

# WDM Streaming USB Audio/Video Driver

May 23<sup>rd</sup> 2003

## 1 INTRODUCTION

With the introduction of USB2.0, supporting up to 480 Mbps (60MBytes/sec), the bandwidth bottleneck on USB1.1 interface for high quality video streaming has been solved.

This white paper introduces the basic concepts of WDM, KSA (Kernel Streaming Architecture), and USB 2.0 client driver architecture and describes how to write WDM video/audio capture and playback driver for USB 2.0 device. This is based on the development of a WDM capture/ playback driver for a USB2.0 device.

## 2 DEVICE DESCRIPTION

The device enables the streaming **video/audio capture and playback** in the host PC via USB. The video input to the host is compressed video while the audio is uncompressed. The device supports USB specification 1.1 and 2.0. It has a single configuration and multiple interfaces. The capture/playback interface is selected based on the application.

As soon as the device is plugged into the USB bus, the Host USB bus driver starts the enumeration process. The host enables the port and addresses of the USB device through the device's control pipe at the default address. The device identifies itself and downloads firmware and the USB descriptor tables over the USB cable from the host. The device enumerates again, this time as a device defined by the downloaded personality information. The device gets a unique USB address from the host. The host communicates with the device using this address and end point zero to find out the power and bandwidth requirements of the device. If the device requirements are acceptable to the host, it enables the configuration and the device driver will get loaded.

## 3 WINDOWS DRIVER MODEL AND KERNEL STREAMING ARCHITECTURE

WDM is the Windows Driver Model from Microsoft, incorporating *plug and play*, *power management*, and *advanced bus management* into a subset of the Windows NT device driver model. The other two driver models/formats are VXD (for Windows 9x/Me) and NT4 Kernel Mode (for Windows NT 4.0, 2000 and XP).

### 3.1 WDM

Windows 98SE, Windows 2000 (formerly Windows NT 5), Windows Me and Windows XP share a common driver model known as WDM and is nothing more and nothing less than a file format for the driver. A driver written to the WDM specifications will be source-code compatible on all Win9x platforms (starting with Win98SE) and Win2k. Most drivers are even binary compatible across these platforms enabling hardware companies develop a single kernel-mode driver.

A WDM driver is loaded and unloaded dynamically based on entries in the registry as shown in Fig 1. The plug-and-play manager together with the IO Manager allocate the resources and set up the environment required for Applications to communicate with the device using IRPs (IO Request Packets). Upon receiving the I/O requests in IRPs they are queued and processed asynchronously by the driver. The rich set of kernel-mode APIs defined in wdm.h of driver development kit (DDK) are used by the driver for accessing various services the driver requires. This makes WDM very flexible compared to other driver models like NDIS and SCSI where only a limited set of services are available.

WDM drivers support layering of devices so that a higher-level device object can intercept requests sent to a lower level device object or can implement its functionality by using lower-level drivers. For example in WDM a driver's functional device object is stacked on top of a lower-level physical device object which provides the bus functionality that was provided by the HAL component in Windows NT 4.

### **3.2 Skeleton of a standard WDM Driver:**

A WDM driver consists of an entry routine that is invoked by the I/O Manager after the PNP manager detects the device and loads the driver into the memory.

#### **3.2.1 DriverEntry**

**Prototype:**

```
extern "C" NTSTATUS DriverEntry (IN PDRIVER_OBJECT DriverObject,  
                                PUNICODE_STRING RegistryPath)
```

**Function:**

This routine receives a pointer to the driver object and fills in the function pointers in the driver object.

Typical function pointers are

1. DriverObject->DriverUnload
2. DriverObject->DriverExtension->AddDevice
3. DriverObject->MajorFunction[IRP\_MJ\_PNP]
4. DriverObject->MajorFunction[IRP\_MJ\_POWER]

and other major function pointers.

#### **3.2.2 The DriverUnload Routine**

**Prototype:**

```
VOID DriverUnload(PDRIVER_OBJECT DriverObject)
```

**Function:**

This function performs the clean up of all global initialization that DriverEntry performed.

