Writing Windows Applications for Communication with MX5 HID Device.
## Table of Content

HID Overview...................................................................................................................................3
Creating a Host Application..............................................................................................................3
  Searching for MX5 HID Device ...........................................................................................4
  Reading from MX5 HID Device .......................................................................................6
Getting MX5 HID Device Capabilities ..................................................................................8
Detecting MX5 Device Change on the Bus ............................................................................9
MX5 HID Device Vender and Product ID# ........................................................................10
MX5 HID Output Report Format............................................................................................10
HID Overview

HID stands for “Human Interface Device”. It is general expression for any computer device that helps interaction between a human and a computer, i.e. mouse, keyboard etc. These devices are often referred to as USB-HID because they are connected to the computer through the USB port. There are two entities in the HID protocol, host and device. The focus of this document is creating Windows (host) applications that will communicate with MX5 HID devices using the standard API functions.

Creating a Host Application

This chapter provides code samples for the host application. The code is C / C++ style and uses standard API functions. In order to use these functions the developer needs to install Windows Driver Development Kit (DDK) on their computer. This kit is freely available on the Microsoft website.

Once the Windows DDK is installed, the developer needs to include the DDK input and library directories in their development environment. For example, for Visual studio:

- Click on Tools -> Options
- In the lest pane, under “Projects and Solutions” select “VC++ Directories”
- Under “Show directories for:” select “Include files” and add “<ddk>\inc\ddk” where <ddk> is the directory where the DDK is installed.
- Under “Show directories for:” select “Library files” and add “<ddk>\lib\wxp\i386” where <ddk> is the directory where the DDK is installed.

Finally the developer needs to include <dbt.h>, <setupapi.h> and <hidsdi.h> files in their source code.

Now, we are ready to use the standard API functions to communicate with HID devices through the USB port.

When an HID device is connected to the computer the operating system will do its best to find the appropriate driver for the device. Drivers for MX5 HID devices are available from the CD or you can download it from your vendors website.
Searching for MX5 HID Device

HID device can be connected to or disconnected from the USB port at any time. Host application needs to be able to detect these changes on the USB. Therefore, the first step is to request from operating system to receive messages when a device is attached or removed. This is done by calling API function `RegisterDeviceNotification` and passing the main window handler to it. The following source code does that:

```c
DEV_BROADCAST_DEVICEINTERFACE DevBroadcastDeviceInterface;
HDEVNOTIFY DeviceNotificationHandle;

DevBroadcastDeviceInterface.dbcc_size = sizeof(DevBroadcastDeviceInterface);
DevBroadcastDeviceInterface.dbcc_devicetype = DBT_DEVTYPE_DEVICEINTERFACE;
DevBroadcastDeviceInterface.dbcc_classguid = HidGuid;

DeviceNotificationHandle =
    RegisterDeviceNotification(m_hWnd, &DevBroadcastDeviceInterface, DEVICE_NOTIFY_WINDOW_HANDLE);
```

After this the operating system will send WM_DEVICECHANGE messages when a change occurs on the USB port. The developer needs to register in the message map the function that will be called on WM_DEVICECHANGE message arrival.

```c
// get teh globally unique identifiers for all HID connected to the USB
HidD_GetHidGuid(&HidGuid);

// get the device information handle for all devices in the system
hDevInfo = SetupDiGetClassDevs(&HidGuid, NULL, NULL, DIGCF_PRESENT|DIGCF_INTERFACEDEVICE);

devInfoData.cbSize = sizeof(devInfoData);

// loop through all the devices and look for the one with desired Vendor ID and Product ID
index = 0;

do
{
    // get the handle to a SP_DEVICE_INTERFACE_DATA structure for a detected device.
    moreDevices=SetupDiEnumDeviceInterfaces(hDevInfo, 0, &HidGuid, index, &devInfoData);

    if (moreDevices != 0)
    {
        // device found - get the information about the device
```
// get the length of the SP_DEVICE_INTERFACE_DETAIL_DATA structure
moreDevices = SetupDiGetDeviceInterfaceDetail(hDevInfo, &devInfoData, NULL, 0, &Length, NULL);

// allocate memory for the hDevInfo
detailData = (PSP_DEVICE_INTERFACE_DETAIL_DATA)malloc(Length);

//Set cbSize in the detailData structure.
detailData -> cbSize = sizeof(SP_DEVICE_INTERFACE_DETAIL_DATA);

// get the pointer to SP_DEVICE_INTERFACE_DETAIL_DATA structure
moreDevices = SetupDiGetDeviceInterfaceDetail(hDevInfo, &devInfoData, detailData, Length, &Required, NULL);

