Table of Contents

About USBMC

This section contains the general information about the library, its installation, licensing and redistribution policies.

It is structurally divided into the following sections:

Introduction

Contains the general information on the USB Monitoring Control library.

Licensing

Contains the information on licensing the library.

Installation

Contains the information on installing the library on the development system.

Distribution

Contains the library redistribution policy.

Activation

Describes the process of activating the USB Monitoring Control.

Introduction

This section contains the general information about the USB Monitoring Control library (USBMC). It is structurally divided into the following sections:

Library Description

The introductory library description.

Library features

Contains the complete list of library features.

Library Description

HHD USB Monitoring library provides USB monitoring functionality for your application. It collects and parses USB packets so that it is very easy to use/parse their content in your application. The library lets you enumerate all installed USB devices (Mass Storage class devices like flash drives, HID devices like mice or keyboards etc). You can attach to any of them and collect/analyze data that is being transferred between this particular device and host (computer). The Monitor object can be attached to the device at any time, no matter if the device is being currently used or not. For example flash drive could be ejected, but it still appears during enumeration.

USBMC, The Library, library, control

The USB Monitoring Control library

User or client code

Code in any language that instantiates the USBMC and calls its methods

The monitored application

The application that has the monitored USB device open.

Library Features

The HHD Software USB Monitoring Control library offers the following functionality:

- Support for PnP and virtual USB devices with hot-plug and hot-unplug functionality.
- Full support for USB 2.0, USB 3.0 and USB 3.1.

- Interception of all data read from and written to the USB device.
- Support for Windows 7, Windows 8, Windows 8.1 and Windows 10 as well as corresponding server editions. Both 32-bit and 64-bit versions are supported.
- Provides two high-performance mechanisms: one for native code (direct COM interfaces) and one for managed code * (automation-compatible interface).
- High compatibility with all modern development environments, including Microsoft Visual Studio 2012 (or later), Embarcadero RAD Studio XE 4 (or later).
- Library client code can be written in any language, including C++, Delphi, C#, VB.NET and any other CLR-compatible language.
- "Layered parsing" technology for native interface allows you to skip several parsing steps in order to optimize performance.

Licensing

This section contains the list of licenses the USB Monitoring Control is distributed under. Please note that the licensing is subject to change. For a most recent list of licenses please visit the HHD Software Web site.

You may find the detailed information on USB Monitoring Control licensing on-line. Click on a link below to read the corresponding license agreement.

License	Single developer	Unlimited number of developers	Distribution right included
Single Developer & Distribution	X		X
Site Developer & Distribution		X	X

Installation

The library is distributed as a single executable file which is signed by the HHD Software, Ltd. You can use the operating system's provided tools to verify the digital signature to make sure the file is delivered unmodified by any third-party and is free of transmission errors.

The library installer, when launched, asks you to provide the library installation path, or to accept the default. You can also choose whether you need the library samples to be installed or not.

After installation, the following structure is created in a destination folder:

- `bin`
  - `hhdusbmc.dll` - USB Monitoring Control ActiveX component
  - `DIFxAPI.dll` - Driver installation framework utility DLL
- `doc`
  - `hhdusbmc.chm` - This documentation
- `drivers`
  - `hhdusbmc.inf` - Driver installation information file
  - `hhdusbmc_x86.cat` - 32-bit driver catalog file
  - `hhdusbmc_x64.cat` - 64-bit driver catalog file
  - `hhdusbmc32.sys` - 32-bit filter driver
  - `hhdusbmc64.sys` - 64-bit filter driver
- `inc`
  - `hhdusbdefs.h` - File with internal driver structures definitions
  - `hhdusbmc.idl` - File with library classes and interfaces
  - `hhdusbmc.h` - "Compiled" version of hhdusbmc.idl file
  - `usb.h` - parts of USB-related declarations, taken from Windows DDK. It is recommended to install Windows DDK and use the file from this kit instead
- `lib`
  - `x64`

        - `hhdusbmc.lib` - 64-bit import library
    - `x86`
        - `hhdusbmc.lib` - 32-bit import library
- `redist`
    - `Manual`
        - `usbmc_redist.exe` - redistributable module
    - `Merge Module`
        - `x64`
            - `usbmc_msm_x64.msm` - 64-bit Windows Installer Merge Module
        - `x86`
            - `usbmc_msm_x86.msm` - 32-bit Windows Installer Merge Module
- `Samples`
    - Library sample solutions and projects

> **NOTE**
> 64-bit device driver cannot be used with 32-bit client code.

Library Redistribution Policy

Library redistribution policy largely depends on the purchased license. Detailed information about library redistribution can be found in the Licensing topic. General information is also provided below.

General Library Distribution Information

The USBMC library can be distributed in two ways:

Distribution of the original USBMC installation package.

You are allowed to distribute the original library installation package provided the following is true: the `hhdusbmc.exe` file is not modified by any means and is accompanied by a link to HHD Software web site. This can be an Internet URL file, a readme file in the same folder or same archive where the `hhdusbmc.exe` file is located on distribution media. If the file is distributed on the Web Site, the link to HHD Software web site must be on the same page as the link to the `hhdusbmc.exe` file.

The `hhdusbmc.exe` file, as all other files available on HHD Software Web Site, is digitally signed. Please verify that the digital signature is still valid before distributing the file. If, on the other side, you somehow received the USB Monitoring Control library and found that signature validation algorithm fails, please be so kind to notify us via our contact page.

Library Redistribution

Two ways of redistributing the library are provided by the current version: manual redistribution and Windows Installer merge module.

Manual Redistribution

To redistribute the library with your application, use the `usbmc_redist.exe` file, located in the `redist\Manual` folder.

This is a Win32 executable without any external references. It supports all operating systems starting from Windows 7 (both 32-bit and 64-bit).

Launch this executable on a target computer to install or uninstall the USBMC library and accompanying filter driver.

It automatically installs 32-bit or 64-bit version of components. Below is executable's command line parameters:

```
PowerShell
usbmc_redist.exe [/q] [/u] [/t <path>] [/?]
```

Option	Description
`/q`	Quiet mode. Do not display any user interface. Note that on some operating systems, driver installation prompt may still be displayed.
`/u`	Uninstall the library. If not specified, performs library installation.
`/t <path>`	Use the given installation path instead of the default one.
`/?`	Display supported command line parameters.

`usbmc_redist.exe` must be launched by a user with administrative privileges. The caller must also be elevated. Library installation may be performed by more restricted user if she has write access to destination folder, has write access to system drivers folder (usually `\Windows\System32\Drivers`) and has been assigned an "Install Device Drivers" privilege.

If `/q` switch is not specified, a message box with a short description of the result of operation is displayed.

Upon exit, usbmc_redist.exe returns an error code, which must be interpreted as described below:

Error code	Meaning
`0` (`S_OK`)	Library installation/uninstallation has been successful.
`1` (`S_FALSE`)	Library installation/uninstallation has been successful. Reboot is required to complete the installation.
any other	Any other code must be treated as a standard `HRESULT` value.

Windows Installer Merge Module

If your application is packaged with Windows Installer, you may use the merge modules provided in the `\redist\Merge Module` folder. Use the correct module for your target platform (32-bit or 64-bit).

Library Activation on Client Computers

Library activation on client computers is performed in your installation code by calling one of the IUsbMonitor.InstallLicense, IUsbMonitor.InstallLicenseInMemory or IUsbMonitor.InstallLicenseInMemoryNative methods.

> **NOTE**
> You are explicitly prohibited from charging any fees for all kinds of library distribution!

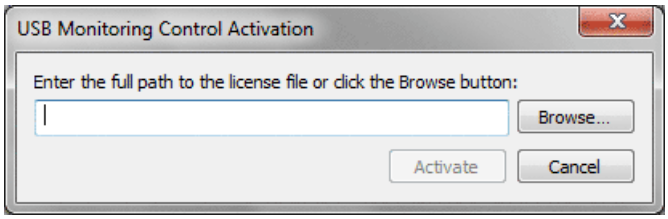
Activating USB Monitoring Control

This section provides a complete description of the steps you need to follow in order to activate your copy of the HHD Software USB Monitoring Control.

First, you need to have an account in our internal database, which means that the product must already be purchased. If you haven't purchased it yet, please follow to the product's homepage.

Manual Activation

Below is the Activation window.



You need to provide a full path to the license file in order to activate the USBMC. You can enter the path manually, or click the Browse button.

Activation from Code

You can also install a license file from code. The following methods are available:

1. IUsbMonitor.InstallLicense, IUsbMonitor.InstallLicenseInMemory and IUsbMonitor.InstallLicenseInMemoryNative methods install a license file contained in a file or in a memory buffer.

2. IUsbMonitor.DisplayActivationDialog method displays an activation dialog, allowing a user to specify the license file to activate the library.

3. The library exports a function with a following signature:

   ```
   C++
   extern "C" void __stdcall DisplayActivationDialogW(HWND hwnd, HINSTANCE hInstance, LPCTST
   ```

   The signature of the function is compatible with `rundll32.exe` utility program. It displays an activation dialog, allowing a user to specify the license file to activate the library.

Using USBMC

Usage Environments

This documentation defines the usage environment as a programming language or development platform, for which it is possible to use the USB Monitoring Control library. Theoretically, there are unlimited languages and platforms that can use the library, because it supports both pure COM and OLE automation interfaces, which makes it highly portable. For clarity, in this documentation we focus on the following two environments:

Native Environment

Native environment means the language that is binary compatible with COM, more precisely, the one that supports COM local interfaces. You can create a native client in C/C++ languages, Borland Delphi and others. As usual, if you are looking for the highest performance and lowest overhead, while keeping your executable as small as possible, the native environment is your right choice.

The native client must load the library as an in-proc component and make sure to use the free threaded marshaller. In fact, the so-called native interfaces in the library require that your runtime does not provide the marshaller for the interface - they are intended to be called directly. INativeListener is the only pure native interface in the library. All other interfaces are dual and can be called through any marshaller (or without one).

Managed Environment

Under managed environment we understand all scripting and similar languages. They include JavaScript, VB script, Visual Basic, Java and the whole .NET platform. All these languages and tools are capable of working with OLE automation-compatible interfaces. The USBMC library supports managed clients, providing the highest possible performance while keeping the library usage as simple as possible.

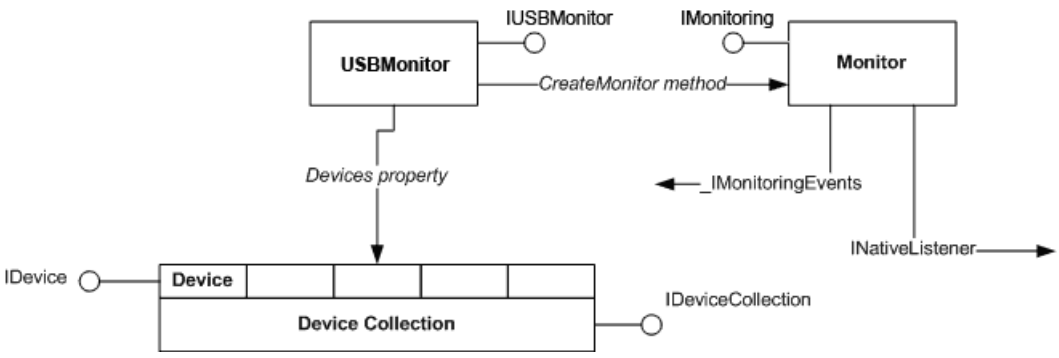
Please note, that you can also create the client in "native" languages, such as C++ or Delphi, that in fact call the managed version of the library interfaces. While this is possible and actually may simplify your code and boost your development time, the performance will be slightly less, compared to the one in pure native environment.

