argument .

Preconditions

[MchpUsbGetAllHubs](#) and [MchpUsbOpen](#) should be called before calling this API.

(or)

[MchpUsbOpenID](#) should be called before calling this API

(or)

[MchpUsbHCEOpen](#) should be called before calling this API.

Parameters

Parameters	Description
Portchain	Port chain of the hub found from MchpUsbGetAllHubsPortChainInfo API

Returns

Hub index - for Success;

INVALID_HANDLE_VALUE - for failure

Remarks

None.

Example

```
typedef HANDLE (*pfMchpUsbOpenID) (UINT16,UINT16);
typedef BOOL (*pfMchpUsbClose) (HANDLE);
typedef BOOL (*pfMchpGetHubIndex) (PCHAR);
typedef INT (*pfMchpUsbGetAllHubsPortChainInfo) (PCHAR);

pfMchpUsbOpenID libMchpUsbOpenID;
pfMchpUsbClose libMchpUsbClose;
pfMchpGetHubIndex libMchpGetHubIndex;
pfMchpUsbGetAllHubsPortChainInfo libMchpUsbGetAllHubsPortChainInfo;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loadedn");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpUsbOpenID = (pfMchpUsbOpenID) GetProcAddress ( LoadLib
,"MchpUsbOpenID");
libMchpUsbClose = (pfMchpUsbClose) GetProcAddress ( LoadLib ,"MchpUsbClose");
libMchpUsbGetAllHubsPortChainInfo = (pfe) GetProcAddress ( LoadLib
,"MchpUsbGetAllHubsPortChainInfo");
libMchpGetHubIndex = (pfMchpGetHubIndex)GetProcAddress ( LoadLib ,"MchpGetHubIndex");

HANDLE hDevice = INVALID_HANDLE_VALUE;

UINT32 dwError;

hDevice = libMchpUsbOpenID(0x424, 0x1234);
if(INVALID_HANDLE_VALUE == hDevice)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Error,%04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");
char chUSBPortChainInf[2048];
libMchpUsbGetAllHubsPortChainInfo(chUSBPortChainInf);

printf ("%s",chUSBPortChainInf);

//Here Portchain is set as "2-3" which is found from the API
```

MchpUsbGetAllHubsPortChainInfo

```
UINT uHubIndex = libMchpGetHubIndex("2-3");
if (INVALID_HANDLE_VALUE == uHubIndex)
{
    dwError = libMchpUsbGetLastError(hDevice);
    printf ("Device Program Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
else
{
    //uHubIndex is valid
}
```

7.3 LAN APIs

7.3.1 Device Open/Close

Functions

	Name	Description
≡	MchpLanGetDevices	This API will allow the user enable/disable the Auto re Program when writing configuration file on OTP Device Support: USB57X4,USB58xx,USB59XX
≡	MchpLanCloseAllHandles	This API will close all the LAN device handles
≡	MchpLanOpen	This API will return handle to the first instance of a device with the specified index
≡	MchpLanOpenID	This API will return handle to the first instance of a device with the specified index
≡	MchpLanClose	This API will close the handle for device specified in the call

7.3.1.1 MchpLanGetDevices Function

C

```
MCHP_LAN_API INT MchpLanGetDevices(  
    PCHAR LanInfo  
);
```

Parameters

Parameters	Description
Autoreprogram	Set to enable Autoreprogram Clear to disable Autoreprogram

Returns

Function returns nothing

7.3.1.2 MchpLanCloseAllHandles Function

C

```
MCHP_LAN_API void MchpLanCloseAllHandles();
```

Devices Supported

LAN7800/LAN7850

Description

This API will close all the LAN device handles

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device - Return value of MchpLanOpen or MchpLanOpenID

Returns

None

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef HANDLE (*pfMchpLanOpenID) (UINT16,UINT16,UINT8*);
typedef void (*pfMchpLanClose) (HANDLE);
typedef void (*pfMchpLanCloseAllHandles)();
typedef void (*pfMchpEnableLogging) (INT);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanOpenID libMchpLanOpenID;
pfMchpLanClose libMchpLanClose;
pfMchpLanCloseAllHandles libMchpLanCloseAllHandles;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices          = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanOpenID              = (pfMchpLanOpenID) GetProcAddress ( LoadLib
,"MchpLanOpenID");
libMchpLanClose               = (pfMchpLanClose) GetProcAddress ( LoadLib
,"MchpLanClose");
libMchpEnableLogging          = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");
libMchpLanCloseAllHandles     = (pfMchpLanCloseAllHandles) GetProcAddress (
LoadLib ,"MchpLanCloseAllHandles");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs found\n");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
BYTE curMacAddr[6] = {0x00,0x80,0x0F,0x78,0x00,0x01};
hDevice = libMchpLanOpenID(0x0424,0x7800,curMacAddr);
if(hDevice == INVALID_HANDLE_VALUE)
{

```

```

        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    printf("Device Opened successfullyn");

    libMchpLanClose(hDevice);

    libMchpLanCloseAllHandles();

```

7.3.1.3 MchpLanOpen Function

C

```

MCHP_LAN_API HANDLE MchpLanOpen(
    UINT32 LANAdapterIndex
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will return handle to the first instance of a device with the specified index

Preconditions

[MchpLanGetDevices](#) API should be called before calling this API

Parameters

Parameters	Description
LANAdapterIndex	Index of the LAN Devices connected to the system - MchpLanGetDevices

Returns

HANDLE of the selected LANAdapterIndex INVALID_HANDLE_VALUE (Call [MchpLanGetLastError](#) for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef HANDLE (*pfMchpLanOpen) (UINT32);
typedef void (*pfMchpEnableLogging) (INT);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanOpen libMchpLanOpen;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loadedn");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
, "MchpLanGetDevices");
libMchpLanOpen = (pfMchpLanOpen) GetProcAddress ( LoadLib , "MchpLanOpen");
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress (
LoadLib, "MchpEnableLogging");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

```

```

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
hDevice = libMchpLanOpen(LanID);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");

```

7.3.1.4 MchpLanOpenID Function

C

```

MCHP_LAN_API HANDLE MchpLanOpenID(
    UINT16 VendorID,
    UINT16 ProductID,
    UINT8* MacAddress
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will return handle to the first instance of a device matched with VendorID, ProductID and Mac Address

Preconditions

[MchpLanGetDevices](#) API should be called before calling this API

Parameters

Parameters	Description
VendorId	Vendor ID(VID) of the LAN device.
ProductID	Product ID(PID) of the LAN device
MacAddress	Mac Address of the LAN device. If this value is NULL, handle to the first instance of a device matched with VendorID, ProductID will be return

Returns

HANDLE of the selected Vendor ID, Product ID and Mac Address matched LAN device - for success
INVALID_HANDLE_VALUE (Call [MchpLanGetLastError](#) for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef HANDLE (*pfMchpLanOpenID) (UINT16,UINT16,UINT8*);
typedef void (*pfMchpEnableLogging) (INT);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanOpenID libMchpLanOpenID;

```

```

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices      = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanOpenID          = (pfMchpLanOpenID) GetProcAddress ( LoadLib
,"MchpLanOpenID");
libMchpEnableLogging      = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
BYTE curMacAddr[6]  = {0x00,0x80,0x0F,0x78,0x00,0x01};
hDevice = libMchpLanOpenID(0x0424,0x7800,curMacAddr);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

```

7.3.1.5 MchpLanClose Function

C