// Create a handle to the device.
DeviceHandle=CreateFile(detailData->DevicePath, 0, FILE_SHARE_READ|FILE_SHARE_WRITE, (LPSECURITY_ATTRIBUTES)NULL, OPEN_EXISTING, 0, NULL);

// get the device attributes (we are interested in Vendor ID and Product ID
Attributes.Size = sizeof(Attributes);
moreDevices = HidD_GetAttributes(DeviceHandle, &Attributes);

found = FALSE;
// is this the device we were looking for?
if((Attributes.VendorID == VendorID) && (Attributes.ProductID == ProductID))
{
    // found the HID device we are looking for
    found = TRUE;

    //Get the device's capabilities.
    _HIDP_PREPARED_DATA *PreparsedData;

    // get the pointer to a buffer containing the information about the device's capabilities.
    HidD_GetPreparsedData(DeviceHandle, &PreparsedData);

    // get the capabilities structure
    HidP_GetCaps(PreparsedData, &Capabilities);

    // free the memory
    HidD_FreePreparsedData(PreparsedData);

    // Get a handle for writing Output reports.
    WriteHandle=CreateFile(detailData->DevicePath, GENERIC_WRITE, FILE_SHARE_READ|FILE_SHARE_WRITE, (LPSECURITY_ATTRIBUTES)NULL, OPEN_EXISTING, 0, NULL);
} 
else 
    // Not 'our' device 
    CloseHandle(DeviceHandle);

    // Free the memory used by the detailData structure (no longer needed). 
    free(detailData);
} 
else 
    // SetupDiEnumDeviceInterfaces returned 0, so there are no more devices to check. 
    break;

    index = index + 1;

} while (found == FALSE);

// Free the memory 
SetupDiDestroyDeviceInfoList(hDevInfo);

In the code sample given above we first get the globally unique identifiers for all HID connected to the USB (HidD_GetHidGuid function), then get the device information handle for all devices in the system (SetupDiGetClassDevs function) and finally loop through all the devices and look for the one with desired Vendor ID and Product ID.

**Reading from MX5 HID Device**

Now that we identified all the HID devices on the USB, we can read the data from them. The following code does that. If we have an HID MAG reader the following code would try to read the data from the card swiped through the reader. If there is data (card swiped before the timeout occurred, i.e. before HID_TESTING_TIMEOUT milliseconds) it is going to be stored in ‘pReadBuffer’ (under ‘case WAIT_OBJECT_0:’) and it should be processed there. If a timeout occurs the read operation is canceled (under ‘case WAIT_TIMEOUT:’)

```c
BOOL retBool;
char pReadBuffer[1024];
ZeroMemory(pReadBuffer, 1024);

if (ReadHandle == INVALID_HANDLE_VALUE)
{
   ReadHandle = CreateFile(m_pCurDeviceInfo->strHIDPath, GENERIC_READ,
       FILE_SHARE_READ | FILE_SHARE_WRITE, NULL,
   ```
OPEN_EXISTING FILE_FLAG_OVERLAPPED | FILE_FLAG_NO_BUFFERING NULL);

if (ReadHandle == INVALID_HANDLE_VALUE)
    return;

// prepare for overlapped transfer
if (hEventObject == 0)
{
    hEventObject = CreateEvent(NULL, FALSE, FALSE, NULL);
    DisplayLastError("CreateEvent: ");

    //Set the members of the overlapped structure.
    HIDOverlapped.hEvent = hEventObject;
    HIDOverlapped.Offset = 0;
    HIDOverlapped.OffsetHigh = 0;
}

} // Test the card
retBool = ReadFile(ReadHandle, pReadBuffer, m_pCurDeviceInfo->hidCaps.InputReportByteLength,
    &NumberOfBytesRead, (LPOVERLAPPED) &HIDOverlapped);
if(retBool != 0)
{
    DisplayLastError("ReadFile: ");
}

DWORD retVal = WaitForSingleObject(hEventObject, HID_TESTING_TIMEOUT);
switch(retVal)
{
    case WAIT_OBJECT_0:
    {
        ResetEvent(hEventObject);
        // pReadBuffer contains the data - process it here
        break;
    }
    case WAIT_TIMEOUT:
    {
        //Cancel the Read operation.
        ResetEvent(hEventObject);
        CancelIo(ReadHandle);

        break;
    }
}
BlockOthers(FALSE);
m_txtSwipeInfo = _T("");  
SetButton(_T("&Test Card"));

UpdateData(FALSE);

Getting MX5 HID Device Capabilities

The following code sample demonstrated how to get the device’s capabilities. This code was already used in the section “Searching for MX5 HID Device” but it is discussed in this section separately.