### **3.2.3 AddDevice Routine**

**Prototype:**

NTSTATUS AddDevice(PDRIVER\_OBJECT DriverObject, PDEVICE\_OBJECT pdo)

**Function:**

This function is invoked for every devices detected by the PnP Manager and a device object is created and attached to the device stack.

### **3.2.4 Handling Plug and Play & Power IRPs**

**DriverObject->MajorFunction[IRP\_MJ\_PNP] = XxxDispatchPnp;**  
**DriverObject->MajorFunction[IRP\_MJ\_POWER] = XxxDispatchPnp;**

The above variables are initialized for handling Plug and Play and Power management with the respective functions from the driver.

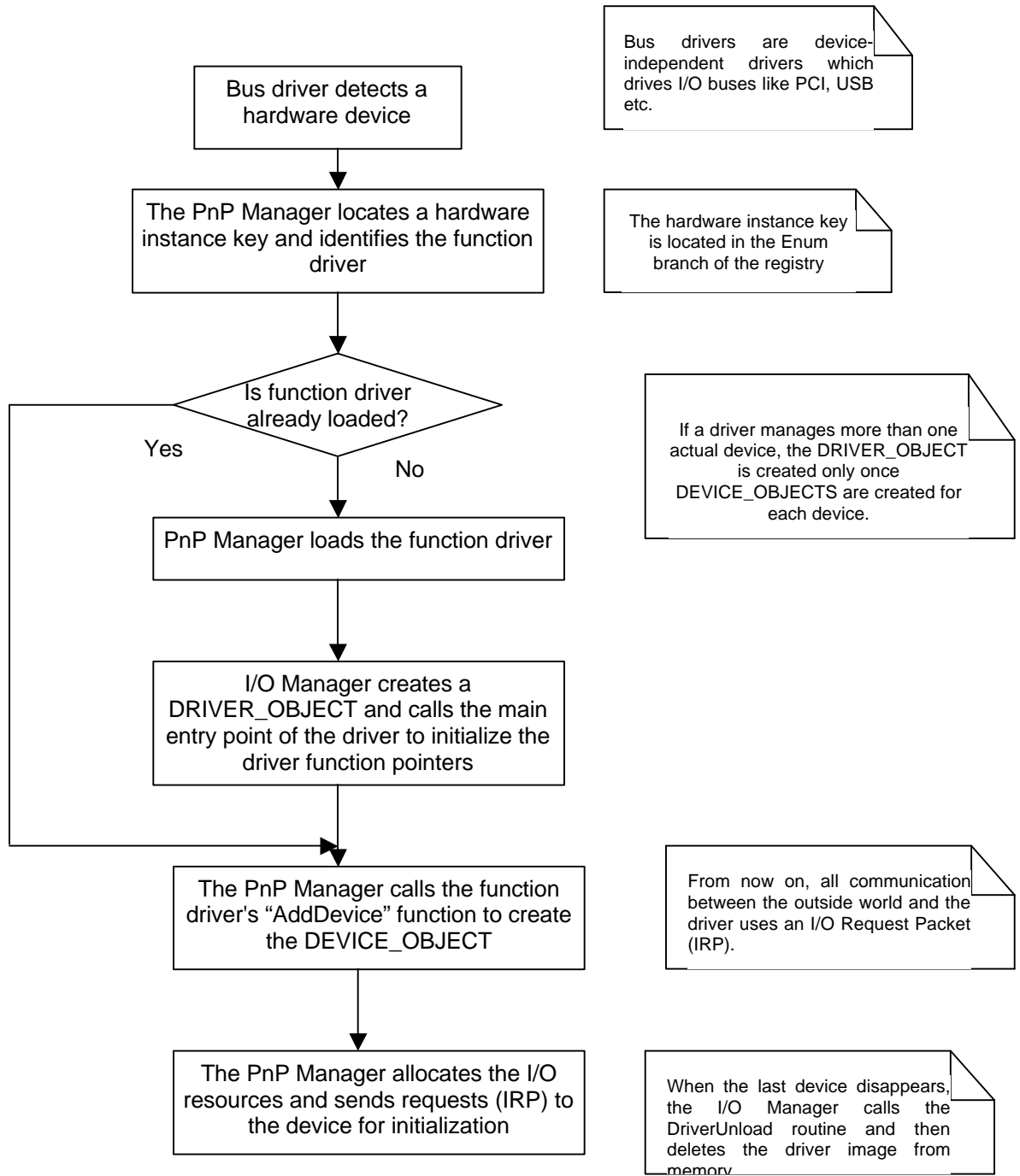


Figure 1: Life Cycle of WDM Driver

### 3.3 WDM Kernel Streaming

WDM Streaming has a kernel-mode streaming architecture that provides a high-bandwidth, low-latency communication mechanism between WDM device drivers and is used for audio and video devices at present. This architecture is similar to DirectShow, which divides the processing of multimedia tasks into a set of steps known as *filters*. Filters have a number of input and output *pins*, which connect them together to form a graph. A WDM Streaming device is represented as a user mode filter in DirectShow. Application interfaces and graph building of DirectShow interact with these user-mode filters of WDM streaming devices. After starting the graph, the kernel-mode components of the WDM Streaming devices communicate directly without transitions to and from the kernel mode thereby achieving high bandwidth and low-latency.