```

MCHP_LAN_API void MchpLanClose(
    HANDLE LanID
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will close the handle for device specified in the call

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device - Return value of MchpLanOpen or MchpLanOpenID

Returns

None

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef HANDLE (*pfMchpLanOpenID) (UINT16,UINT16,UINT8*);
typedef void (*pfMchpLanClose) (HANDLE);
typedef void (*pfMchpEnableLogging) (INT);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanOpenID libMchpLanOpenID;
pfMchpLanClose libMchpLanClose;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices      = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanOpenID          = (pfMchpLanOpenID) GetProcAddress ( LoadLib
,"MchpLanOpenID");
libMchpLanClose           = (pfMchpLanClose) GetProcAddress ( LoadLib
,"MchpLanClose");
libMchpEnableLogging      = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs found\n");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
BYTE curMacAddr[6]  = {0x00,0x80,0x0F,0x78,0x00,0x01};
hDevice = libMchpLanOpenID(0x0424,0x7800,curMacAddr);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}



```



```
printf("Device Opened successfullyn");  
libMchpLanClose(hDevice);
```

7.3.2 LAN Programming APIs

Functions

	Name	Description
	MchpLanProgramFile	This API will program the Input file given as argument to the selected LAN device ID.
	MchpLanProgramFileWithSerial	This API will program the Input file along with serial number given as argument to the selected LAN device ID.

7.3.2.1 MchpLanProgramFile Function

C

```
MCHP_LAN_API BOOL MchpLanProgramFile(  
    HANDLE LanID,  
    UINT AccessType,  
    PCHAR InputFileName,  
    PCHAR MacAddress  
);
```

Devices Supported

LAN7800/LAN7850

Description

This API will program the Input file given as argument to the selected LAN device ID.

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
AccessType	1- EEPROM , 2- OTP
InputFileName	Input file to be programmed into the device. Supported file type .bin/.ini
MacAddress	Mac Address to be overwritten. If NULL, Mac Address will not be overwritten

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastErr for more details) - for failure

Example

```
typedef INT (*pfMchpLanGetDevices) (PCHAR);  
typedef HANDLE (*pfMchpLanOpenID) (UINT16,UINT16,UINT8*);  
typedef BOOL (*pfMchpLanProgramFile) (HANDLE,UINT,PCHAR,PCHAR);  
typedef void (*pfMchpEnableLogging) (INT);  
  
pfMchpLanGetDevices libMchpLanGetDevices;  
pfMchpEnableLogging libMchpEnableLogging;  
pfMchpLanOpenID libMchpLanOpenID;  
pfMchpLanProgramFile libMchpLanProgramFile;  
  
//Load library  
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
```

```

if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices      = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanOpenID          = (pfMchpLanOpenID) GetProcAddress ( LoadLib
,"MchpLanOpenID");
libMchpLanProgramFile     = (pfMchpLanProgramFile) GetProcAddress ( LoadLib
,"MchpLanProgramFile");
libMchpEnableLogging      = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
BYTE curMacAddr[6]  = {0x00,0x80,0x0F,0x78,0x00,0x01};
hDevice = libMchpLanOpenID(0x0424,0x7800,curMacAddr);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

//Program File
if (FALSE == libMchpLanProgramFile(hDevice,1,"7800eep.bin",(PCHAR)curMacAddr))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Device Program Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Device Programmed Successfully n";

```

7.3.2.2 MchpLanProgramFileWithSerial Function

C

```

MCHP_LAN_API BOOL MchpLanProgramFileWithSerial(
    HANDLE LanID,
    UINT AccessType,
    PCHAR InputFileName,
    PCHAR SerialNumber,

```

```
    PCHAR MacAddress
);
```

Devices Supported

LAN7800/LAN7850

Description

This API will program the Input file along with serial number given as argument to the selected LAN device ID.

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
AccessType	1- EEPROM , 2- OTP
InputFileName	Input file to be programmed into the device. Supported file type .bin/.ini
SerialNumber	Serial Number to be programmed
MacAddress	Mac Address to be overwritten. If NULL, Mac Address will not be overwritten

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastErr for more details) - for failure

Example

```
typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef HANDLE (*pfMchpLanOpenID) (UINT16,UINT16,UINT8*);
typedef BOOL (*pfMchpLanProgramFileWithSerial) (HANDLE,UINT,PCHAR,PCHAR,PCHAR);
typedef void (*pfMchpEnableLogging) (INT);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanOpenID libMchpLanOpenID;
pfMchpLanProgramFileWithSerial libMchpLanProgramFileWithSerial;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanOpenID = (pfMchpLanOpenID) GetProcAddress ( LoadLib
,"MchpLanOpenID");
libMchpLanProgramFileWithSerial = (pfMchpLanProgramFileWithSerial)
GetProcAddress ( LoadLib ,"MchpLanProgramFileWithSerial");
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
```

```

        cout<<"LAN Count" << hub_count << endl;
        cout<<sztext<<endl;
        cout<<"Selected hub_index is "<< LanID <<endl;
    }
    else
    {
        printf ("zero hubs foundn");
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }

    HANDLE hDevice = INVALID_HANDLE_VALUE;
    BYTE curMacAddr[6]  = {0x00,0x80,0x0F,0x78,0x00,0x01};
    hDevice = libMchpLanOpenID(0x0424,0x7800,curMacAddr);
    if(hDevice == INVALID_HANDLE_VALUE)
    {
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    printf("Device Opened successfulln");

    //Program File
    if (FALSE ==
libMchpLanProgramFileWithSerial(hDevice,1,"7800eep.bin","1234",(PCHAR)curMacAddr))
    {
        dwError = libMchpLanGetLastError(hDevice);
        printf ("Device Program Failed, Error %04xn",dwError);
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    cout << "Device Programmed Successfully n";

```

7.3.3 Register Read/Write

Functions

	Name	Description
≡	MchpLanRegisterRead	This API will Read LAN register given as argument to the specified LAN Device ID
≡	MchpLanRegisterWrite	This API will write Lan register given as argument to the specified LAN Device ID
≡	MchpLanPhyRegisterWrite	This API will write phy register given as argument to the specified LAN Device ID
≡	MchpLanPhyRegisterRead	This API will Read Phy register given as argument to the specified LAN Device ID

7.3.3.1 MchpLanRegisterRead Function

C

```

MCHP_LAN_API BOOL MchpLanRegisterRead(
    HANDLE LanID,
    UINT32 Address,
    UINT32 Length,
    UINT32 * Value
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will Read LAN register given as argument to the specified LAN Device ID

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
Address	LAN register address to read
Length	Length to be read
Value	Pointer to the buffer where data from registers will be stored

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastErr for more details) - for failure