// Get the Capabilities structure for the device.
_HIDP_PREPARSED_DATA *PreparsedData;

// Get a pointer to a buffer containing the information about the device's capabilities.
HidD_GetPreparsedData(DeviceHandle, &PreparsedData);

// Get device's capabilities.
HidP_GetCaps(PreparsedData, &Capabilities);

*******************************************************************************
* Device capabilities structure:
typedef struct _HIDP_CAPS
{
    USAGE Usage;
    USAGE UsagePage;
    USHORT InputReportByteLength;
    USHORT OutputReportByteLength;
    USHORT FeatureReportByteLength;
    USHORT Reserved[17];
USHORT NumberLinkCollectionNodes;
USHORT NumberInputButtonCaps;
USHORT NumberInputValueCaps;
USHORT NumberInputDataIndices;
USHORT NumberOutputButtonCaps;
USHORT NumberOutputValueCaps;
USHORT NumberOutputDataIndices;
USHORT NumberFeatureButtonCaps;
USHORT NumberFeatureValueCaps;
USHORT NumberFeatureDataIndices;
}

// Free the memory
HidD_FreePreparsedData(PreparsedData);

In this code excerpt first the function HidD_GetPreparsedData allocates memory for the device pre-parsed data using a handle previously obtained from the CreateFile function. Then, function HidP_GetCaps is used to get the device’s capabilities. Finally, the memory allocated by HidD_GetPreparsedData is released.

The capabilities structure is given in this code (commented out) just for programmer’s reference.

**Detecting MX5 Device Change on the Bus**

As mentioned before the device can be attached or removed from the USB at any time. Therefore the application needs to register its interest in these changes with the system. Then the operating system will send WM_DEVICECHANGE messages when a change occurs on the USB port. Callback function provided takes two parameters:
- wParam - device path name
- lParam – tehnature of the event

The following line will make that OnMsgDeviceChange function is called whenever the operating system sends WM_DEVICECHANGE message.

```cpp
ON_MESSAGE(WM_DEVICECHANGE, OnMsgDeviceChange)
```

The function can be defined as:

```cpp
OnDeviceChange(WPARAM wParam, LPARAM lParam);
```
The second parameter will contain the path name for the device in question and the first will contain the nature of the event. Possible values for the first parameter are:

```c
#define DBT_DEVICEARRIVAL               0x8000  // system detected a new device
#define DBT_DEVICEQUERYREMOVE           0x8001  // wants to remove, may fail
#define DBT_DEVICEQUERYREMOVEFAILED     0x8002  // removal aborted
#define DBT_DEVICEREMOVEPENDING         0x8003  // about to remove, still avail.
#define DBT_DEVICEREMOVECOMPLETE        0x8004  // device is gone
#define DBT_DEVICETYPESPECIFIC          0x8005  // type specific event
#if(WINVER >= 0x040A)
#define DBT_CUSTOMEVENT                 0x8006  // user-defined event
#endif /* WINVER >= 0x040A */
```

**MX5 HID Device Vendor and Product ID**

The MX5-HID USB device has the following decimal ID numbers:

- **Vendor ID:** 0x0980
- **Product ID:** 0x9025
- **Revision ID:** 0x0100

**MX5 HID Output Report Format**

The MX5 HID card swipe reports 337 ASCII characters in the following format:

**Data Report Format:** `<Count> - <ASCII number> - <character>`

**Data Definition:**

- `<Count>` runs from 0 to 336.
- `<ASCII number>` values range from 0 to 255 (displayed in decimal base).
- `<Character>` is the character that corresponds to the ASCII number; you may see a square or some other odd character if the value is cannot be displayed as a character.

The values represented:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Usage Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Track 1 decode status</td>
</tr>
<tr>
<td>1</td>
<td>Track 2 decode status</td>
</tr>
<tr>
<td>2</td>
<td>Track 3 decode status</td>
</tr>
<tr>
<td>3</td>
<td>Track 1 data length</td>
</tr>
<tr>
<td>4</td>
<td>Track 2 data length</td>
</tr>
<tr>
<td>5</td>
<td>Track 3 data length</td>
</tr>
<tr>
<td>6</td>
<td>Card encoding type</td>
</tr>
<tr>
<td>7 - 116</td>
<td>Track 1 ASCII data (RFID card ID numbers are displayed in track 1)</td>
</tr>
<tr>
<td>117 - 226</td>
<td>Track 2 ASCII data</td>
</tr>
<tr>
<td>227 - 336</td>
<td>Track 3 ASCII data</td>
</tr>
</tbody>
</table>