General Guidelines

This section contains the common conventions defined in the library.

Common Information

The library has the following structure:



Native Conventions

Dual Interfaces

IUsbMonitor, IDeviceCollection, IDevice and IMonitoring interfaces are declared dual. To use their binary-compatible (local) part in native code, include the `hhdusbmc.h` file in your project. Remember, that the library must be loaded into your process address space to use the local interfaces without a marshaller.

`INativeListener` Local Interface

The INativeListener interface is a local interface, so it can only be used in the native code. Unlike other interfaces declared in the USBMC, the library does not have any object that implement this interface. Instead, you must implement this interface in your native listener object. Each Monitor object supports adding one or more native listeners via the IMonitoring.AddNativeListener method call.

The monitor object is capable of providing two different protocols for each native listener. Your implementation must support exactly one of these protocols. If you support the first (raw) protocol, you will be able to achieve the highest possible speed, but will have to parse the monitored data in your own code. If you choose to support the second (standard) protocol, the library will parse each monitored request for you and call the appropriate method in the INativeListener interface for each request type.

Time Values

Every standard protocol method in the INativeListener interface accepts the `FILETIME *` value as a first parameter. This parameter points to the library-allocated `FILETIME` structure containing the time of the monitored event. Time is stored in UTC. Do not modify the contents of the variable, treat it as read only.

`INativeListener` Methods Parameters

Each standard protocol method in the INativeListener interface provides one or more parameters which contain or point to the parsed request data. The first parameter is always a pointer to a `FILETIME` structure, containing the UTC time of the event, while others (if present) contain the parsed request data. Consult the documentation for individual methods to get more information.

The general rule states that all data passed to the `INativeListener` methods should be considered read only.

Managed Conventions

Dual Interfaces

All managed environments are capable of using dual interfaces. Unlike the native environments, which use the local, binary compatible part of the dual interface, the managed environment uses the `IDispatch` interface to call methods and query/set properties. IUsbMonitor, IDeviceCollection, IDevice and IMonitoring interfaces are all dual interfaces and can be easily used in managed environments.

Event Interfaces

The USB Monitoring Control library has two event interfaces: _IUsbMonitorEvents and _IMonitoringEvents. Different managed environments support different styles of connecting event handlers to event sources. Please consult your language documentation to find out how to handle OLE automation events. Native clients are also able to attach event handlers to event sources. Please consult the MSDN for more information, or look at the included sample code for an example.

Processing Sequence

This section describes the library's data processing sequence. Whenever a new data packet is captured by the USBMC, the following occurs:

1. Library calls the INativeListener.ProcessRAWBuffer method of each registered native listener. The full data buffer is passed to the listener. A data buffer may contain several URB packets. It is listener's responsibility to separate and process them. If the listener sets the bStopParsing parameter to TRUE, no

further processing is performed on this data buffer by the library.

2. If none of native listeners deny the further processing, raw data is parsed. After that, for each `URB` in the buffer, USBMC calls INativeListener.OnPacketDown or INativeListener.OnPacketUp (_IMonitoringEvents.OnPacketDown or _IMonitoringEvents.OnPacketUp). Native listener can set the `bStopParsing` parameter to `TRUE` to stop further processing of each individual `URB`.

3. USBMC parses the packet and calls the appropriate methods of theINativeListener or _IMonitoringEvents interfaces, depending on the type of the packet.

The following is a pseudo-code of library's data processing pipeline:

C++

```
// pData is a pointer to data chunk, received from driver
void ProcessBuffer(void* pData, DWORD dwDataSize)
{
 // for each client
 for(...)
 {
  // this step is for NATIVE clients only
  pClient->ProcessRawBuffer(...);
 }

 for (USBPACKET* packet=(USBPACKET *) pData, *stop=(USBPACKET*) ((BYTE *) pData + dwSize);
  packet < stop && packet->Size;
        (BYTE*&) packet+=packet->Size)
 {
  // for each client
  for(...)
  {
   if(packet->Flags & UPF_DOWN)
    pClient->OnPacketDown(...);
   else
    pClient->OnPacketUp(...);

   switch(packet->EventType)
   {
   case EVENT_URB:
    {
    // parsing urb here
    USBPACKET_URB* pUrb = (USBPACKET_URB *)pPacket;
    BYTE* pUrbData = ...;
    DWORD dwUrbSize = ...;
    pClient->OnUrb(pUrbData,dwUrbSize);

    switch (pUrb->urb.UrbHeader.Function)
    {
    case URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE:
     pClient->OnGetDescriptorFromDevice();
     break;
    // ...
    }

    break;
    }
   case EVENT_DEVICECONNECTED:
    pClient->OnConnection(...);
    break;
   // ...
   }
  }
 }
};
```

Let's take a [OnGetDescriptorFromInterface] method as an example.

USBMC uses `URB_CONTROL_DESCRIPTOR_REQUEST` structure to get the data from a URB packet. Native listener

has the following signature for `OnGetDescriptorFromInterface` method:

```
C++
HRESULT OnGetDescriptorFromInterface (FILETIME *fTime, void* pData, ULONG Size,
    BYTE Index, BYTE DescriptorType, USHORT LanguageId);
```

`pData` points to the packet that has a type `USBPACKET` and can be cast to `USBPACKET_URB`. So, native client can manually parse this packet and extract fields it needs. Besides USBMC automatically decodes and provides several packet fields, such as `Index`, `DescriptorType` and `LanguageId`.

Manual parsing can be implemented in C++ in this way:

```
C++
USBPACKET_URB* pUrb = (USBPACKET_URB*) pData;
_URB_CONTROL_DESCRIPTOR_REQUEST &r = pUrb->urb.UrbControlDescriptorRequest;
// use r variable here
```CPP

Please note that managed (dispatched) interface has a different method signature, without `pD

```CPP
OnGetDescriptorFromInterface([in] DATE time,
    [in] BYTE Index, [in] BYTE DescriptorType, [in] USHORT LanguageId);
```

Nevertheless, several managed callback methods (such as _IMonitoringEvents.OnPacketDown, _IMonitoringEvents.OnPacketUp and _IMonitoringEvents.OnUrb) receive raw data by means of a safe array, allowing them to manually parse data.

How To

This section contains the step-by-step procedures that will show you how to use the USB Monitoring Control library.

Select one of the following links to find out how to...

- Initialize the USBMC library.
- Enumerate USB Devices.
- Retrieve USB Device Properties.
- Create a monitoring object.
- Receive monitored events.

How to Initialize the USBMC Library

This section describes the steps you need to carry in order to successfully initialize the USBMC library.

Native Environment

1. Include the `hhdusbmc.h` file in one of your source files:

   ```
   C++
   #include <hhdusbmc.h>
   ```

2. Declare the pointer to the IUsbMonitor interface.

   ```
   C++
   CComPtr<IUsbMonitor> pUsbMonitor;
   ```

3. Create the instance of the UsbMonitor object.

```
C++
pUsbMonitor.CoCreateInstance(__uuidof(UsbMonitor));
```

Remember that you cannot create more than one UsbMonitor object in single process. Although, you can have as many Monitor objects as you need.

4. Link to `hhdusbmc.lib` library to eliminate unresolved externals for library GUIDs.

Managed Environment

Use your language-provided tools to add the reference to the USB Monitoring Control library into your project.

How to Enumerate USB Devices

This section describes the steps you need to carry in order to enumerate the USB devices installed on the computer.

1. Initialize the `usbMonitor` object, as described in the this tutorial.
2. Obtain the pointer to the IDeviceCollection interface of the USB device collection object by taking the value of the IUsbMonitor.Devices property:

```
C++
CComPtr<IDeviceCollection> pDeviceCollection;
pUsbMonitor->get_Devices(&pDeviceCollection);
```

```
C#
DeviceCollection devices = sm.Devices;
```

3. Get the value of the IDeviceCollection.Count property:

```
C++
ULONG Count;
pDeviceCollection->get_Count(&Count);
```

```
C#
uint Count = devices.Count;
```

4. Cycle through all items of the collection:

```
C++
for (int i = 0; i < Count; ++i)
{
    CComPtr<IDevice> pDevice;
    pDeviceCollection->get_Item(CComVariant(i), &pDevice);
    // ...
}
```

```
C#
for (int i = 0; i < Count; ++i)
{
    Device device = devices[i];
    // ...
}
```

or

```
C#
foreach (var device in devices)
{
    // ...
}
```

How to Retrieve USB Device Properties

This section describes the steps you need to carry in order to retrieve the properties of the USB device.

1. Obtain the IDevice pointer for the device in question.
2. Take the values of the IDevice.Name, IDevice.Icon, IDevice.Location, IDevice.Manufacturer and IDevice.Key properties.

```
C++
CComBSTR Location, Name, Manufacturer, DeviceKey;

pDevice->get_Location(&Location);
pDevice->get_Name(&Name);
pDevice->get_Manufacturer(&Manufacturer);
pDevice->get_Key(&DeviceKey);

CComVariant bPresent;
pDevice->get_Present(&bPresent.boolVal);
```

```
C#
hhdusbmcLib.Device device = monitor.ConnectedDevice;
string strName = device.Name;
bool bPresent = device.Present;
string strLocation = device.Location;
string strMan = device.Manufacturer;
string strKey = device.Key;
Icon ico  = Icon.FromHandle(device.Icon);
```

How to Create a Monitoring Object

This section describes the steps you need to carry in order to create a monitor object.

1. Initialize the UsbMonitor object, as described in the this tutorial.
2. Call the IUsbMonitor.CreateMonitor method to create a monitor object and receive the IMonitoring interface.

```
C++
CComPtr<IMonitoring> pMonitor;
pUsbMonitor->CreateMonitor(&pMonitor);
```

```
C#
hhdusbmcLib.Monitoring monitor = mc.CreateMonitor();
```

You can create as many monitor objects as you need. Each Monitor object can be attached to one USB device at a time and can have as many native listeners or event handlers attached, as you need.

How to Receive Monitored Events

This section describes the steps you need to carry in order to receive monitored events in your code.