Example

```
typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef HANDLE (*pfMchpLanOpenID) (UINT16,UINT16,UINT8*);
typedef BOOL (*pfMchpLanRegisterRead)(HANDLE,UINT32, UINT32, UINT32*);
typedef void (*pfMchpEnableLogging) (INT);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanOpenID libMchpLanOpenID;
pfMchpLanRegisterRead libMchpLanRegisterRead;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanOpenID = (pfMchpLanOpenID) GetProcAddress ( LoadLib
,"MchpLanOpenID");
libMchpLanRegisterRead =
(pfMchpLanRegisterRead)GetProcAddress(LoadLib,"MchpLanRegisterRead");
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
}
```

```

    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
BYTE curMacAddr[6] = {0x00,0x80,0x0F,0x78,0x00,0x01};
hDevice = libMchpLanOpenID(0x0424,0x7800,curMacAddr);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

UINT32 Data;
if(FALSE == libMchpLanRegisterRead(hDevice,0x080,1,&Data))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Register read Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Register read Successfully n";

```

7.3.3.2 MchpLanRegisterWrite Function

C

```

MCHP_LAN_API BOOL MchpLanRegisterWrite(
    HANDLE LanID,
    UINT32 Address,
    UINT32 Length,
    UINT32 * Value
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will write Lan register given as argument to the specified LAN Device ID

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
Address	LAN register address
Length	Length to be written
Value	Pointer to the buffer containing data to write to registers.

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastError for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef HANDLE (*pfMchpLanOpenID) (UINT16,UINT16,UINT8*);
typedef BOOL (*pfMchpLanRegisterWrite)(HANDLE,UINT32, UINT32, UINT32*);
typedef void (*pfMchpEnableLogging) (INT);

```

```

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanOpenID libMchpLanOpenID;
pfMchpLanRegisterWrite libMchpLanRegisterWrite;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices          = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanOpenID              = (pfMchpLanOpenID) GetProcAddress ( LoadLib
,"MchpLanOpenID");
libMchpLanRegisterWrite =
(pfMchpLanRegisterWrite)GetProcAddress(LoadLib,"MchpLanRegisterWrite");
libMchpEnableLogging          = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs found\n");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
BYTE curMacAddr[6] = {0x00,0x80,0x0F,0x78,0x00,0x01};
hDevice = libMchpLanOpenID(0x0424,0x7800,curMacAddr);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

UINT32 Data = 0x00000001;
if(FALSE == libMchpLanRegisterWrite(hDevice,0x030,1,&Data))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Register write Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Register write Successfully n";

```

7.3.3.3 MchpLanPhyRegisterWrite Function

C

```
MCHP_LAN_API BOOL MchpLanPhyRegisterWrite(
    HANDLE LanID,
    UINT32 Address,
    UINT32 Length,
    UINT16 * Value
);
```

Devices Supported

LAN7800/LAN7850

Description

This API will write phy register given as argument to the specified LAN Device ID

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
Address	Phy register address
Length	Length to be written
Value	Pointer to the buffer containing data to write to registers.

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastErr for more details) - for failure

Example

```
typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef HANDLE (*pfMchpLanOpenID) (UINT16,UINT16,UINT8*);
typedef BOOL (*pfMchpLanPhyRegisterWrite)(HANDLE,UINT32, UINT32, UINT16*);
typedef void (*pfMchpEnableLogging) (INT);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanOpenID libMchpLanOpenID;
pfMchpLanPhyRegisterWrite libMchpLanPhyRegisterWrite;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanOpenID = (pfMchpLanOpenID) GetProcAddress ( LoadLib
,"MchpLanOpenID");
libMchpLanPhyRegisterWrite =
(pfMchpLanPhyRegisterWrite)GetProcAddress(LoadLib,"MchpLanPhyRegisterWrite");
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress (
```



```

LoadLib, "MchpEnableLogging");

//Get List of LAN Devices
int hub_count = 0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
BYTE curMacAddr[6] = {0x00,0x80,0x0F,0x78,0x00,0x01};
hDevice = libMchpLanOpenID(0x0424,0x7800,curMacAddr);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");

UINT16 Data = 0x00000000;
if(FALSE == libMchpLanRegisterWrite(hDevice,0x0,1,&Data))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Phy Register write Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Phy Register write Successfully n";

```

7.3.3.4 MchpLanPhyRegisterRead Function

C

```

MCHP_LAN_API BOOL MchpLanPhyRegisterRead(
    HANDLE LanID,
    UINT32 Address,
    UINT32 Length,
    UINT16 * Value
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will Read Phy register given as argument to the specified LAN Device ID

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device

Address	Phy register address to read
Length	Length to be read
Value	Pointer to the buffer where data from registers will be stored

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastErr for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef HANDLE (*pfMchpLanOpenID) (UINT16,UINT16,UINT8*);
typedef BOOL (*pfMchpLanPhyRegisterRead)(HANDLE,UINT32, UINT32, UINT16*);
typedef void (*pfMchpEnableLogging) (INT);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanOpenID libMchpLanOpenID;
pfMchpLanPhyRegisterRead libMchpLanPhyRegisterRead;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices          = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanOpenID              = (pfMchpLanOpenID) GetProcAddress ( LoadLib
,"MchpLanOpenID");
libMchpLanPhyRegisterRead =
(pfMchpLanPhyRegisterRead)GetProcAddress(LoadLib,"MchpLanPhyRegisterRead");
libMchpEnableLogging         = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
BYTE curMacAddr[6]  = {0x00,0x80,0x0F,0x78,0x00,0x01};
hDevice = libMchpLanOpenID(0x0424,0x7800,curMacAddr);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

```

printf("Device Opened successfullyn");

UINT16 Data;
if(FALSE == libMchpLanPhyRegisterRead(hDevice,0x01,1,&Data))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Phy Register read Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Phy Register read Successfully n";

```

7.3.4 Memory Read/Write APIs

Functions

	Name	Description
≡	MchpLanReadMemory	This API will read OTP/EEPROM memory for specified LAN Device ID
≡	MchpLanReadEepromByte	This API will Read value from specified EEPROM address
≡	MchpLanWriteEepromByte	This API will Write value in the specified EEPROM address

7.3.4.1 MchpLanReadMemory Function

C

```

MCHP_LAN_API BOOL MchpLanReadMemory(
    HANDLE LanID,
    UINT AccessType,
    UINT8 * Readbuffer,
    UINT16 Address,
    UINT nSize
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will read OTP/EEPROM memory for specified LAN Device ID

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
AccessType	1- EEPROM , 2- OTP
Address	Start Address of memory
nSize	Number of bytes to read

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastError for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef HANDLE (*pfMchpLanOpenID) (UINT16,UINT16,UINT8*);
typedef BOOL (*pfMchpLanReadMemory) (HANDLE,UINT,UINT8*,UINT16,UINT);
typedef void (*pfMchpEnableLogging) (INT);

```

```

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanOpenID libMchpLanOpenID;
pfMchpLanReadMemory libMchpLanReadMemory;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices      = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanOpenID          = (pfMchpLanOpenID) GetProcAddress ( LoadLib
,"MchpLanOpenID");
libMchpLanReadMemory = (pfMchpLanReadMemory)GetProcAddress(LoadLib,"MchpLanReadMemory");
libMchpEnableLogging      = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs found\n");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
BYTE curMacAddr[6] = {0x00,0x80,0x0F,0x78,0x00,0x01};
hDevice = libMchpLanOpenID(0x0424,0x7800,curMacAddr);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

UINT8 *byReadData = (UINT8 *) malloc(512);
if(FALSE == libMchpLanReadMemory(hDevice,1,(UINT8 *)byReadData,0x0,256))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Read memory Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Read Memory Successfully n";

```

7.3.4.2 MchpLanReadEepromByte Function

C