WDM Streaming filters uses the standard *i/o request packet* model in which the interfaces and methods used by DirectShow are replaced with IOCTL codes and property sets. The graph of connected kernel-mode WDM Streaming filters is built as a stack of layered device drivers. Higher-level devices can pass read or write requests to lower level devices as IRPs. However, the lower-level device cannot make unsolicited calls back up the stack. But a pair of DirectShow pins can use the COM QueryInterface mechanism to communicate in both directions at once.

### 3.4 USB STREAMING driver architecture

Typically, a minidriver that connects through the USB bus will be a client of the WDM Kernel Stream Subsystem on the upper end and the USB port driver on the lower end as shown in the following diagram.

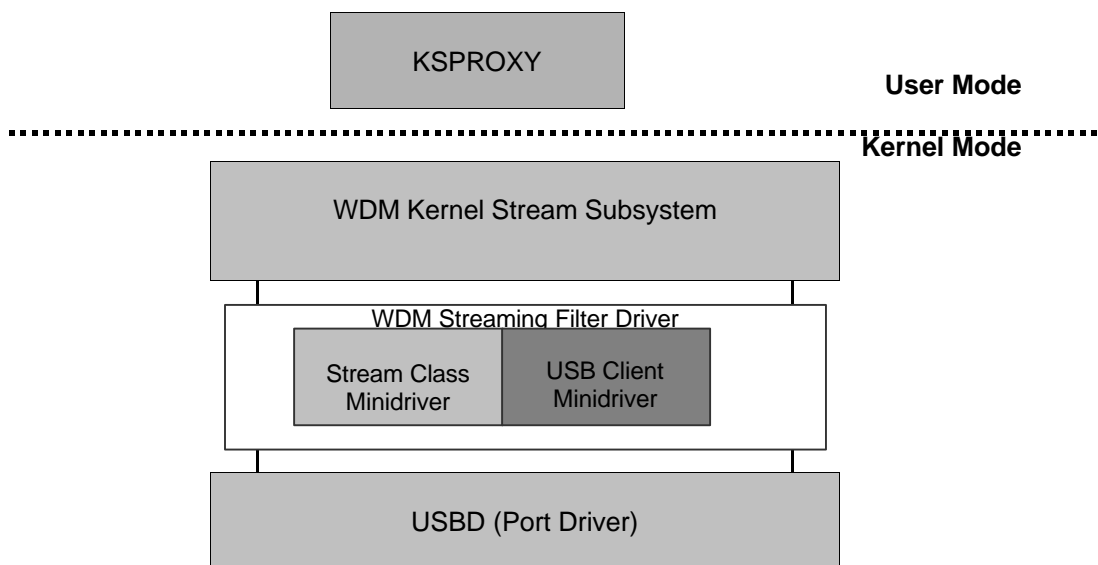


Figure 2: Simple USB client driver

The KSPROXY (*WDM User-Mode Stream Proxy*) presents a standard *DirectShow* filter for every *WDM Streaming Filter Driver* in the system. The proxy uses Device IO Control commands to the *WDM Kernel Stream Subsystem* to interact with *WDM Streaming Filter Driver* for enumeration, controlling properties, streaming data to/from the filter object etc. WDM streaming Driver Implementation is simplified due to the filter architecture.