1. Obtain the Monitor object, as described in this tutorial.
2. Add your listener object (for native code) to the Monitor object or connect event handlers (for managed

code):

```
C++
class CMyListener : public CComObjectRoot<CMyListener>, public INativeListener
{
public:
    // override all pure virtual methods in the INativeListener
    STDMETHOD(OnConnection)(FILETIME *,BOOL,LPCTSTR);
    STDMETHOD(OnGetDescriptorFromDevice)(FILETIME *fTime,void* Data,ULONG Size,BYTE Index
    // ...
};

...

CComObject<CMyListener> pMyListenerObject;
CComObject<CMyListener>::CreateInstance(&pMyListenerObject);
CComPtr<INativeListener> pMyListener;
pMyListenerObject->QueryInterface(&pMyListener);
pMonitor->AddNativeListener(pMyListener);
```

```
C#
monitor.OnConnection += new hhdusbmcLib._IMonitoringEvents_OnConnectionEventHandler(monit
monitor.OnPacketUp += new hhdusbmcLib._IMonitoringEvents_OnPacketUpEventHandler(monitor_O
monitor.OnPacketDown += new hhdusbmcLib._IMonitoringEvents_OnPacketDownEventHandler(monit
...
```

3. Attach the Monitor object to the USB device:

   - Attach to the given USB device:

     ```
     C++
     pMonitor->Connect(CComVariant(pDevice), /*headers-only*/ CComVariant(VARIANT_FALSE));
     ```

     ```
     C#
     hhdusbmcLib.Device device  = monitor.Devices[nIndex];
     monitor.Connect(device, /*headers-only*/ false);
     ```

   - Attach to the next USB device the user plugs into the computer:

     ```
     C++
     CComPtr<IDeviceCollection> pDeviceCollection;
     HRESULT hr = m_pUM->get_Devices(&pDeviceCollection);
     if(FAILED(hr))
         return hr;

     CComPtr<IDevice> pDevice;
     // first item is always NextConnected
     pDeviceCollection->get_Item(CComVariant(0),&pDevice);

     pMonitor->Connect(pDevice, /*headers-only*/ CComVariant(VARIANT_FALSE));
     ```

     ```
     C#
     // first device is always NextConnected device
     hhdusbmcLib.Device device  = monitor.Devices[0];
     monitor.Connect(device, /*headers-only*/ false);
     ```

Reference

Interfaces

IDevice Interface

Description

This interface is implemented by the USB Monitoring library for each registered USB device. Use this interface to query the USB device properties.

Declaration

TypeScript

```typescript
interface IDevice extends IDispatch {
    // Properties
    readonly Manufacturer: string;
    readonly Location: string;
    readonly Key: string;
    readonly Present: boolean;
    readonly Name: string;
}
```

C#

```csharp
public interface IDevice : IDispatch
{
    // Properties
    string Manufacturer { get; }
    string Location { get; }
    string Key { get; }
    IntPtr Icon { get; }
    bool Present { get; }
    string Name { get; }
}
```

C++

```cpp
struct IDevice : IDispatch
{
    // Properties
    _bstr_t Manufacturer;  // get
    _bstr_t Location;  // get
    _bstr_t Key;  // get
    HICON Icon;  // get
    VARIANT_BOOL Present;  // get
    _bstr_t Name;  // get
};
```

IDevice Properties

Manufacturer

TypeScript

```typescript
readonly Manufacturer: string;
```

C#

```csharp
string Manufacturer { get; }
```

C++

```cpp
_bstr_t Manufacturer;  // get
```

Description

Device manufacturer string.

Location

```typescript
readonly Location: string;
```

```csharp
string Location { get; }
```

```cpp
_bstr_t Location;  // get
```

Description

Device location string.

Key

```typescript
readonly Key: string;
```

```csharp
string Key { get; }
```

```cpp
_bstr_t Key;  // get
```

Description

Device key. For example, `\??\usb#vid_0458&pid_003a#5&35eda8e7&0&2#{889bf6d2-b5a9-42af-9364-dcc1b727885b}`

Icon

```typescript
// This property is not available in scripting environment
```

```csharp
IntPtr Icon { get; }
```

```cpp
HICON Icon;  // get
```

Description

Handle for device icon.

Present

```typescript
readonly Present: boolean;
```

```
C#
bool Present { get; }
```

```
C++
VARIANT_BOOL Present;  // get
```

Description

This property equals `true` if device is ready and running and `false` otherwise. Note that USBMC enumerates all devices - both connected and disconnected. For example - if you will remove Flash Drive - it will still be enumerated, but `Present` property will be set to `false`.

Name

```
TypeScript
readonly Name: string;
```

```
C#
string Name { get; }
```

```
C++
_bstr_t Name;  // get
```

Description

Device friendly name. For example, `USB Human Interface Device`.

IUsbMonitor Interface

Description

This is the first interface you get from the USBMC library. It is used to get the list of installed USB devices and create Monitor objects. You get a reference to this interface when you create the `UsbMonitor` object. It is a central entry point to the monitoring library.

```
C++
CComPtr<IUsbMonitor> pUM;
HRESULT hr = pUM.CoCreateInstance(__uuidof(UsbMonitor));
if (FAILED(hr))
{
    MessageBox(L"Error opening the USBMC control", L"Error", MB_ICONSTOP | MB_OK);
    return TRUE;
}
// use pUM here
```

```
C#
hhdusbmcLib.UsbMonitor mc = new hhdusbmcLib.UsbMonitor();
// use mc here
```

Declaration

```
TypeScript
interface IUsbMonitor extends IDispatch {
    // Properties
    readonly Devices: IDeviceCollection;
    // Methods
    CreateMonitor(): IMonitoring;
    InstallLicense(LicenseFile: string): void;
}
```

```C#
public interface IUsbMonitor : IDispatch
{
    // Properties
    IDeviceCollection Devices { get; }
    // Methods
    IMonitoring CreateMonitor();
    void InstallLicense(string LicenseFile);
    void InstallLicenseInMemory(byte[] data);
    void DisplayActivationDialog(IntPtr hWnd);
}
```

```C++
struct IUsbMonitor : IDispatch
{
    // Properties
    IDeviceCollectionPtr Devices;  // get
    // Methods
    IMonitoringPtr CreateMonitor();
    HRESULT InstallLicense(_bstr_t LicenseFile);
    HRESULT InstallLicenseInMemoryNative(UCHAR * pData, ULONG size);
    HRESULT DisplayActivationDialog(HWND hWnd);
};
```

IUsbMonitor Properties

Devices

```TypeScript
readonly Devices: IDeviceCollection;
```

```C#
IDeviceCollection Devices { get; }
```

```C++
IDeviceCollectionPtr Devices;  // get
```

Description

Reference to the device collection object.

IUsbMonitor Methods

CreateMonitor

```TypeScript
CreateMonitor(): IMonitoring;
```

```C#
IMonitoring CreateMonitor();
```

```C++
IMonitoringPtr CreateMonitor();
```

Description

Creates a new monitor object that is ready to be attached to USB device.

InstallLicense

```
TypeScript
InstallLicense(LicenseFile: string): void;
```

```
C#
void InstallLicense(string LicenseFile);
```

```
C++
HRESULT InstallLicense(_bstr_t LicenseFile);
```

Parameters

`LicenseFile`

The full path to the license file.

Description

Install a license stored in a given file.

InstallLicenseInMemory

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void InstallLicenseInMemory(byte[] data);
```

```
C++
// This method is not available in native environment
```

Parameters

`data`

Byte array that stores the contents of the license file.

Description

Install a license stored in a memory buffer.

InstallLicenseInMemoryNative

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT InstallLicenseInMemoryNative(UCHAR * pData, ULONG size);
```

Parameters

`pData`

Pointer to a memory buffer that holds the contents of the license file.

`size`

Size of the buffer pointed by `pData`, in bytes.

Description

Install a license stored in a memory buffer.

DisplayActivationDialog

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void DisplayActivationDialog(IntPtr hWnd);
```

```
C++
HRESULT DisplayActivationDialog(HWND hWnd);
```

Parameters

`hWnd`

Dialog's parent window handle. May be `0`.

Description

Display the activation dialog.

INativeListener Interface

Description

The high-performance event interface for native (C/C++) listeners.

This is a local interface, which means that it can only be used by the in-proc binary compatible clients, written, for example, in unmanaged C++ language.

Register your native listener with a call to IMonitoring.AddNativeListener method call. Using native listener interface, you will be able to match the performance of the HHD Software USB Monitor application.

You must implement this interface in your native code to be able to receive monitored events. The USB Monitoring Control library uses the methods of this interface to notify your code about monitored events.

Declaration

```
TypeScript
// This interface is not available in scripting environment
```

```
C#
// This interface is not available in managed environment
```

```
C++
struct INativeListener : IUnknown
{
    // Methods
    HRESULT OnClearFeatureToEndpoint(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short FeatureSelector,
        unsigned short nIndex);
    HRESULT OnClearFeatureToInterface(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short FeatureSelector,
```

```
        unsigned short nIndex);
    HRESULT OnClearFeatureToOther(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short FeatureSelector);
    HRESULT OnConnection(FILETIME * fTime,
        long bConnected,
        wchar_t * DeviceName);
    HRESULT OnControlTransfer(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        void * SetupPacket,
        unsigned long SetupPacketSize);
    HRESULT OnGetConfiguration(FILETIME * fTime,
        void * Data,
        unsigned long Size);
    HRESULT OnGetCurrentFrameNumber(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned long FrameNumber);
    HRESULT OnGetDescriptorFromDevice(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char Index,
        unsigned char DescriptorType,
        unsigned short LanguageId);
    HRESULT OnGetDescriptorFromEndpoint(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char Index,
        unsigned char DescriptorType,
        unsigned short LanguageId);
    HRESULT OnGetDescriptorFromInterface(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char Index,
        unsigned char DescriptorType,
        unsigned short LanguageId);
    HRESULT OnGetFrameLength(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned long FrameLength,
        unsigned long FrameNumber);
    HRESULT OnGetInterface(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short Interface);
    HRESULT OnGetStatusFromDevice(FILETIME * fTime,
        void * pData,
        unsigned long Size);
    HRESULT OnGetStatusFromEndpoint(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short Index);
    HRESULT OnGetStatusFromInterface(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short Index);
    HRESULT OnGetStatusFromOther(FILETIME * fTime,
        void * pData,
        unsigned long Size);
    HRESULT OnIsochTransfer(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned long nTransferFlags,
        unsigned long StartFrame,
        unsigned long NumberOfPackets,
        unsigned long ErrorCount);
    HRESULT OnPacketDown(FILETIME * fTime,
        void * pData,
        unsigned long Size
```

```
        unsigned long Size,
        long * bStopParsing);
    HRESULT OnPacketUp(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        long * bStopParsing);
    HRESULT OnQueryID(FILETIME * fTime);
    HRESULT OnQueryInterface(FILETIME * fTime);
    HRESULT OnQueryText(FILETIME * fTime);
    HRESULT OnReleaseFrameLengthControl(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short Interface);
    HRESULT OnResetPipe(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned __int64 PipeHandle);
    HRESULT OnSelectConfiguration(FILETIME * fTime,
        void * pData,
        unsigned long Size);
    HRESULT OnSelectInterface(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned long InterfaceNumber,
        unsigned long AlternateSetting);
    HRESULT OnSetDescriptorToDevice(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char Index,
        unsigned char DescriptorType,
        unsigned long LanguageId);
    HRESULT OnSetDescriptorToEndpoint(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char Index,
        unsigned char DescriptorType,
        unsigned long LanguageId);
    HRESULT OnSetDescriptorToInterface(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char Index,
        unsigned char DescriptorType,
        unsigned long LanguageId);
    HRESULT OnSetFeatureToDevice(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short FeatureSelector);
    HRESULT OnSetFeatureToEndpoint(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short FeatureSelector,
        unsigned short Index);
    HRESULT OnSetFeatureToInterface(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short FeatureSelector,
        unsigned short Index);
    HRESULT OnSetFeatureToOther(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short FeatureSelector);
    HRESULT OnSetFrameLength(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        long FrameLengthDelta);
    HRESULT OnSurpriseRemoval(FILETIME * fTime);
    HRESULT OnTakeFrameLengthControl(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short Interface);
    HRESULT OnUrb(FILETIME * fTime,
```