```
MCHP_LAN_API BOOL MchpLanReadEepromByte(
    HANDLE LanID,
    UINT16 Address,
    UINT8* Value
);
```

Devices Supported

LAN7800/LAN7850

Description

This API will Read value from specified EEPROM address

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
Address	EEPROM Address
Value	Pointer to the Buffer which contains the read data to be stored.

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastError for more details) - for failure

Example

```
typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanReadEepromByte)(HANDLE,UINT16, UINT8*);
typedef void (*pfMchpEnableLogging) (INT);
typedef HANDLE (*pfMchpLanOpen) (UINT32);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpLanOpen libMchpLanOpen;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanReadEepromByte libMchpLanReadEepromByte;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanReadEepromByte =
(pfMchpLanReadEepromByte)GetProcAddress(LoadLib,"MchpLanReadEepromByte");
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");
libMchpLanOpen = (pfMchpLanOpen) GetProcAddress ( LoadLib ,"MchpLanOpen");

//Get List of LAN Devices
int hub_count =0;
```

```

hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
hDevice = libMchpLanOpen(LanID);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");

UINT8 Val;
if(FALSE == libMchpLanReadEepromByte(hDevice,0x00,&Val))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Read EEPROM Byte Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout<<"Read EEPROM Byte: Successn";

```

7.3.4.3 MchpLanWriteEepromByte Function

C

```

MCHP_LAN_API BOOL MchpLanWriteEepromByte(
    HANDLE LanID,
    UINT16 Address,
    UINT8* Value
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will Write value in the specified EEPROM address

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
Address	EEPROM Address
Value	Pointer to the Buffer to be written

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastErr for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanWriteEepromByte)(HANDLE, UINT16, UINT8*);
typedef void (*pfMchpEnableLogging) (INT);
typedef HANDLE (*pfMchpLanOpen) (UINT32);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpLanOpen libMchpLanOpen;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanWriteEepromByte libMchpLanWriteEepromByte;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices      = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanWriteEepromByte =
(pfMchpLanWriteEepromByte)GetProcAddress(LoadLib,"MchpLanWriteEepromByte");
libMchpEnableLogging      = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");
libMchpLanOpen            = (pfMchpLanOpen) GetProcAddress ( LoadLib ,"MchpLanOpen");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs found\n");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
hDevice = libMchpLanOpen(LanID);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

UINT8 Val;
Val = 0x71;
if(FALSE == libMchpLanWriteEepromByte(hDevice,0x06,&Val))
{
    dwError = libMchpLanGetLastError(hDevice);
}

```

```

printf ("Write EEPROM Byte Failed, Error %04xn",dwError);
printf ("nPress any key to exit...");
_getch ();
exit (1);
}
cout<<"Write EEPROM Byte: Successn";

```

7.3.5 Miscellaneous APIs

Functions

	Name	Description
≡	MchpLanVerifyParameters	This API will Verify hub Configuration parameters on the selected device ID.
≡	MchpLanGetDriverVersion	This API will get driver version.
≡	MchpLanReadUsbPhyStat	This API will Read Phy Status
≡	MchpLanReadTxRxStat	This API will Read Tx Rx Status
≡	MchpLanOtpConfigureCommand	This API will send command to configure OTP
≡	MchpLanGetLastErr	This API will get last error occurred when handling other API's in this library.
≡	MchpLanGetEepromSize	This API will get size of EEPROM
≡	MchpLanEepromConfigureCommand	This API will send command to configure EEPROM
≡	MchpLanEraseEeprom	This API will erase EEPROM content
≡	MchpLanGetAdapterBits	This API will get adapter bits.
≡	MchpLanGetAdapterKey	This API will get adapter key.
≡	MchpLanGetAdapterPDO	This API will get adapter PDO.
≡	MchpLanGetAdapterPort	This API will get adapter Port.

7.3.5.1 MchpLanVerifyParameters Function

C

```

MCHP_LAN_API BOOL MchpLanVerifyParameters(
    HANDLE LanID,
    PCHAR Widgets,
    PCHAR Values
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will Verify hub Configuration parameters on the selected device ID.

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
Widgets	Pointer to the buffer containing hub configurations parameters
Values	Pointer to the buffer containing hub configurations parameters

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastErr for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef HANDLE (*pfMchpLanOpenID) (UINT16,UINT16,UINT8*);
typedef BOOL (*pfMchpLanVerifyParameters)(HANDLE,PCHAR,PCHAR);
typedef void (*pfMchpEnableLogging) (INT);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanOpenID libMchpLanOpenID;
pfMchpLanVerifyParameters libMchpLanVerifyParameters;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices          = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanOpenID              = (pfMchpLanOpenID) GetProcAddress ( LoadLib
,"MchpLanOpenID");
libMchpLanVerifyParameters =
(pfMchpLanVerifyParameters)GetProcAddress(LoadLib,"MchpLanVerifyParameters");
libMchpEnableLogging         = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
BYTE curMacAddr[6]  = {0x00,0x80,0x0F,0x78,0x00,0x01};
hDevice = libMchpLanOpenID(0x0424,0x7800,curMacAddr);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

char widget_buffer_names[4098];
char widget_buffer_values[4098];
memset(widget_buffer_names,0x00,sizeof(widget_buffer_names));

```

```

string verifyParams [] = {"langid",
                          "usb2vid",
                          "usb2pid",
                          "END"};
for (int i=0;verifyParams[i] != "END"; i++)
{
    strcat(widget_buffer_names,verifyParams[i].c_str());
    strcat(widget_buffer_names,",");
}

if(FALSE ==
libMchpLanVerifyParameters(hDevice,widget_buffer_names,widget_buffer_values))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Verify Parameters Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout << "Verify parameters Successful n";

```

7.3.5.2 MchpLanGetDriverVersion Function

C

```

MCHP_LAN_API BOOL MchpLanGetDriverVersion(
    void * Buffer
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will get driver version.

Preconditions

[MchpLanGetDevices](#) API should be called before calling this API

Parameters

Parameters	Description
buffer	Pointer to the buffer contains driver version to be stored

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastError for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanGetDriverVersion)(void*);
typedef void (*pfMchpEnableLogging) (INT);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanGetDriverVersion libMchpLanGetDriverVersion;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loadedn");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

```

    }

    libMchpLanGetDevices          = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
, "MchpLanGetDevices");
    libMchpLanGetDriverVersion =
(pfMchpLanGetDriverVersion)GetProcAddress(LoadLib, "MchpLanGetDriverVersion");
    libMchpEnableLogging          = (pfMchpEnableLogging) GetProcAddress (
LoadLib, "MchpEnableLogging");

    //Get List of LAN Devices
    int hub_count =0;
    hub_count = libMchpLanGetDevices(sztext);

    if(hub_count)
    {
        cout<<"Connected LAN Devices are ..nn";
        cout<<"LAN Count" << hub_count << endl;
        cout<<sztext<<endl;
        cout<<"Selected hub_index is "<< LanID <<endl;
    }
    else
    {
        printf ("zero hubs foundn");
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }

    char DisplayBuffer[256];
    if(FALSE == libMchpLanGetDriverVersion(DisplayBuffer))
    {
        printf ("Failed to Get driver version");
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    cout <<"GetDriverVersion: Successn";