The *WDM Kernel Stream Subsystem* and a *WDM Streaming Filter Driver* interacts with a predefined set of commands, calls and callbacks. Commands to *WDM Streaming Filter Driver* are contained in a structure called a *Service Request Block (SRB)*. SRB contains the information that a filter driver needs to complete a request. There are different implementations of processing SRBs in *WDM Streaming Filter Driver*: simultaneous processing of several SRBs or one SRB at a time.

## 3.5 USB Client Driver Classes

We used a NuMega DriverWorks for our implementation of USB client driver because the DriverWorks stream classes and USB classes implement dispatching mechanisms and methods for all the defined SRB command codes, allowing developers to implement commands by simply overriding these methods.

### 3.5.1 Stream Classes

NuMega DriverWorks provides the following three classes to make the development of stream minidrivers easy:

1. **class KstreamMinidriver:** This is the WDM *filter device* driver interface to the operating system. It is derived from KDriver and provides user mode components with a *DirectShow* compliant filter graph object. Windows Plug and Play constructs a WDM *filter device object* when a kernel-mode driver (streaming filter) is loaded. This class provides a default implementation for basic *WDM filter device object* commands, and dispatches commands to the appropriate **KStreamAdapter** object.
2. **class KStreamAdapter :** This class controls an individual hardware. **KStreamAdapter** is created by the **KStreamMinidriver** object. **KStreamMinidriver** forwards most of the commands received to the appropriate **KStreamAdapter** object. Implementation of **KStreamAdapter** object includes control of device specific properties, and should necessary **KStream** derived classes. Usually **KStreamAdapter** initializes and opens the stream objects and also controls the hardware.
3. **class Kstream:** This provides control and data handling for an individual media stream like stream control, stream properties, and stream data buffer management etc. The **KStream** class provides only default implementations for all stream commands. Necessary virtual methods are overridden to implement appropriate functionality.

### 3.5.2 USB Classes

All USB devices can be represented with three components namely a Logical Device, Interface and an EndPoint.

- **Logical Device:** USB bus driver communicates with the logical device over the default control pipe (End Point 0) for functions like accessing the device descriptor, setting the configuration etc.  
  
DriverWorks Class **KUsbLowerDevice** provides this functionality. This class creates an interface to the upper edge of the system USB bus driver.
- **Interfaces:** All USB devices have one or more interfaces representing a different subset of the device's capabilities. Multiple interfaces of a device can be enabled at the same time as long as same end points are not used.

DriverWorks Class **KUsbInterface** provides this functionality. An instance of this class will be created for each of the interfaces.

- **Endpoints:** Data flows in and out on from the physical device over Endpoints. This is similar to an I/O register on the device. The connection between an endpoint and the driver is referred as *pipe*. The USB specification defines four transfer types over the pipes: Control, Interrupt, Bulk, and Isochronous.

DriverWorks Class **KUsbPipe** provides this functionality. An instance of this class will be created for each of the pipes used for data transfer.

#### 4 USB CLIENT DRIVER

The USB client driver suite for the device consists of:

1. Audio driver (Capture and Playback)
2. Video driver (Capture and Playback)

The audio and video drivers support four different interfaces (video capture, video playback, audio capture and audio playback) each carrying out a specific function as shown in Figure 3.