```
    HRESULT OnUrb(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        long * bStopParsing);
    HRESULT OnVendorDevice(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char RequestTypeReservedBits,
        unsigned char Request,
        unsigned short Value);
    HRESULT OnVendorEndpoint(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char RequestTypeReservedBits,
        unsigned char Request,
        unsigned short Value,
        unsigned short Index);
    HRESULT OnVendorInterface(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char RequestTypeReservedBits,
        unsigned char Request,
        unsigned short Value,
        unsigned short Index);
    HRESULT OnVendorOther(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char RequestTypeReservedBits,
        unsigned char Request,
        unsigned short Value);
    HRESULT ProcessRAWBuffer(void * pData, unsinged long Size, long * bStopParsing);
    HRESULT OnAbortPipe(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned __int64 PipeHandle);
    HRESULT OnBulkOrInterruptTransfer(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        void * Payload,
        unsigned long PayloadSize);
    HRESULT OnClassDevice(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char RequestTypeReservedBits,
        unsigned char Request,
        unsigned short Value);
    HRESULT OnClassEndpoint(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char RequestTypeReservedBits,
        unsigned char Request,
        unsigned short Value,
        unsigned short Index);
    HRESULT OnClassInterface(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char RequestTypeReservedBits,
        unsigned char Request,
        unsigned short Value,
        unsigned short Index);
    HRESULT OnClassOther(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned char RequestTypeReservedBits,
        unsigned char Request,
        unsigned short Value);
    HRESULT OnClearFeatureToDevice(FILETIME * fTime,
        void * pData,
        unsigned long Size,
        unsigned short FeatureSelector);
};
```

```
};
```

INativeListener Methods

OnClearFeatureToEndpoint

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnClearFeatureToEndpoint(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short FeatureSelector,
    unsigned short nIndex);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the wValue field of the setup packet.

`nIndex`

Specifies the device-defined index, returned by a successful configuration request. The bus driver will copy the value in the Index member to the wIndex field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLEAR_FEATURE_TO_ENDPOINT` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnClearFeatureToInterface

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnClearFeatureToInterface(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short FeatureSelector,
    unsigned short nIndex);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

`nIndex`

Specifies the device-defined index, returned by a successful configuration request. The bus driver will copy the value in the Index member to the `wIndex` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLEAR_FEATURE_TO_INTERFACE` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnClearFeatureToOther

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnClearFeatureToOther(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short FeatureSelector);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLEAR_FEATURE_TO_OTHER` is

received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnConnection

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnConnection(FILETIME * fTime,
    long bConnected,
    wchar_t * DeviceName);
```

Parameters

`fTime`

The time of the event.

`bConnected`

The value of this parameter is `TRUE` if device is connected and `FALSE` if device is disconnected.

`DeviceName`

Name of the device. You can also get it by retrieving IDevice.Name at any time.

Description

Called when the control attaches/detaches itself to/from the USB device.

OnControlTransfer

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnControlTransfer(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    void * SetupPacket,
    unsigned long SetupPacketSize);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`SetupPacket`

Pointer to the setup packet.

`SetupPacketSize`

Size of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CONTROL_TRANSFER` is received. See `_URB_CONTROL_DESCRIPTOR_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetConfiguration

TypeScript
```
// This method is not available in scripting environment
```

C#
```
// This method is not available in managed environment
```

C++
```
HRESULT OnGetConfiguration(FILETIME * fTime,
    void * Data,
    unsigned long Size);
```

Parameters

`fTime`

The time of the event.

`Data`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

Description

See _URB_BULK_OR_INTERRUPT_TRANSFER in MSDN for more details.

OnGetCurrentFrameNumber

TypeScript
```
// This method is not available in scripting environment
```

C#
```
// This method is not available in managed environment
```

C++
```
HRESULT OnGetCurrentFrameNumber(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned long FrameNumber);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`FrameNumber`

Contains the current 32-bit frame number, on the USB bus, on return from the host controller driver.

Description

Fired when URB packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_CURRENT_FRAME_NUMBER` is received. See `_URB_GET_CURRENT_FRAME_NUMBER` in MSDN for more details. The library decodes several parameters from this packet.

OnGetDescriptorFromDevice

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnGetDescriptorFromDevice(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char Index,
    unsigned char DescriptorType,
    unsigned short LanguageId);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`Index`

Specifies the device-defined index of the descriptor that is being retrieved or set.

`DescriptorType`

Indicates what type of descriptor is being retrieved or set. One of the following values must be specified: `USB_DEVICE_DESCRIPTOR_TYPE` / `USB_CONFIGURATION_DESCRIPTOR_TYPE` / `USB_STRING_DESCRIPTOR_TYPE`.

`LanguageId`

Specifies the language ID of the descriptor to be retrieved when `USB_STRING_DESCRIPTOR_TYPE` is set in `DescriptorType`. This member must be set to zero for any other value in `DescriptorType`.

Description

Fired when URB packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE` is received. See `_URB_CONTROL_DESCRIPTOR_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetDescriptorFromEndpoint

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnGetDescriptorFromEndpoint(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char Index,
    unsigned char DescriptorType,
    unsigned short LanguageId);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`Index`

Specifies the device-defined index of the descriptor that is being retrieved or set.

`DescriptorType`

Indicates what type of descriptor is being retrieved or set. One of the following values must be specified: `USB_DEVICE_DESCRIPTOR_TYPE` / `USB_CONFIGURATION_DESCRIPTOR_TYPE` / `USB_STRING_DESCRIPTOR_TYPE`.

`LanguageId`

Specifies the language ID of the descriptor to be retrieved when `USB_STRING_DESCRIPTOR_TYPE` is set in `DescriptorType`. This member must be set to zero for any other value in `DescriptorType`.

Description

Fired when URB packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_DESCRIPTOR_FROM_ENDPOINT` is received. See `_URB_CONTROL_DESCRIPTOR_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetDescriptorFromInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnGetDescriptorFromInterface(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char Index,
    unsigned char DescriptorType,
    unsigned short LanguageId);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`Index`

Specifies the device-defined index of the descriptor that is being retrieved or set.

`DescriptorType`

Indicates what type of descriptor is being retrieved or set. One of the following values must be specified: `USB_DEVICE_DESCRIPTOR_TYPE` / `USB_CONFIGURATION_DESCRIPTOR_TYPE` / `USB_STRING_DESCRIPTOR_TYPE`.

`LanguageId`

Specifies the language ID of the descriptor to be retrieved when `USB_STRING_DESCRIPTOR_TYPE` is set in `DescriptorType`. This member must be set to zero for any other value in `DescriptorType`.

Description

Fired when URB packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_DESCRIPTOR_FROM_INTERFACE` is received. See `_URB_CONTROL_DESCRIPTOR_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetFrameLength

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnGetFrameLength(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned long FrameLength,
    unsigned long FrameNumber);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`FrameLength`

Contains the length of each bus frame in USB-defined bit times.

`FrameNumber`

Contains the earliest bus frame number that the frame length can be altered on return from the host controller driver.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_FRAME_LENGTH` is received. See `_URB_GET_FRAME_LENGTH` in MSDN for more details. The library decodes several parameters from this packet.

OnGetInterface

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnGetInterface(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short Interface);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`Interface`

Specifies the device-defined index of the interface descriptor being retrieved.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_INTERFACE` is received. See `_URB_CONTROL_GET_INTERFACE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetStatusFromDevice

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnGetStatusFromDevice(FILETIME * fTime,
    void * pData,
    unsigned long Size);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_STATUS_FROM_DEVICE` is received. See `_URB_CONTROL_GET_STATUS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetStatusFromEndpoint

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnGetStatusFromEndpoint(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short Index);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`Index`

Specifies the device-defined index, returned by a successful configuration request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_STATUS_FROM_ENDPOINT` is received. See `_URB_CONTROL_GET_STATUS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetStatusFromInterface

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

```
C++
HRESULT OnGetStatusFromInterface(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short Index);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`Index`

Specifies the device-defined index, returned by a successful configuration request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_STATUS_FROM_INTERFACE` is received. See `_URB_CONTROL_GET_STATUS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetStatusFromOther

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnGetStatusFromOther(FILETIME * fTime,
    void * pData,
    unsigned long Size);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_STATUS_FROM_OTHER` is received. See `_URB_CONTROL_GET_STATUS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnIsochTransfer

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnIsochTransfer(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned long nTransferFlags,
    unsigned long StartFrame,
    unsigned long NumberOfPackets,
    unsigned long ErrorCount);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`nTransferFlags`

Specifies zero, one, or a combination of the following flags: `USBD_TRANSFER_DIRECTION_IN`, `USBD_SHORT_TRANSFER_OK`, `USBD_START_ISO_TRANSFER_ASAP`.

`StartFrame`

Specifies the frame number the transfer should begin on. This variable must be within a system-defined range of the current frame. The range is specified by the constant `USBD_ISO_START_FRAME_RANGE`. If `START_ISO_TRANSFER_ASAP` is set in TransferFlags, this member contains the frame number that the transfer began on, when the request is returned by the host controller driver. Otherwise, this member must contain the frame number that this transfer will begin on.

`NumberOfPackets`

Specifies the number of packets described by the variable-length array member `IsoPacket`.

`ErrorCount`

Contains the number of packets that completed with an error condition on return from the host controller driver.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_ISOCH_TRANSFER` is received. See `_URB_ISOCH_TRANSFER` and `USBD_ISO_PACKET_DESCRIPTOR` for more details. The library decodes several parameters from this packet.

OnPacketDown

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnPacketDown(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    long * bStopParsing);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`bStopParsing`

Set this parameter to `TRUE` if you don't need USBMC to parse this packet any more. That means that USBMC will not call methods like `OnUrb` / `OnGetDescriptorFromDevice` / `OnGetDescriptorFromEndpoint` for this packet. That could be used for optimization if you manually parse `URB` packet.

Description

Called when packet is going down.

OnPacketUp

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnPacketUp(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    long * bStopParsing);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`bStopParsing`

Set this parameter to `TRUE` if you don't need USBMC to parse this packet any more. That means that USBMC will not call methods like `OnUrb` / `OnGetDescriptorFromDevice` / `OnGetDescriptorFromEndpoint` for this packet. That could be used for optimization if you manually parse `URB` packet.

Description

Called when packet is going up.

OnQueryID

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnQueryID(FILETIME * fTime);
```

Parameters

`fTime`

The time of the event.

Description

Called when id is queried with `EVENT_DEVICEQUERYID`.

OnQueryInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnQueryInterface(FILETIME * fTime);
```

Parameters

`fTime`

The time of the event.

Description

Called when id is queried with `EVENT_DEVICEQUERYINTERFACE`.