```

7.3.5.3 MchpLanReadUsbPhyStat Function

C

```

MCHP_LAN_API BOOL MchpLanReadUsbPhyStat(
    HANDLE LanID,
    void * Buffer
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will Read Phy Status

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
buffer	Pointer to the buffer contains Phy status to be stored

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastErr for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanReadUsbPhyStat)(HANDLE, void*);
typedef void (*pfMchpEnableLogging) (INT);
typedef HANDLE (*pfMchpLanOpen) (UINT32);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpLanOpen libMchpLanOpen;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanReadUsbPhyStat libMchpLanReadUsbPhyStat;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices      = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanReadUsbPhyStat =
(pfMchpLanReadUsbPhyStat)GetProcAddress(LoadLib,"MchpLanReadUsbPhyStat");
libMchpEnableLogging      = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");
libMchpLanOpen            = (pfMchpLanOpen) GetProcAddress ( LoadLib ,"MchpLanOpen");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs found\n");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
hDevice = libMchpLanOpen(LanID);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

char DisplayBuffer[256];
if(FALSE == libMchpLanReadUsbPhyStat(hDevice,DisplayBuffer))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Read Phy Status Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

```

}
cout<<"ReadPhyStatus: Successn";

```

7.3.5.4 MchpLanReadTxRxStat Function

C

```

MCHP_LAN_API BOOL MchpLanReadTxRxStat (
    HANDLE LanID,
    void * Buffer
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will Read Tx Rx Status

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
buffer	Pointer to the buffer contains Tx and Rx status to be stored

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastErr for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanReadTxRxStat) (HANDLE, void*);
typedef void (*pfMchpEnableLogging) (INT);
typedef HANDLE (*pfMchpLanOpen) (UINT32);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpLanOpen libMchpLanOpen;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanReadTxRxStat libMchpLanReadTxRxStat;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loadedn");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
, "MchpLanGetDevices");
libMchpLanReadTxRxStat =
(pfMchpLanReadTxRxStat)GetProcAddress(LoadLib, "MchpLanReadTxRxStat");
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress (
LoadLib, "MchpEnableLogging");
libMchpLanOpen = (pfMchpLanOpen) GetProcAddress ( LoadLib , "MchpLanOpen");

//Get List of LAN Devices
int hub_count =0;

```

```

hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
hDevice = libMchpLanOpen(LanID);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");

char DisplayBuffer[256];
if(FALSE == libMchpLanReadTxRxStat(hDevice,DisplayBuffer))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Read Rx Tx Status Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout<<"ReadTxRxStat: Successn";

```

7.3.5.5 MchpLanOtpConfigureCommand Function

C

```

MCHP_LAN_API BOOL MchpLanOtpConfigureCommand(
    HANDLE LanID,
    UINT32 command
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will send command to configure OTP

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
command	OTP command

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastError for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanOtpConfigureCommand)(HANDLE, UINT32);
typedef void (*pfMchpEnableLogging) (INT);
typedef HANDLE (*pfMchpLanOpen) (UINT32);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpLanOpen libMchpLanOpen;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanOtpConfigureCommand libMchpLanOtpConfigureCommand;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices          = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanOtpConfigureCommand = (pfMchpLanOtpConfigureCommand)GetProcAddress(LoadLib,
"MchpLanOtpConfigureCommand");
libMchpEnableLogging         = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");
libMchpLanOpen               = (pfMchpLanOpen) GetProcAddress ( LoadLib ,"MchpLanOpen");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs found\n");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
hDevice = libMchpLanOpen(LanID);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

if(FALSE == libMchpLanOtpConfigureCommand(hDevice,0x01000000UL))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("OTP configure command Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

```

```
cout<<"OTP Configure command: Successn";
```

7.3.5.6 MchpLanGetLastError Function

C

```
MCHP_LAN_API UINT32 MchpLanGetLastError(
    HANDLE LanID
);
```

Devices Supported

LAN7800/LAN7850

Description

This API will get last error occurred when handling other API's in this library.

Preconditions

None

Parameters

Parameters	Description
LanID	Handle to the device

Returns

APISTATUS Error codes.

Example

```
typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef void (*pfMchpEnableLogging) (INT);
typedef HANDLE (*pfMchpLanOpen) (UINT32);
typedef UINT32 (*pfMchpLanGetLastError)(HANDLE);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpLanOpen libMchpLanOpen;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanGetLastError libMchpLanGetLastError;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loadedn");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanGetLastError = (pfMchpLanGetLastError) GetProcAddress ( LoadLib
,"MchpLanGetLastError");
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");
libMchpLanOpen = (pfMchpLanOpen) GetProcAddress ( LoadLib ,"MchpLanOpen");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
```



```

        cout<<"LAN Count" << hub_count << endl;
        cout<<sztext<<endl;
        cout<<"Selected hub_index is "<< LanID <<endl;
    }
    else
    {
        printf ("zero hubs foundn");
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }

    HANDLE hDevice =  INVALID_HANDLE_VALUE;
    hDevice = libMchpLanOpen(LanID);
    if(hDevice ==  INVALID_HANDLE_VALUE)
    {
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    printf("Device Opened successfulln");

    dwError = libMchpLanGetLastError(hDevice);

```

7.3.5.7 MchpLanGetEepromSize Function

C

```

MCHP_LAN_API BOOL MchpLanGetEepromSize(
    HANDLE LanID,
    INT* E2pSize
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will get size of EEPROM

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
E2pSize	Pointer to which eeprom size to be stored

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastError for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanGetEepromSize)(HANDLE, INT*);
typedef void (*pfMchpEnableLogging) (INT);
typedef HANDLE (*pfMchpLanOpen) (UINT32);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpLanOpen libMchpLanOpen;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanGetEepromSize libMchpLanGetEepromSize;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");

```

```

    if (LoadLib != NULL)
    {
        printf("Loadlib library Loaded\n");
    }
    else
    {
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }

    libMchpLanGetDevices          = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
, "MchpLanGetDevices");
    libMchpLanGetEepromSize =
(pfMchpLanGetEepromSize)GetProcAddress(LoadLib, "MchpLanGetEepromSize");
    libMchpEnableLogging         = (pfMchpEnableLogging) GetProcAddress (
LoadLib, "MchpEnableLogging");
    libMchpLanOpen               = (pfMchpLanOpen) GetProcAddress ( LoadLib , "MchpLanOpen");

    //Get List of LAN Devices
    int hub_count =0;
    hub_count = libMchpLanGetDevices(sztext);

    if(hub_count)
    {
        cout<<"Connected LAN Devices are ..nn";
        cout<<"LAN Count" << hub_count << endl;
        cout<<sztext<<endl;
        cout<<"Selected hub_index is " << LanID <<endl;
    }
    else
    {
        printf ("zero hubs found\n");
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }

    HANDLE hDevice = INVALID_HANDLE_VALUE;
    hDevice = libMchpLanOpen(LanID);
    if(hDevice == INVALID_HANDLE_VALUE)
    {
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    printf("Device Opened successfully\n");

    int e2psize;
    if(FALSE == libMchpLanGetEepromSize(hDevice,&e2psize))
    {
        dwError = libMchpLanGetLastError(hDevice);
        printf ("Failed to get EEPROM Size, Error %04xn",dwError);
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    cout<<"Get EEPROM Size: Successn";

```

7.3.5.8 MchpLanEepromConfigureCommand Function

C