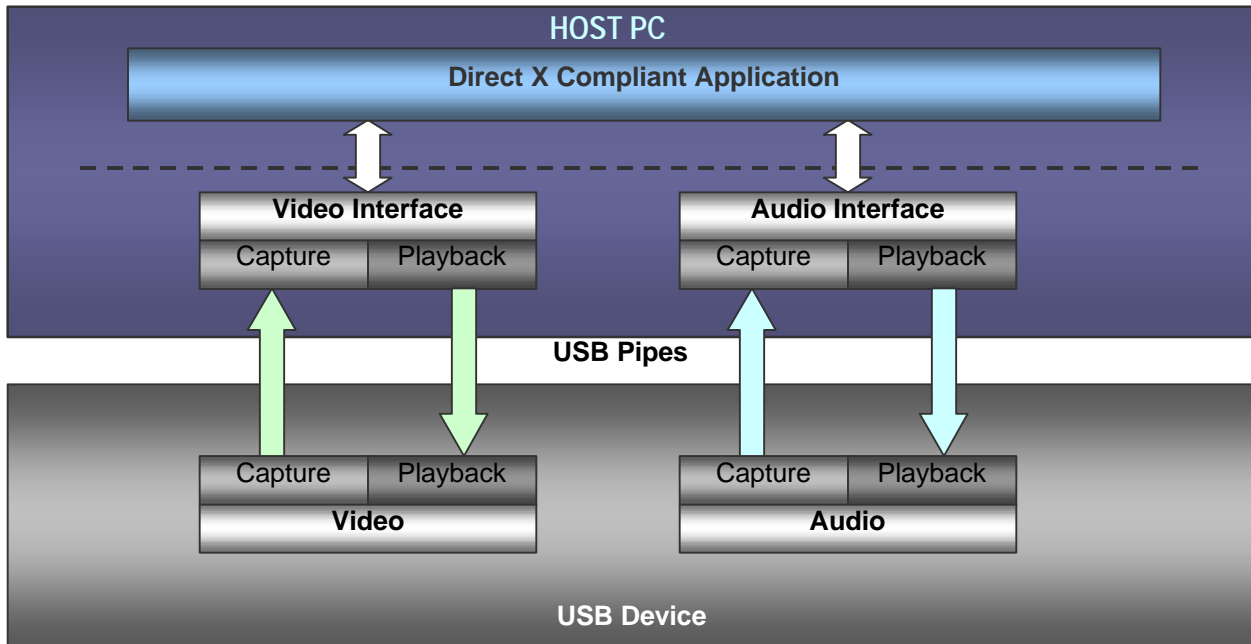


Figure 3: communication path between the USB Device and the HostPC.

## 4.1 Audio Driver

The audio driver supports Capture and Playback of audio streams and will make both audio capture and playback interfaces visible.

Audio Capture Interface is a part of Audio WDM streaming driver, which interacts with the audio interface of the device and streams audio into the PC for capturing or previewing as shown in Figure 4. It exposes the properties of the audio capture device to the OS. The audio is transferred from the device through the USB pipe.

The Audio Capture interface can be divided into two parts: one communicates with the USB device and the other with the OS. The streaming audio data is continuously received over the USB port and saved into a temporary buffer. Once an entire frame is received, the data is moved from the temporary buffer to the buffer received from the OS in the driver. It passes through the DirectX layer and is consumed by the Application.

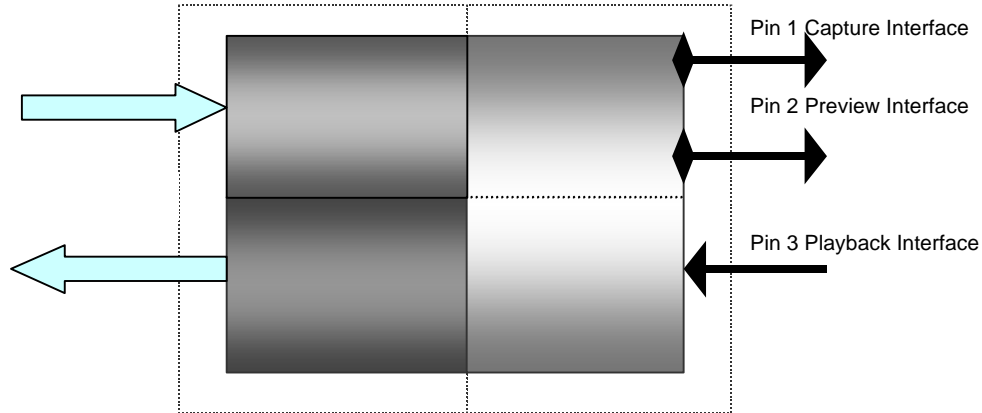
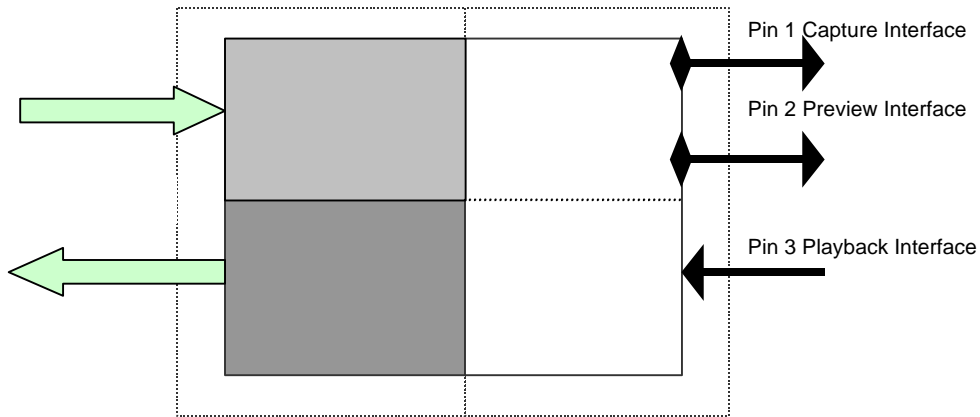