OnQueryText

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnQueryText(FILETIME * fTime);
```

Parameters

`fTime`

The time of the event.

Description

Called when id is queried with `EVENT_DEVICEQUERYTEXT`.

OnReleaseFrameLengthControl

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnReleaseFrameLengthControl(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short Interface);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`Interface`

Specifies the device-defined index of the interface descriptor being retrieved.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_RELEASE_FRAME_LENGTH_CONTROL` is received. See `_URB_CONTROL_GET_INTERFACE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnResetPipe

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnResetPipe(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned __int64 PipeHandle);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`PipeHandle`

Specifies an opaque handle to the bulk or interrupt pipe. The host controller driver returns this handle when the client driver selects the device configuration with a `URB` of type `URB_FUNCTION_SELECT_CONFIGURATION` or when the client driver changes the settings for an interface with a `URB` of type `URB_FUNCTION_SELECT_INTERFACE`.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_RESET_PIPE` is received. See `_URB_PIPE_REQUEST` in MSDN for more details.

OnSelectConfiguration

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnSelectConfiguration(FILETIME * fTime,
    void * pData,
    unsigned long Size);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SELECT_CONFIGURATION` is received. See `_URB_SELECT_CONFIGURATION` for more details.

OnSelectInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnSelectInterface(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned long InterfaceNumber,
    unsigned long AlternateSetting);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`InterfaceNumber`

Specifies the device-defined index identifier for this interface.

`AlternateSetting`

Specifies a device-defined index identifier that indicates which alternate setting this interface is using, should use, or describes.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SELECT_INTERFACE` is received. See `_URB_SELECT_INTERFACE` in MSDN for more details. The library decodes several parameters from this packet.

OnSetDescriptorToDevice

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnSetDescriptorToDevice(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char Index,
    unsigned char DescriptorType,
    unsigned long LanguageId);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`Index`

Specifies the device-defined index of the descriptor that is being retrieved or set.

`DescriptorType`

Indicates what type of descriptor is being retrieved or set. One of the following values must be specified: `USB_DEVICE_DESCRIPTOR_TYPE`, `USB_CONFIGURATION_DESCRIPTOR_TYPE`, `USB_STRING_DESCRIPTOR_TYPE`.

`LanguageId`

Specifies the language ID of the descriptor to be retrieved when `USB_STRING_DESCRIPTOR_TYPE` is set in DescriptorType. This member must be set to zero for any other value in `DescriptorType`.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_DESCRIPTOR_TO_DEVICE` is received. See `_URB_CONTROL_DESCRIPTOR_REQUEST` for more details. The library decodes several parameters from this packet.

OnSetDescriptorToEndpoint

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnSetDescriptorToEndpoint(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char Index,
    unsigned char DescriptorType,
    unsigned long LanguageId);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`Index`

Specifies the device-defined index of the descriptor that is being retrieved or set.

`DescriptorType`

Indicates what type of descriptor is being retrieved or set. One of the following values must be specified: `USB_DEVICE_DESCRIPTOR_TYPE`, `USB_CONFIGURATION_DESCRIPTOR_TYPE`, `USB_STRING_DESCRIPTOR_TYPE`.

`LanguageId`

Specifies the language ID of the descriptor to be retrieved when `USB_STRING_DESCRIPTOR_TYPE` is set in DescriptorType. This member must be set to zero for any other value in `DescriptorType`.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_DESCRIPTOR_TO_ENDPOINT` is received. See `_URB_CONTROL_DESCRIPTOR_REQUEST` for more details. The library decodes several parameters from this packet.

OnSetDescriptorToInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnSetDescriptorToInterface(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char Index,
    unsigned char DescriptorType,
    unsigned long LanguageId);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`Index`

Specifies the device-defined index of the descriptor that is being retrieved or set.

`DescriptorType`

Indicates what type of descriptor is being retrieved or set. One of the following values must be specified: `USB_DEVICE_DESCRIPTOR_TYPE`, `USB_CONFIGURATION_DESCRIPTOR_TYPE`, `USB_STRING_DESCRIPTOR_TYPE`.

`LanguageId`

Specifies the language ID of the descriptor to be retrieved when `USB_STRING_DESCRIPTOR_TYPE` is set in DescriptorType. This member must be set to zero for any other value in `DescriptorType`.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_DESCRIPTOR_TO_INTERFACE` is received. See `_URB_CONTROL_DESCRIPTOR_REQUEST` for more details. The library decodes several parameters from this packet.

OnSetFeatureToDevice

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnSetFeatureToDevice(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short FeatureSelector);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_FEATURE_TO_DEVICE` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnSetFeatureToEndpoint

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnSetFeatureToEndpoint(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short FeatureSelector,
    unsigned short Index);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

`Index`

Specifies the device-defined index, returned by a successful configuration request. The bus driver will copy the value in the `Index` member to the `wIndex` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_FEATURE_TO_ENDPOINT` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several

parameters from this packet.

OnSetFeatureToInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnSetFeatureToInterface(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short FeatureSelector,
    unsigned short Index);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

`Index`

Specifies the device-defined index, returned by a successful configuration request. The bus driver will copy the value in the `Index` member to the `wIndex` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_FEATURE_TO_INTERFACE` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnSetFeatureToOther

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnSetFeatureToOther(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short FeatureSelector);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_FEATURE_TO_OTHER` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnSetFrameLength

TypeScript
```
// This method is not available in scripting environment
```

C#
```
// This method is not available in managed environment
```

C++
```
HRESULT OnSetFrameLength(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    long FrameLengthDelta);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`FrameLengthDelta`

Specifies the number of USB-defined bit times to be added or subtracted from the current frame length. The maximum increase or decrease per `URB` is 1.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_FRAME_LENGTH` is received. See `_URB_SET_FRAME_LENGTH` for more details. The library decodes several parameters from this packet.

OnSurpriseRemoval

TypeScript
```
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnSurpriseRemoval(FILETIME * fTime);
```

Parameters

`fTime`

Description

Called when device is removed with `EVENT_DEVICESURPRISEREMOVAL`.

OnTakeFrameLengthControl

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnTakeFrameLengthControl(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short Interface);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`Interface`

Specifies the device-defined index of the interface descriptor being retrieved.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_TAKE_FRAME_LENGTH_CONTROL` is received. See `_URB_CONTROL_GET_INTERFACE_REQUEST` for more details. The library decodes several parameters from this packet.

OnUrb

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnUrb(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    long * bStopParsing);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`bStopParsing`

Set this to `TRUE` if you don't need USBMC to parse this packet any more. That means that USBMC will not call methods like `OnGetDescriptorFromDevice` / `OnGetDescriptorFromEndpoint` for this packet. That could be used for optimization if you manually parse URB packet.

Description

Called when `URB` is transmitted. See `USBPACKET_URB` for more information.

OnVendorDevice

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnVendorDevice(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char RequestTypeReservedBits,
    unsigned char Request,
    unsigned short Value);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-

defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in Request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_VENDOR_DEVICE` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more information. The library decodes several parameters from this packet.

OnVendorEndpoint

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnVendorEndpoint(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char RequestTypeReservedBits,
    unsigned char Request,
    unsigned short Value,
    unsigned short Index);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in Request.

`Index`

Specifies the device-defined index, returned by a successful configuration request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_VENDOR_ENDPOINT` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more information. The library decodes several parameters from this packet.

OnVendorInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnVendorInterface(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char RequestTypeReservedBits,
    unsigned char Request,
    unsigned short Value,
    unsigned short Index);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in Request.

`Index`

Specifies the device-defined index, returned by a successful configuration request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_VENDOR_INTERFACE` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more information. The library decodes several parameters from this packet.

OnVendorOther

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnVendorOther(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char RequestTypeReservedBits,
    unsigned char Request,
    unsigned short Value);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in Request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_VENDOR_OTHER` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more information. The library decodes several parameters from this packet.

ProcessRAWBuffer

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT ProcessRAWBuffer(void * pData, unsinged long Size, long * bStopParsing);
```

Parameters

`pData`

Pointer to a raw buffer that may contain multiple packets. Use it to manually parse all packets. See MFCSample for more details.

`Size`

Total size of the raw buffer.

`bStopParsing`

Set this to `TRUE` if you don't need USBMC to parse this buffer any more. That means that USBMC will

not call methods like `OnPacketUp`, `OnPacketDown`, `OnUrb`, `OnGetDescriptorFromDevice`, `OnGetDescriptorFromEndpoint` for all packets that are contained in buffer. That could be used for optimization if you manually parse this buffer.

Description

Called by the control to process the monitored events at the lowest possible level.

OnAbortPipe

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnAbortPipe(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned __int64 PipeHandle);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`PipeHandle`

Specifies an opaque handle to the bulk or interrupt pipe. The host controller driver returns this handle when the client driver selects the device configuration with a `URB` of type `URB_FUNCTION_SELECT_CONFIGURATION` or when the client driver changes the settings for an interface with a `URB` of type `URB_FUNCTION_SELECT_INTERFACE`.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_ABORT_PIPE` is received. See `_URB_PIPE_REQUEST` in MSDN for more details.

OnBulkOrInterruptTransfer

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnBulkOrInterruptTransfer(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    void * Payload,
    unsigned long PayloadSize);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`Payload`

Pointer to payload that trails after all headers. Use it to manually parse all fields. See MFCSample for more details. Note that in headers-only mode this pointer is invalid.

`PayloadSize`

The size of the payload. This parameter is zero when in headers-only mode.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER` is received. See `_URB_BULK_OR_INTERRUPT_TRANSFER` for more details.

OnClassDevice

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnClassDevice(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char RequestTypeReservedBits,
    unsigned char Request,
    unsigned short Value);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in Request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLASS_DEVICE` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnClassEndpoint

TypeScript

```
// This method is not available in scripting environment
```

C#

```
// This method is not available in managed environment
```

C++

```
HRESULT OnClassEndpoint(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char RequestTypeReservedBits,
    unsigned char Request,
    unsigned short Value,
    unsigned short Index);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in Request.

`Index`

Specifies the device-defined index, returned by a successful configuration request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLASS_ENDPOINT` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnClassInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnClassInterface(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char RequestTypeReservedBits,
    unsigned char Request,
    unsigned short Value,
    unsigned short Index);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in Request.