```

MCHP_LAN_API BOOL MchpLanEepromConfigureCommand(
    HANDLE LanID,
    UINT32 command
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will send command to configure EEPROM

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
command	EEPROM command Erase/Write Disable- 0x10000000UL Erase/Write Enable- 0x20000000UL Erase All- 0x60000000UL

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastError for more details) - for failure

Example

```
typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanEepromConfigureCommand)(HANDLE, UINT32);
typedef void (*pfMchpEnableLogging) (INT);
typedef HANDLE (*pfMchpLanOpen) (UINT32);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpLanOpen libMchpLanOpen;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanEepromConfigureCommand libMchpLanEepromConfigureCommand;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loadedn");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
, "MchpLanGetDevices");
libMchpLanEepromConfigureCommand =
(pfMchpLanEepromConfigureCommand)GetProcAddress(LoadLib, "MchpLanEepromConfigureCommand");
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress (
LoadLib, "MchpEnableLogging");
libMchpLanOpen = (pfMchpLanOpen) GetProcAddress ( LoadLib , "MchpLanOpen");

//Get List of LAN Devices
int hub_count = 0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
}
```

```

    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
hDevice = libMchpLanOpen(LanID);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfull\n");

if(FALSE == libMchpLanEepromConfigureCommand(hDevice, E2P_CMD_ERAL))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("EEPROM Configure Command Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout<<"EEPROM Configure Command: Successn";

```

7.3.5.9 MchpLanEraseEeprom Function

C

```

MCHP_LAN_API BOOL MchpLanEraseEeprom(
    HANDLE LanID
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will erase EEPROM content

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastError for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanEraseEeprom)(HANDLE);
typedef void (*pfMchpEnableLogging) (INT);
typedef HANDLE (*pfMchpLanOpen) (UINT32);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpLanOpen libMchpLanOpen;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanEraseEeprom libMchpLanEraseEeprom;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{

```

```

        printf("Loadlib library Loaded\n");
    }
    else
    {
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }

    libMchpLanGetDevices          = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
, "MchpLanGetDevices");
    libMchpLanWriteEepromByte =
(pfMchpLanWriteEepromByte)GetProcAddress(LoadLib, "MchpLanWriteEepromByte");
    libMchpEnableLogging        = (pfMchpEnableLogging) GetProcAddress (
LoadLib, "MchpEnableLogging");
    libMchpLanOpen              = (pfMchpLanOpen) GetProcAddress ( LoadLib , "MchpLanOpen");

    //Get List of LAN Devices
    int hub_count =0;
    hub_count = libMchpLanGetDevices(sztext);

    if(hub_count)
    {
        cout<<"Connected LAN Devices are ..nn";
        cout<<"LAN Count" << hub_count << endl;
        cout<<sztext<<endl;
        cout<<"Selected hub_index is "<< LanID <<endl;
    }
    else
    {
        printf ("zero hubs foundn");
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }

    HANDLE hDevice = INVALID_HANDLE_VALUE;
    hDevice = libMchpLanOpen(LanID);
    if(hDevice == INVALID_HANDLE_VALUE)
    {
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    printf("Device Opened successfull\n");

    if(FALSE == libMchpLanEraseEeprom(hDevice))
    {
        dwError = libMchpLanGetLastError(hDevice);
        printf ("EEPROM Erase Failed, Error %04xn",dwError);
        printf ("nPress any key to exit....");
        _getch ();
        exit (1);
    }
    cout<<"EEPROM Erase: Successn";

```

7.3.5.10 MchpLanGetAdapterBits Function

C

```

MCHP_LAN_API BOOL MchpLanGetAdapterBits(
    void * Buffer
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will get adapter bits

Preconditions

[MchpLanGetDevices](#) API should be called before calling this API

Parameters

Parameters	Description
buffer	Pointer to the buffer contains adapter bits to be stored

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastErr for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanGetAdapterBits)(void*);
typedef void (*pfMchpEnableLogging) (INT);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanGetAdapterBits libMchpLanGetAdapterBits;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
,"MchpLanGetDevices");
libMchpLanGetAdapterBits =
(pfMchpLanGetAdapterBits)GetProcAddress(LoadLib,"MchpLanGetAdapterBits");
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress (
LoadLib,"MchpEnableLogging");

//Get List of LAN Devices
int hub_count =0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

char DisplayBuffer[256];
if(FALSE == libMchpLanGetAdapterBits(DisplayBuffer))
{
    printf ("Failed to Get Adapter bits");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout <<"GetAdapterBits: Successn";

```

7.3.5.11 MchpLanGetAdapterKey Function

C

```
MCHP_LAN_API BOOL MchpLanGetAdapterKey(
    HANDLE LanID,
    void * Buffer
);
```

Devices Supported

LAN7800/LAN7850

Description

This API will get adapter key

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
buffer	Pointer to the buffer contains adapter key to be stored

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastErr for more details) - for failure

Example

```
typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanGetAdapterKey) (HANDLE, void*);
typedef void (*pfMchpEnableLogging) (INT);
typedef HANDLE (*pfMchpLanOpen) (UINT32);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanGetAdapterKey libMchpLanGetAdapterKey;
pfMchpLanOpen libMchpLanOpen;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
, "MchpLanGetDevices");
libMchpLanGetAdapterKey =
(pfMchpLanGetAdapterKey)GetProcAddress(LoadLib, "MchpLanGetAdapterKey");
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress (
LoadLib, "MchpEnableLogging");
libMchpLanOpen = (pfMchpLanOpen) GetProcAddress ( LoadLib , "MchpLanOpen");

//Get List of LAN Devices
int hub_count = 0;
hub_count = libMchpLanGetDevices(sztext);
```

```

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
HANDLE hDevice = INVALID_HANDLE_VALUE;
hDevice = libMchpLanOpen(LanID);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");

char DisplayBuffer[256];
if(FALSE == libMchpLanGetAdapterKey(hDevice,DisplayBuffer))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Get adapter key Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout <<"GetAdapterKey: Successn";

```

7.3.5.12 MchpLanGetAdapterPDO Function

C

```

MCHP_LAN_API BOOL MchpLanGetAdapterPDO(
    HANDLE LanID,
    void * Buffer
);

```

Devices Supported

LAN7800/LAN7850

Description

This API will get adapter PDO

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
buffer	Pointer to the buffer contains adapter PDO to be stored

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastError for more details) - for failure

Example

```

typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanGetAdapterPDO)(HANDLE, void*);

```



```

typedef void (*pfMchpEnableLogging) (INT);
typedef HANDLE (*pfMchpLanOpen) (UINT32);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpLanOpen libMchpLanOpen;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanGetAdapterPDO libMchpLanGetAdapterPDO;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
, "MchpLanGetDevices");
libMchpLanGetAdapterPDO =
(pfMchpLanGetAdapterPDO)GetProcAddress(LoadLib, "MchpLanGetAdapterPDO");
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress (
LoadLib, "MchpEnableLogging");
libMchpLanOpen = (pfMchpLanOpen) GetProcAddress ( LoadLib , "MchpLanOpen");

//Get List of LAN Devices
int hub_count = 0;
hub_count = libMchpLanGetDevices(sztext);

if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
hDevice = libMchpLanOpen(LanID);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");

char DisplayBuffer[256];
if(FALSE == libMchpLanGetAdapterPort(hDevice,DisplayBuffer))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Get adapter port Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout <<"GetAdapterPort: Successn";

```

7.3.5.13 MchpLanGetAdapterPort Function

C

```
MCHP_LAN_API BOOL MchpLanGetAdapterPort(
    HANDLE LanID,
    void * Buffer
);
```

Devices Supported

LAN7800/LAN7850

Description

This API will get adapter port

Preconditions

[MchpLanGetDevices](#) and [MchpLanOpen](#) API should be called before calling this API (or) [MchpLanOpenID](#) should be called before calling this API

Parameters

Parameters	Description
LanID	Handle to the device
buffer	Pointer to the buffer contains adapter port to be stored

Returns

TRUE - for Success; FALSE - (Call GetMchpLanLastError for more details) - for failure

Example

```
typedef INT (*pfMchpLanGetDevices) (PCHAR);
typedef BOOL (*pfMchpLanGetAdapterPort)(HANDLE, void*);
typedef void (*pfMchpEnableLogging) (INT);
typedef HANDLE (*pfMchpLanOpen) (UINT32);

pfMchpLanGetDevices libMchpLanGetDevices;
pfMchpEnableLogging libMchpEnableLogging;
pfMchpLanGetAdapterPort libMchpLanGetAdapterPort;
pfMchpLanOpen libMchpLanOpen;

//Load library
HMODULE LoadLib = LoadLibrary ("pt2lib.dll");
if (LoadLib != NULL)
{
    printf("Loadlib library Loaded\n");
}
else
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

libMchpLanGetDevices = (pfMchpLanGetDevices) GetProcAddress ( LoadLib
, "MchpLanGetDevices");
libMchpLanGetAdapterPort =
(pfMchpLanGetAdapterPort)GetProcAddress(LoadLib, "MchpLanGetAdapterPort");
libMchpLanOpen = (pfMchpLanOpen) GetProcAddress ( LoadLib , "MchpLanOpen");
libMchpEnableLogging = (pfMchpEnableLogging) GetProcAddress (
LoadLib, "MchpEnableLogging");

//Get List of LAN Devices
int hub_count = 0;
hub_count = libMchpLanGetDevices(sztext);
```

```
if(hub_count)
{
    cout<<"Connected LAN Devices are ..nn";
    cout<<"LAN Count" << hub_count << endl;
    cout<<sztext<<endl;
    cout<<"Selected hub_index is "<< LanID <<endl;
}
else
{
    printf ("zero hubs foundn");
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}

HANDLE hDevice = INVALID_HANDLE_VALUE;
hDevice = libMchpLanOpen(LanID);
if(hDevice == INVALID_HANDLE_VALUE)
{
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
printf("Device Opened successfulln");

char DisplayBuffer[256];
if(FALSE == libMchpLanGetAdapterPort(hDevice,DisplayBuffer))
{
    dwError = libMchpLanGetLastError(hDevice);
    printf ("Get adapter port Failed, Error %04xn",dwError);
    printf ("nPress any key to exit....");
    _getch ();
    exit (1);
}
cout <<"GetAdapterPort: Successn";
```

8 Error Codes

C

```
enum _APISTATUS {
    Error_Success = 0x0000,
    Error_Device_Not_Found = 0x0001,
    Error_Invalid_Argument = 0x0002,
    Error_Invalid_Device_Handle = 0x0003,
    Error_WinUSBAPI_Fail = 0x0004,
    Error_Required_Reboot = 0x0005,
    Error_Install_FilterDriver = 0x0006,
    Error_Success_Required_Reboot = 0x0007,
    Error_Invalid_BinFile_Size = 0x0008,
    Error_Read_Cfg_File = 0x0009,
    Error_Code_Running_From_SPI = 0x0010,
    Error_Install_Winusb_Driver = 0x0011,
    Error_Invalid_Arguments = 0x0012,
    Error_No_Class_Filter = 0x0013,
    Error_No_Admin_Privilege = 0x0014,
    Error_In_Device_Power_State = 0x0015,
    Error_Not_Supported = 0x0016,
    Error_Maximum_Size = 0x0017,
    Error_OTP_Content_Match_Fail = 0x0018,
    Loadbinfail = 0x1000,
    ReadBackFailed = 0x1001,
    WrongFileSize = 0x1002,
    SpiPassThruWriteFailed = 0x1003,
    SpiPassThruEnterFailed = 0x1004,
    SpiNoDevice = 0x1005,
    SpiCancelDl = 0x1006,
    SpiProgrammingFailed = 0x1007,
    SpiPassThruExitFailed = 0x1008,
    SpiPassThruReadFailed = 0x1009,
    SpiFlashWrongDeviceID = 0x100A,
    SpiFWCompareFailed = 0x100B,
    SpiOpenEraseSigFileFailed = 0x100C,
    SpiReadEraseSigFileFailed = 0x100D,
    SpiSRAMProgFailed = 0x100E,
    SpiEraseSignatureFailed = 0x100F,
    SpiChipEraseFailed = 0x1010,
    LoadJsonfail = 0x1011,
    Loadinifail = 0x1012,
    FlexRegnotprogrammed = 0x1013,
    SpiFlashAccessNotSupported = 0x1014,
    Error_I2C_Passthrough_Enter_Cmd_Failed = 0x2000,
    Error_I2C_Passthrough_Exit_Cmd_Failed = 0x2001,
    Error_I2C_Transfer_Cmd_Failed = 0x2002,
    Error_I2C_Max_Size_Error = 0x2003,
    Error_OTP_Invalid_BufferSize = 0x3000,
    Error_UART_BaudrateErrorPercentTooHigh = 0x4000,
    Error_UART_Set_Regs = 0x4001,
    Error_UART_PC_To_Device_Sent0Bytes = 0x4002,
    Error_UART_PC_To_Device_Failed = 0x4003,
    Error_UART_Rx_Buffer_Overflow = 0x4004,
    Error_UART_RxFIFO_Status_Unexpected = 0x4005,
    Error_UART_RxThread_Creation_Failed = 0x4006,
    Error_UART_Rx_Pending = 0x4007,
    Error_UART_Rx_UserAbort = 0x4008,
    Error_UART_Rx_Command_Failed = 0x4009,
    Error_UART_Rx_0_Bytes = 0x400A,
    Error_UART_Rx_TimeOut = 0x400B,
    Error_GPIO_Invalid_PinNumber = 0x5000,
    Error_LAN78XX_Access_Denied = 0x6000,
    Error_LAN78XX_Adapter_Busy = 0x6001,
    Error_LAN78XX_Fail = 0x6002,
    Error_LAN78XX_EEPROM_Absent_OTP_Fresh = 0x6003,
```

```

Error_LAN78XX_EEPROM_Absent_OTP_FreeSpace = 0x6004,
Error_LAN78XX_EEPROM_Absent_OTP_NoFreeSpace = 0x6005,
Error_LAN78XX_EEPROM_Present_OTP_NoFreeSpace = 0x6006,
Error_LAN78XX_EEPROM_Present_OTP_Fresh = 0x6007,
Error_LAN78XX_EEPROM_Present_OTP_Invalid = 0x6008,
Error_LAN78XX_EEPROM_Absent_OTP_Invalid = 0x6009,
Error_LAN78XX_EEPROM_Absent = 0x600A,
Error_LAN78XX_EEPROM_Present_NotBoot_From_OTP = 0x600B,
Error_Undefined = 0xFFFF
};

```

Description

USB Hub instance error codes.