`Index`

Specifies the device-defined index, returned by a successful configuration request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLASS_INTERFACE` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnClassOther

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnClassOther(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned char RequestTypeReservedBits,
    unsigned char Request,
    unsigned short Value);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in Request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLASS_OTHER` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnClearFeatureToDevice

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
HRESULT OnClearFeatureToDevice(FILETIME * fTime,
    void * pData,
    unsigned long Size,
    unsigned short FeatureSelector);
```

Parameters

`fTime`

The time of the event.

`pData`

Pointer to a `USBPACKET`. Use it to manually parse all fields. See MFCSample for more details.

`Size`

Total size of `USBPACKET` and all payload data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLEAR_FEATURE_TO_DEVICE` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

IMonitoring Interface

Description

This interface is implemented by the Monitor object in the USB Monitoring Control library. You get this interface by calling the IUsbMonitor.CreateMonitor method and use it to start monitoring the USB device.

Declaration

TypeScript

```typescript
interface IMonitoring {
    // Properties
    readonly Connected: boolean;
    readonly ConnectedDevice: IDevice;
    readonly UsbMonitor: IUsbMonitor;
    // Methods
    Connect(Device?: IDevice, HeadersOnly?: boolean): void;
    Disconnect(): void;
}
```

C#

```csharp
public interface IMonitoring
{
    // Properties
    bool Connected { get; }
    IDevice ConnectedDevice { get; }
    IUsbMonitor UsbMonitor { get; }
    // Methods
    void Connect(IDevice Device, bool HeadersOnly);
    void Disconnect();
}
```

C++

```cpp
struct IMonitoring : IDispatch
{
    // Properties
    VARIANT_BOOL Connected;  // get
    IDevicePtr ConnectedDevice;  // get
    IUsbMonitorPtr UsbMonitor;  // get
    // Methods
    HRESULT Connect(_variant_t Device, _variant_t HeadersOnly);
    HRESULT Disconnect();
    HRESULT AddNativeListener(INativeListener * Listener);
    HRESULT RemoveNativeListener(INativeListener * Listener);
};
```

IMonitoring Properties

Connected

TypeScript
```
readonly Connected: boolean;
```

C#
```
bool Connected { get; }
```

C++
```
VARIANT_BOOL Connected;  // get
```

Description

Returns `true` if it is currently connected to the USB device.

ConnectedDevice

TypeScript
```
readonly ConnectedDevice: IDevice;
```

C#
```
IDevice ConnectedDevice { get; }
```

C++
```
IDevicePtr ConnectedDevice;  // get
```

Description

Returns the device this monitor object is currently connected to, or `null` if it is not connected.

UsbMonitor

TypeScript
```
readonly UsbMonitor: IUsbMonitor;
```

C#
```
IUsbMonitor UsbMonitor { get; }
```

C++
```
IUsbMonitorPtr UsbMonitor;  // get
```

Description

Returns the reference to the main UsbMonitor object.

IMonitoring Methods

Connect

TypeScript
```
Connect(Device?: IDevice, HeadersOnly?: boolean): void;
```

C#
```
void Connect(IDevice Device, bool HeadersOnly);
```

C++
```
HRESULT Connect(_variant_t Device, _variant_t HeadersOnly);
```

Parameters

`Device`

Optional reference to the device to monitor. If missing, a session connects to the next connected USB device.

`HeadersOnly`

`true` to only process packet headers, `false` to process full packets. Default is `false` if parameter is omitted.

Description

Connect to the USB device.

Disconnect

TypeScript
```
Disconnect(): void;
```

C#
```
void Disconnect();
```

C++
```
HRESULT Disconnect();
```

Description

Disconnect from the USB device.

AddNativeListener

TypeScript
```
// This method is not available in scripting environment
```

C#
```
// This method is not available in managed environment
```

C++
```
HRESULT AddNativeListener(INativeListener * Listener);
```

Parameters

`Listener`

Pointer to native listener client provides. Can be used only by a native code. Please see MFCSample for more details.

Description

Add new native listener.

RemoveNativeListener

TypeScript
```
// This method is not available in scripting environment
```

C#
```
// This method is not available in managed environment
```

```
C++
HRESULT RemoveNativeListener(INativeListener * Listener);
```

Parameters

`Listener`

Pointer to native listener client provides. Can be used only by a native code. Please see MFCSample for more details.

Description

Remove native listener.

IDeviceCollection Interface

Description

You obtain this interface by taking the value of the IUsbMonitor.Devices property and use to enumerate the installed USB devices. There are two ways for using this interface. You can get the `Count` property value to get the number of devices in the collection and then use the default Item property to get the IDevice interface for each device in a collection.

Another way of enumerating the devices in the collection is to take the value of the `_NewEnum` property to get the object exposing the `IEnumVARIANT` interface and use its properties and members to enumerate the collection.

Note that usually this process is somehow automated in scripting and CLR languages. You will find the code samples in topics for the `_NewEnum` and Item properties.

Declaration

```
TypeScript
interface IDeviceCollection extends IDispatch {
    // Properties
    readonly [Item: number]: IDevice;
    Count: number;
    _NewEnum: object;
}
```

```
C#
public interface IDeviceCollection : IDispatch
{
    // Properties
    IDevice Item[int Index] { get; }
    int Count { get; set; }
    object _NewEnum { get; set; }
}
```

```
C++
struct IDeviceCollection : IDispatch
{
    // Properties
    IDevicePtr Item(_variant_t Index);  // get
    long Count;  // get set
    IUnknownPtr _NewEnum;  // get set
};
```

IDeviceCollection Properties

Item

```
TypeScript
readonly [Item: number]: IDevice;
```

```
C#
IDevice Item[int Index] { get; }
```

```
C++
IDevicePtr Item(_variant_t Index);  // get
```

Description

Reference to Device object.

Count

```
TypeScript
Count: number;
```

```
C#
int Count { get; set; }
```

```
C++
long Count;  // get set
```

Description

Number of devices.

_NewEnum

```
TypeScript
_NewEnum: object;
```

```
C#
object _NewEnum { get; set; }
```

```
C++
IUnknownPtr _NewEnum;  // get set
```

Description

Returns the enumerator object implementing `IEnumVARIANT` interface.

IDevice

_IUsbMonitorEvents Interface

Description

You implement this interface to receive the events fired by the USB Monitor Control library's main object.

The USB Monitor Control calls the_IUsbMonitorEvents.OnChange method when there is a change in the device collection entries. Please see documentation for the _IMonitoringEvents interface for information on binding to the event source.

Declaration

```
TypeScript
// This interface is not available in scripting environment
```

```
C#
public interface _IUsbMonitorEvents
{
    // Methods
    void OnChange();
}
```

```
C++
struct _IUsbMonitorEvents : IDispatch
{
    // Methods
    HRESULT OnChange();
};
```

_IUsbMonitorEvents Methods

OnChange

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnChange();
```

```
C++
HRESULT OnChange();
```

Description

Fired when device appears or disappears.

_IMonitoringEvents Interface

Description

An event source interface for managed and scripting clients.

You do not explicitly implement this interface. It is usually implemented by the language runtime. You register the so-called “events”, one for each method of this interface to handle specific monitored requests.

This interface is used by the managed and scripting clients. The language runtime usually uses the methods of this interface automatically, allowing the client to register the events, callbacks or delegates, which are called when the USB Monitoring Control library fires these events. There is a proprietary interface in each managed language to connect to event sources. You will see Microsoft C# examples in this documentation, for other languages, please consult their documentation for a proper syntax to handle events.

Declaration

```
TypeScript
// This interface is not available in scripting environment
```

```
C#
public interface _IMonitoringEvents
{
    // Methods
    void OnClassEndpoint(DateTime time,
        byte[] array
```

```
        byte[] array,
        byte RequestTypeReservedBits,
        byte Request,
        ushort Value,
        ushort Index);
    void OnClassInterface(DateTime time,
        byte[] array,
        byte RequestTypeReservedBits,
        byte Request,
        ushort Value,
        ushort Index);
    void OnClassOther(DateTime time,
        byte[] array,
        byte RequestTypeReservedBits,
        byte Request,
        ushort Value);
    void OnClearFeatureToDevice(DateTime time,
        byte[] array,
        ushort FeatureSelector);
    void OnClearFeatureToEndpoint(DateTime time,
        byte[] array,
        ushort FeatureSelector,
        ushort Index);
    void OnClearFeatureToInterface(DateTime time,
        byte[] array,
        ushort FeatureSelector,
        ushort Index);
    void OnClearFeatureToOther(DateTime time,
        byte[] array,
        ushort FeatureSelector);
    void OnConnection(DateTime time,
        ConnectionState cs,
        string Name);
    void OnControlTransfer(DateTime time,
        byte[] array,
        uint payloadOffset,
        uint payloadSize);
    void OnGetConfiguration(DateTime time, byte[] array);
    void OnGetCurrentFrameNumber(DateTime time,
        byte[] array,
        uint FrameNumber);
    void OnGetDescriptorFromDevice(DateTime time,
        byte[] array,
        byte Index,
        byte DescriptorType,
        ushort LanguageId);
    void OnGetDescriptorFromEndpoint(DateTime time,
        byte[] array,
        byte Index,
        byte DescriptorType,
        ushort LanguageId);
    void OnGetDescriptorFromInterface(DateTime time,
        byte[] array,
        byte Index,
        byte DescriptorType,
        ushort LanguageId);
    void OnGetFrameLength(DateTime time,
        byte[] array,
        uint FrameLength,
        uint FrameNumber);
    void OnGetInterface(DateTime time, byte[] array, ushort Interface);
    void OnGetStatusFromDevice(DateTime time, byte[] array);
    void OnGetStatusFromEndpoint(DateTime time, byte[] array, ushort Index);
    void OnGetStatusFromInterface(DateTime time, byte[] array, ushort Index);
    void OnGetStatusFromOther(DateTime time, byte[] array);
    void OnIsochTransfer(DateTime time,
        byte[] array,
        uint nTransferFlags,
        uint StartFrame,
        uint NumberOfPackets,
        uint ErrorCount);
```

```
        uint ErrorCount);
    void OnPacketDown(DateTime time, byte[] array);
    void OnPacketUp(DateTime time, byte[] array);
    void OnQueryID(DateTime time);
    void OnQueryInterface(DateTime time);
    void OnQueryText(DateTime time);
    void OnReleaseFrameLengthControl(DateTime time, byte[] array, ushort Interface);
    void OnResetPipe(DateTime time, byte[] array, ulong PipeHandle);
    void OnSelectConfiguration(DateTime time, byte[] array);
    void OnSelectInterface(DateTime time,
        byte[] array,
        uint InterfaceNumber,
        byte AlternateSetting);
    void OnSetDescriptorToDevice(DateTime time,
        byte[] array,
        byte Index,
        byte DescriptorType,
        ushort LanguageId);
    void OnSetDescriptorToEndpoint(DateTime time,
        byte[] array,
        ushort FeatureSelector,
        ushort Index);
    void OnSetDescriptorToInterface(DateTime time,
        byte[] array,
        ushort FeatureSelector,
        ushort Index,
        byte DescriptorType,
        ushort LanguageId);
    void OnSetFeatureToDevice(DateTime time, byte[] array, ushort FeatureSelector);
    void OnSetFeatureToEndpoint(DateTime time,
        byte[] array,
        ushort FeatureSelector,
        ushort Index);
    void OnSetFeatureToInterface(DateTime time,
        byte[] array,
        ushort FeatureSelector,
        ushort Index);
    void OnSetFeatureToOther(DateTime time,
        byte[] array,
        ushort FeatureSelector);
    void OnSetFrameLength(DateTime time, byte[] array, int FrameLengthDelta);
    void OnSurpriseRemoval(DateTime time);
    void OnTakeFrameLengthControl(DateTime time, byte[] array, ushort Interface);
    void OnUrb(DateTime time, byte[] array);
    void OnVendorDevice(DateTime time,
        byte[] array,
        byte RequestTypeReservedBits,
        byte Request,
        ushort Value);
    void OnVendorEndpoint(DateTime time,
        byte[] array,
        byte RequestTypeReservedBits,
        byte Request,
        ushort Value,
        ushort Index);
    void OnVendorInterface(DateTime time,
        byte[] array,
        byte RequestTypeReservedBits,
        byte Request,
        ushort Value,
        ushort Index);
    void OnVendorOther(DateTime time,
        byte[] array,
        byte RequestTypeReservedBits,
        byte Request,
        ushort Value);
    void OnAbortPipe(DateTime time,
        byte[] array,
        ulong PipeHandle);
    void OnBulkOrInterruptTransfer(DateTime time,
        byte[] array
```

```
        byte[] array,
        uint payloadOffset,
        uint payloadSize);
    void OnClassDevice(DateTime time,
        byte[] array,
        byte RequestTypeReservedBits,
        byte Request,
        ushort Value);
}
```