Identifies the Error codes of the USB Hub instance state

Members

Members	Description
Error_Success = 0x0000	Operation Success
Error_Device_Not_Found = 0x0001	The specific device was not found
Error_Invalid_Argument = 0x0002	Argument passed to the API is invalid
Error_Invalid_Device_Handle = 0x0003	Device handle passed to the API is not valid
Error_WinUSBAPI_Fail = 0x0004	API of the winusb library failed
Error_Required_Reboot = 0x0005	Required system reboot
Error_Install_FilterDriver = 0x0006	Error in installing VSM filter driver
Error_Success_Required_Reboot = 0x0007	Operation success but it requires reboot
Error_Invalid_BinFile_Size = 0x0008	Bin file size is invalid
Error_Read_Cfg_File = 0x0009	Error while reading cfg/bin file
Error_Code_Running_From_SPI = 0x0010	Code is running from SPI
Error_Install_Winusb_Driver = 0x0011	Error in installing winusb driver
Error_Invalid_Arguments = 0x0012	Invalid arguments
Error_No_Class_Filter = 0x0013	Error when VSM Filter is not available
Error_No_Admin_Privilege = 0x0014	error when application does not have admin rights
Error_In_Device_Power_State = 0x0015	Error if VSM command is failed due to power state of the device
Error_Not_Supported = 0x0016	Error if Firmware and configuration file are programmed at one shot for USB253x/USB4604
Error_Maximum_Size = 0x0017	Error if Memory of the device reached maximum size.
Error_OTP_Content_Match_Fail = 0x0018	Error if OTP content programmed does not match with input configuration
Loadbinfail = 0x1000	Could not load the binary file
ReadBackFailed = 0x1001	Reading from SPI flash failed
WrongFileSize = 0x1002	File size did not match
SpiPassThruWriteFailed = 0x1003	SPI pass through write command failed
SpiPassThruEnterFailed = 0x1004	SPI passthru Enter command failed
SpiNoDevice = 0x1005	SPI flash could not be detected or not present
SpiCancelDI = 0x1006	SPI Cancel Download
SpiProgrammingfailed = 0x1007	SPI flash programming failed
SpiPassThruExitFailed = 0x1008	SPI passthru Enter command failed
SpiPassThruReadFailed = 0x1009	SPI pass through read command failed
SpiFlashWrongDeviceID = 0x100A	Unsupported SPI flash detected
SpiFWCompareFailed = 0x100B	SPI flash read back and compare failed with programmed binary
SpiOpenEraseSigFileFailed = 0x100C	Open Erase signature bin file failed
SpiReadEraseSigFileFailed = 0x100D	Read Erase signature bin file failed

SpiSRAMProgFailed = 0x100E	SRAM programming failed
SpiEraseSignatureFailed = 0x100F	SPI_ERASE_4KBSECTOR command failed.
SpiChipEraseFailed = 0x1010	Chip Erase command failed
LoadJsonfail = 0x1011	Could not load the Json file
Loadinifail = 0x1012	Could not load the ini file
FlexRegnotprogrammed = 0x1013	Flex Reg Field was not programmed
SpiFlashAccessNotSupported = 0x1014	SPI Firmware programming not supported for the device
Error_I2C_Passthrough_Enter_Cmd_Failed = 0x2000	Cannot enable I2C Passthrough interface
Error_I2C_Passthrough_Exit_Cmd_Failed = 0x2001	Cannot disable I2C Passthrough interface
Error_I2C_Transfer_Cmd_Failed = 0x2002	I2C Transfer failed
Error_OTP_Invalid_BufferSize = 0x3000	Invalid buffer size
Error_UART_BaudrateErrorPercentTooHigh = 0x4000	Communication at the specified baud rate will be error prone
Error_UART_Set_Regs = 0x4001	Cannot set USB2534 UART registers, probably command failure
Error_UART_PC_To_Device_Sent0Bytes = 0x4002	Transmit failed without transmitting any data
Error_UART_PC_To_Device_Failed = 0x4003	Transmit failed after transmitting some data
Error_UART_Rx_Buffer_Overflow = 0x4004	Receive failed due to buffer overrun, reduce baud rate
Error_UART_RxFIFO_Status_Unexpected = 0x4005	Receive failed due to unexpected Rx FIFO status
Error_UART_RxThread_Creation_Failed = 0x4006	Receive failed since worker thread creation failed
Error_UART_Rx_Pending = 0x4007	UART Rx is pending due to asynchronous mode
Error_UART_Rx_UserAbort = 0x4008	UART receive aborted as per user request
Error_UART_Rx_Command_Failed = 0x4009	UART Receive command failed by the firmware
Error_UART_Rx_0_Bytes = 0x400A	Receive failed without receiving any data
Error_UART_Rx_TimeOut = 0x400B	UART Receive Timeout
Error_GPIO_Invalid_PinNumber = 0x5000	Invalid GPIO PIN Number
Error_LAN78XX_Access_Denied = 0x6000	Cannot access LAN78XX products
Error_LAN78XX_Adapter_Busy = 0x6001	LAN78XX adapter busy
Error_LAN78XX_Fail = 0x6002	Requested LAN operation failed
Error_LAN78XX_EEPROM_Absent_OTP_Fresh = 0x6003	Physical eeprom is absent and OTP is blank
Error_LAN78XX_EEPROM_Absent_OTP_FreeSpace = 0x6004	EEPROM absent and OTP has free space
Error_LAN78XX_EEPROM_Absent_OTP_NoFreeSpace = 0x6005	EEPROM absent and no free space in OTP
Error_LAN78XX_EEPROM_Present_OTP_NoFreeSpace = 0x6006	EEPROM present and no free space in OTP
Error_LAN78XX_EEPROM_Present_OTP_Fresh = 0x6007	EEPROM present and OTP is blank
Error_LAN78XX_EEPROM_Present_OTP_Invalid = 0x6008	EEPROM present and OTP has invalid signature
Error_LAN78XX_EEPROM_Absent_OTP_Invalid = 0x6009	EEPROM absent and OTP has invalid signature
Error_LAN78XX_EEPROM_Absent = 0x600A	EEPROM absent
Error_LAN78XX_EEPROM_Present_NotBoot_From_OTP = 0x600B	EEPROM is present and highest priority goes to EEPROM
Error_Undefined = 0xFFFF	Unknown error occurred

Remarks

None.

9 Frequently Asked Questions

1. Is the library thread safe?

- No. It is not thread safe. Only one Microchip device can be programmed at a time

2. Is it possible to configure the log output (PT2.log) to be more verbose?

-PT2.log levels can be configured using [MchpEnableLogging](#) API

3. Is it possible to disable the log output?

- [Logging](#) is disabled by default

4. Is the library compiled in release or debug mode?

- Library provided was compiled in release mode

5. Is it possible to have the pt2lib.dll compiled in debug mode?

- pt2lib.dll will be provided only in release mode and not in debug mode