C++

```
// This interface is not available in native environment
```

_IMonitoringEvents Methods

OnClassEndpoint

TypeScript

```
// This method is not available in scripting environment
```

C#

```
void OnClassEndpoint(DateTime time,
    byte[] array,
    byte RequestTypeReservedBits,
    byte Request,
    ushort Value,
    ushort Index);
```

C++

```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in Request.

`Index`

Specifies the device-defined index, returned by a successful configuration request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLASS_ENDPOINT` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnClassInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnClassInterface(DateTime time,
    byte[] array,
    byte RequestTypeReservedBits,
    byte Request,
    ushort Value,
    ushort Index);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in Request.

`Index`

Specifies the device-defined index, returned by a successful configuration request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLASS_INTERFACE` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnClassOther

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnClassOther(DateTime time,
    byte[] array,
    byte RequestTypeReservedBits,
    byte Request,
    ushort Value);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in Request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLASS_OTHER` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnClearFeatureToDevice

TypeScript

```
// This method is not available in scripting environment
```

C#

```
void OnClearFeatureToDevice(DateTime time,
    byte[] array,
    ushort FeatureSelector);
```

C++

```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the FeatureSelector member to the wValue field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLEAR_FEATURE_TO_DEVICE` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnClearFeatureToEndpoint

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnClearFeatureToEndpoint(DateTime time,
    byte[] array,
    ushort FeatureSelector,
    ushort Index);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

`Index`

Specifies the device-defined index, returned by a successful configuration request. The bus driver will copy the value in the `Index` member to the `wIndex` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLEAR_FEATURE_TO_ENDPOINT` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnClearFeatureToInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnClearFeatureToInterface(DateTime time,
    byte[] array,
    ushort FeatureSelector,
    ushort Index);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

`Index`

Specifies the device-defined index, returned by a successful configuration request. The bus driver will copy the value in the `Index` member to the `wIndex` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLEAR_FEATURE_TO_INTERFACE` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnClearFeatureToOther

TypeScript

```
// This method is not available in scripting environment
```

C#

```
void OnClearFeatureToOther(DateTime time,
    byte[] array,
    ushort FeatureSelector);
```

C++

```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLEAR_FEATURE_TO_OTHER` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnConnection

TypeScript

```
// This method is not available in scripting environment
```

C#

```
void OnConnection(DateTime time,
    ConnectionState cs,
    string Name);
```

C++

```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`cs`

The value of this parameter is `DeviceConnected` if device is connected and `DeviceDisconnected` if device is disconnected.

`Name`

Name of the device. You can get it by retrieving IDevice.Name at any time.

Description

Called when the control attaches/detaches itself to/from the USB device. (it is fired when USB packet with `EventType == EVENT_DEVICECONNECTED` or `EventType == EVENT_DEVICEDISCONNECTED` is received).

OnControlTransfer

TypeScript

```
// This method is not available in scripting environment
```

C#

```
void OnControlTransfer(DateTime time,
    byte[] array,
    uint payloadOffset,
    uint payloadSize);
```

C++

```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`payloadOffset`

Payload offset, in bytes.

`payloadSize`

Payload size, in bytes.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CONTROL_TRANSFER` is received. See `_URB_CONTROL_DESCRIPTOR_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetConfiguration

TypeScript

```
// This method is not available in scripting environment
```

C#

```
void OnGetConfiguration(DateTime time, byte[] array);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

Description

See `_URB_BULK_OR_INTERRUPT_TRANSFER` in MSDN for more details.

OnGetCurrentFrameNumber

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnGetCurrentFrameNumber(DateTime time,
    byte[] array,
    uint FrameNumber);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FrameNumber`

Contains the current 32-bit frame number, on the USB bus, on return from the host controller driver.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_CURRENT_FRAME_NUMBER` is received. See `_URB_GET_CURRENT_FRAME_NUMBER` in MSDN for more details. The library decodes several parameters from this packet.

OnGetDescriptorFromDevice

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnGetDescriptorFromDevice(DateTime time,
    byte[] array,
    byte Index,
    byte DescriptorType,
    ushort LanguageId);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Original URB packet data.

`Index`

Specifies the device-defined index of the descriptor that is being retrieved or set.

`DescriptorType`

Indicates what type of descriptor is being retrieved or set. One of the following values must be specified: `USB_DEVICE_DESCRIPTOR_TYPE`, `USB_CONFIGURATION_DESCRIPTOR_TYPE` or `USB_STRING_DESCRIPTOR_TYPE`.

`LanguageId`

Specifies the language ID of the descriptor to be retrieved when USB_STRING_DESCRIPTOR_TYPE is set in DescriptorType. This member must be set to zero for any other value in DescriptorType.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_DESCRIPTOR_FROM_DEVICE` is received. See `_URB_CONTROL_DESCRIPTOR_REQUEST` for more details. The library decodes several parameters from this packet.

OnGetDescriptorFromEndpoint

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnGetDescriptorFromEndpoint(DateTime time,
    byte[] array,
    byte Index,
    byte DescriptorType,
    ushort LanguageId);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Original URB packet data.

`Index`

Specifies the device-defined index of the descriptor that is being retrieved or set.

`DescriptorType`

Indicates what type of descriptor is being retrieved or set. One of the following values must be specified: `USB_DEVICE_DESCRIPTOR_TYPE`, `USB_CONFIGURATION_DESCRIPTOR_TYPE` or

`USB_STRING_DESCRIPTOR_TYPE`.

`LanguageId`

Specifies the language ID of the descriptor to be retrieved when USB_STRING_DESCRIPTOR_TYPE is set in DescriptorType. This member must be set to zero for any other value in DescriptorType.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_DESCRIPTOR_FROM_ENDPOINT` is received. See `_URB_CONTROL_DESCRIPTOR_REQUEST` for more details. The library decodes several parameters from this packet.

OnGetDescriptorFromInterface

TypeScript

```
// This method is not available in scripting environment
```

C#

```
void OnGetDescriptorFromInterface(DateTime time,
    byte[] array,
    byte Index,
    byte DescriptorType,
    ushort LanguageId);
```

C++

```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Original URB packet data.

`Index`

Specifies the device-defined index of the descriptor that is being retrieved or set.

`DescriptorType`

Indicates what type of descriptor is being retrieved or set. One of the following values must be specified: `USB_DEVICE_DESCRIPTOR_TYPE`, `USB_CONFIGURATION_DESCRIPTOR_TYPE` or `USB_STRING_DESCRIPTOR_TYPE`.

`LanguageId`

Specifies the language ID of the descriptor to be retrieved when USB_STRING_DESCRIPTOR_TYPE is set in DescriptorType. This member must be set to zero for any other value in DescriptorType.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_DESCRIPTOR_FROM_INTERFACE` is received. See `_URB_CONTROL_DESCRIPTOR_REQUEST` for more details. The library decodes several parameters from this packet.

OnGetFrameLength

TypeScript

```
// This method is not available in scripting environment
```

```
C#
void OnGetFrameLength(DateTime time,
    byte[] array,
    uint FrameLength,
    uint FrameNumber);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FrameLength`

Contains the length of each bus frame in USB-defined bit times.

`FrameNumber`

Contains the earliest bus frame number that the frame length can be altered on return from the host controller driver.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_FRAME_LENGTH` is received. See `_URB_GET_FRAME_LENGTH` in MSDN for more details. The library decodes several parameters from this packet.

OnGetInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnGetInterface(DateTime time, byte[] array, ushort Interface);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`Interface`

Specifies the device-defined index of the interface descriptor being retrieved.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_INTERFACE` is received. See `_URB_CONTROL_GET_INTERFACE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetStatusFromDevice

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnGetStatusFromDevice(DateTime time, byte[] array);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_STATUS_FROM_DEVICE` is received. See `_URB_CONTROL_GET_STATUS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetStatusFromEndpoint

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnGetStatusFromEndpoint(DateTime time, byte[] array, ushort Index);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`Index`

Specifies the device-defined index, returned by a successful configuration request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_STATUS_FROM_ENDPOINT` is received. See `_URB_CONTROL_GET_STATUS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetStatusFromInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnGetStatusFromInterface(DateTime time, byte[] array, ushort Index);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`Index`

Specifies the device-defined index, returned by a successful configuration request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_STATUS_FROM_INTERFACE` is received. See `_URB_CONTROL_GET_STATUS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnGetStatusFromOther

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnGetStatusFromOther(DateTime time, byte[] array);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_GET_STATUS_FROM_OTHER` is received. See `_URB_CONTROL_GET_STATUS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnIsochTransfer

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnIsochTransfer(DateTime time,
    byte[] array,
    uint nTransferFlags,
    uint StartFrame,
    uint NumberOfPackets,
    uint ErrorCount);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`nTransferFlags`

Specifies zero, one, or a combination of the following flags: `USBD_TRANSFER_DIRECTION_IN`, `USBD_SHORT_TRANSFER_OK`, `USBD_START_ISO_TRANSFER_ASAP`.

`StartFrame`

Specifies the frame number the transfer should begin on. This variable must be within a system-defined range of the current frame. The range is specified by the constant `USBD_ISO_START_FRAME_RANGE`. If `START_ISO_TRANSFER_ASAP` is set in `TransferFlags`, this member contains the frame number that the transfer began on, when the request is returned by the host controller driver. Otherwise, this member must contain the frame number that this transfer will begin on.

`NumberOfPackets`

Specifies the number of packets described by the variable-length array member IsoPacket.

`ErrorCount`

Contains the number of packets that completed with an error condition on return from the host controller driver.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_ISOCH_TRANSFER` is received. See `_URB_ISOCH_TRANSFER` and `USBD_ISO_PACKET_DESCRIPTOR` in MSDN for more details. The library decodes several parameters from this packet.

OnPacketDown

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnPacketDown(DateTime time, byte[] array);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

Description

Called when packet is going down.

OnPacketUp

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnPacketUp(DateTime time, byte[] array);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

Description

Called when packet is going up.

OnQueryID

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnQueryID(DateTime time);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

Description

Called when id is queried by `EVENT_DEVICEQUERYID`.

OnQueryInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnQueryInterface(DateTime time);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

Description

Called when interface is queried by `EVENT_DEVICEQUERYINTERFACE`.

OnQueryText

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnQueryText(DateTime time);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

Description

Called when text is queried by `EVENT_DEVICEQUERYTEXT`.

OnReleaseFrameLengthControl

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnReleaseFrameLengthControl(DateTime time, byte[] array, ushort Interface);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`Interface`

Specifies the device-defined index of the interface descriptor being retrieved.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_RELEASE_FRAME_LENGTH_CONTROL` is received. See `_URB_CONTROL_GET_INTERFACE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnResetPipe

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnResetPipe(DateTime time, byte[] array, ulong PipeHandle);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`PipeHandle`

Specifies an opaque handle to the bulk or interrupt pipe. The host controller driver returns this handle when the client driver selects the device configuration with a `URB` of type `URB_FUNCTION_SELECT_CONFIGURATION` or when the client driver changes the settings for an interface with a `URB` of type `URB_FUNCTION_SELECT_INTERFACE`.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_RESET_PIPE` is received. See `_URB_PIPE_REQUEST` in MSDN for more details.

OnSelectConfiguration

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnSelectConfiguration(DateTime time, byte[] array);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SELECT_CONFIGURATION` is received. See `_URB_SELECT_CONFIGURATION` in MSDN for more details.

OnSelectInterface

TypeScript

```
// This method is not available in scripting environment
```

C#

```
void OnSelectInterface(DateTime time,
    byte[] array,
    uint InterfaceNumber,
    byte AlternateSetting);
```

C++

```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`InterfaceNumber`

Specifies the device-defined index identifier for this interface.

`AlternateSetting`

Specifies a device-defined index identifier that indicates which alternate setting this interface is using, should use, or describes.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SELECT_INTERFACE` is received. See `_URB_SELECT_INTERFACE` in MSDN for more details. The library decodes several parameters from this packet.

OnSetDescriptorToDevice

TypeScript

```
// This method is not available in scripting environment
```

C#

```
void OnSetDescriptorToDevice(DateTime time,
    byte[] array,
    byte Index,
    byte DescriptorType,
    ushort LanguageId);
```

C++

```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`Index`

Specifies the device-defined index of the descriptor that is being retrieved or set.

`DescriptorType`

Indicates what type of descriptor is being retrieved or set. One of the following values must be specified: `USB_DEVICE_DESCRIPTOR_TYPE`, `USB_CONFIGURATION_DESCRIPTOR_TYPE`, `USB_STRING_DESCRIPTOR_TYPE`.

`LanguageId`

Specifies the language ID of the descriptor to be retrieved when `USB_STRING_DESCRIPTOR_TYPE` is set in `DescriptorType`. This member must be set to zero for any other value in `DescriptorType`.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_DESCRIPTOR_TO_DEVICE` is received. See `_URB_CONTROL_DESCRIPTOR_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnSetDescriptorToEndpoint

TypeScript

```
// This method is not available in scripting environment
```

C#

```
void OnSetDescriptorToEndpoint(DateTime time,
    byte[] array,
    ushort FeatureSelector,
    ushort Index);
```

C++

```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the FeatureSelector member to the wValue field of the setup packet.

`Index`

Specifies the device-defined index, returned by a successful configuration request, if the request is for an endpoint or interface. Otherwise, Index must be zero. The bus driver will copy the value in the `Index` member to the `wIndex` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_DESCRIPTOR_TO_ENDPOINT` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnSetDescriptorToInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnSetDescriptorToInterface(DateTime time,
    byte[] array,
    ushort FeatureSelector,
    ushort Index,
    byte DescriptorType,
    ushort LanguageId);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the FeatureSelector member to the wValue field of the setup packet.

`Index`

Specifies the device-defined index, returned by a successful configuration request, if the request is for an endpoint or interface. Otherwise, Index must be zero. The bus driver will copy the value in the `Index` member to the `wIndex` field of the setup packet.

`DescriptorType`

Indicates what type of descriptor is being retrieved or set. One of the following values must be specified: `USB_DEVICE_DESCRIPTOR_TYPE`, `USB_CONFIGURATION_DESCRIPTOR_TYPE`, `USB_STRING_DESCRIPTOR_TYPE`.

`LanguageId`

Specifies the language ID of the descriptor to be retrieved when `USB_STRING_DESCRIPTOR_TYPE` is set in `DescriptorType`. This member must be set to zero for any other value in `DescriptorType`.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_DESCRIPTOR_TO_INTERFACE` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnSetFeatureToDevice

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnSetFeatureToDevice(DateTime time, byte[] array, ushort FeatureSelector);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_FEATURE_TO_DEVICE` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnSetFeatureToEndpoint

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnSetFeatureToEndpoint(DateTime time,
    byte[] array,
    ushort FeatureSelector,
    ushort Index);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

`Index`

Specifies the device-defined index, returned by a successful configuration request. The bus driver will copy the value in the `Index` member to the `wIndex` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_FEATURE_TO_ENDPOINT` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnSetFeatureToInterface

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnSetFeatureToInterface(DateTime time,
    byte[] array,
    ushort FeatureSelector,
    ushort Index);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

`Index`

Specifies the device-defined index, returned by a successful configuration request. The bus driver will copy the value in the `Index` member to the `wIndex` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_FEATURE_TO_INTERFACE` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnSetFeatureToOther

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnSetFeatureToOther(DateTime time,
    byte[] array,
    ushort FeatureSelector);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FeatureSelector`

Specifies the USB-defined feature code to be cleared or set. Using a feature code that is invalid, cannot be set, or cannot be cleared will cause the target to stall. The bus driver will copy the value in the `FeatureSelector` member to the `wValue` field of the setup packet.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_FEATURE_TO_OTHER` is received. See `_URB_CONTROL_FEATURE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnSetFrameLength

TypeScript
```
// This method is not available in scripting environment
```

C#
```
void OnSetFrameLength(DateTime time, byte[] array, int FrameLengthDelta);
```

C++
```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`FrameLengthDelta`

Specifies the number of USB-defined bit times to be added or subtracted from the current frame length. The maximum increase or decrease per URB is 1.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_SET_FRAME_LENGTH` is received. See `_URB_SET_FRAME_LENGTH` in MSDN for more details. The library decodes several parameters from this packet.

OnSurpriseRemoval

TypeScript
```
// This method is not available in scripting environment
```

C#
```
void OnSurpriseRemoval(DateTime time);
```

C++
```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

Description

Called when device is removed by `EVENT_DEVICESURPRISEREMOVAL`.

OnTakeFrameLengthControl

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnTakeFrameLengthControl(DateTime time, byte[] array, ushort Interface);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`Interface`

Specifies the device-defined index of the interface descriptor being retrieved.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_TAKE_FRAME_LENGTH_CONTROL` is received. See `_URB_CONTROL_GET_INTERFACE_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

OnUrb

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnUrb(DateTime time, byte[] array);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

Description

Called when `URB` is transmitted. See `USBPACKET_URB` for more information.

OnVendorDevice

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnVendorDevice(DateTime time,
    byte[] array,
    byte RequestTypeReservedBits,
    byte Request,
    ushort Value);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in `Request`.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_VENDOR_DEVICE` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more information. The library decodes several parameters from this packet.

OnVendorEndpoint

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnVendorEndpoint(DateTime time,
    byte[] array,
    byte RequestTypeReservedBits,
    byte Request,
    ushort Value,
    ushort Index);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in `Request`.

`Index`

Specifies the device-defined index, returned by a successful configuration request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_VENDOR_ENDPOINT` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more information. The library decodes several parameters from this packet.

OnVendorInterface

`TypeScript`

```
// This method is not available in scripting environment
```

`C#`

```
void OnVendorInterface(DateTime time,
    byte[] array,
    byte RequestTypeReservedBits,
    byte Request,
    ushort Value,
    ushort Index);
```

`C++`

```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in `Request`.

`Index`

Specifies the device-defined index, returned by a successful configuration request.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_VENDOR_INTERFACE` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more information. The library decodes several parameters from this packet.

OnVendorOther

TypeScript

```
// This method is not available in scripting environment
```

C#

```
void OnVendorOther(DateTime time,
    byte[] array,
    byte RequestTypeReservedBits,
    byte Request,
    ushort Value);
```

C++

```
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to Request, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in `Request`.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_VENDOR_OTHER` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more information. The library decodes several parameters from this packet.

OnAbortPipe

TypeScript

```
// This method is not available in scripting environment
```

C#

```
void OnAbortPipe(DateTime time,
    byte[] array,
    ulong PipeHandle);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`PipeHandle`

Specifies an opaque handle to the bulk or interrupt pipe. The host controller driver returns this handle when the client driver selects the device configuration with a `URB` of type `URB_FUNCTION_SELECT_CONFIGURATION` or when the client driver changes the settings for an interface with a `URB` of type `URB_FUNCTION_SELECT_INTERFACE`.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_ABORT_PIPE` is received. See `_URB_PIPE_REQUEST` in MSDN for more details.

OnBulkOrInterruptTransfer

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnBulkOrInterruptTransfer(DateTime time,
    byte[] array,
    uint payloadOffset,
    uint payloadSize);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`payloadOffset`

Payload offset, in bytes.

`payloadSize`

Payload size, in bytes.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER` is received. See `_URB_BULK_OR_INTERRUPT_TRANSFER` in MSDN for more details.

OnClassDevice

```
TypeScript
// This method is not available in scripting environment
```

```
C#
void OnClassDevice(DateTime time,
    byte[] array,
    byte RequestTypeReservedBits,
    byte Request,
    ushort Value);
```

```
C++
// This method is not available in native environment
```

Parameters

`time`

The time of the event.

`array`

Packet data.

`RequestTypeReservedBits`

Specifies a value, from 4 to 31 inclusive, that becomes part of the request type code in the USB-defined setup packet. This value is defined by USB for a class request or the vendor for a vendor request.

`Request`

Specifies the USB or vendor-defined request code for the device, interface, endpoint, or other device-defined target.

`Value`

Specifies a value, specific to `Request`, that becomes part of the USB-defined setup packet for the target. This value is defined by the creator of the code used in `Request`.

Description

Fired when `URB` packet with `urb.UrbHeader.Function == URB_FUNCTION_CLASS_DEVICE` is received. See `_URB_CONTROL_VENDOR_OR_CLASS_REQUEST` in MSDN for more details. The library decodes several parameters from this packet.

Enumerations

ConnectionState Enumeration

Symbol	Value	Description
`DeviceDisconnected`	`0x00000000`	Device is connected.
`DeviceConnected`	`0x00000001`	Device is disconnected. For example: flash drive is removed from computer.

Functions

ConfigureLibrary Function

ConfigureLibrary

```
TypeScript
// This method is not available in scripting environment
```

```
C#
// This method is not available in managed environment
```

```
C++
DWORD ConfigureLibrary(BOOL bInstall, BOOL bUI);
```

Parameters

`bInstall`

`TRUE` to install monitoring capabilities, `FALSE` to uninstall.

`bUI`

`TRUE` to display the progress dialog while configuring the system, `FALSE` to suppress it

Return Value

Returns the error code. See the Remarks section for details.

Description

Enables or disables the monitoring capabilities of the control. The function will restart all USB devices on the computer as part of its operation. You must be sure that no critical work is performed through one of these devices. If any USB device is being used at this time, the operation succeeds, but the function returns 1, indicating that the restart is required. In order to eliminate the need to restart the computer, make sure the USB devices are not being used before calling this function.

If you call this function, passing `TRUE` for the `bUI` parameter, the function displays the top-most progress dialog. The dialog shows the progress of operation. Please note however, that it does not although let the user to interrupt the process and does not provide the progress to the calling application. Depending on the number of installed USB devices on your computer, and their "nature", the process can take some time.