Figure 4: Audio Capture Driver

The Audio Playback Interface is a part of Audio WDM streaming driver, which interacts with the audio interface of the device and streams audio from the PC to the device. The audio data received from the Application via the DirectX layer is passed through the USB interface and reaches the audio rendering device.

## 4.2 Video Driver

The video mini driver supports capture/preview and playback of video streams and makes a capture and playback interfaces visible as shown in Figure 5.



*Figure 5: Video Capture Driver*

Video capture mini driver interface is a client of the Stream class and controls hardware devices that produce streams of video images providing the capability to capture video streams both compressed and uncompressed. It uses Kernel streaming services for better performance with lesser latency. The Streams created for each data type are represented as a WDM streaming pin. Application connects the Streams with a Kernel streaming filter or to a User mode filter.

The Video Capture interface can be divided into two parts: one communicates with the USB device and the other with the OS. The streaming video data is continuously received over the USB port and the driver allows viewing the captured video data over the preview pin in real time.

The Video Playback driver enables the device to accept video data from the .avi files saved on the host PC via the AVI splitter, by making a video playback device visible in the system. The USB interface of the driver receives the video data to send it across the USB bus to the device.

### 4.3 Development environment

1. Microsoft Visual C++ Version 6.0 Professional Edition.
2. Microsoft Driver Development Kit for Windows XP.
3. NuMega SoftICE Driver Suite.
4. MSDN Library.

### 4.4 Driver installation

The first step to install a WDM driver is to create the INF file. An inf contains many named sections with the section name enclosed in square brackets. There can be any number of sections in an INF file, and can be put in any order. The ".inf" file specifies the Version, Manufacturer, DestinationDirs, SourceDiskFiles, SourceDiskNames, Registry entries etc. As soon as the device is connected to the USB port, the bus driver will detect the device and the installation wizard will walk through the steps once the directory containing the "inf" file is selected.

## 5 CONCLUSION

Writing directly to the largely undocumented WDM Streaming interface is made easy by using the class driver – minidriver model to split out the common code. The class driver manages all the formal interfaces thereby relieving the mini-driver from those tasks. The mini-driver consists of the device-specific components and can call WDM services as necessary rather than being restricted to services offered by the class driver, thereby making the WDM driver coding easier.

## 6 REFERENCES

SI No	Description	Ver/Ref No
1	Microsoft DDK Documentation	Build 2600
2	NuMega DriverWorks Documentation	2.6
3	MSDN Library	April 2003
4	<a href="http://www.usb.org">www.usb.org</a>	
5	<a href="http://www.gdcl.co.uk">www.gdcl.co.uk</a>	

## 7 TRADEMARKS

"Microsoft", "Windows", "Windows NT", "Microsoft DDK", "MSDN Library" are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

"NuMega DriverWorks" and "NuMega SoftICE" are either registered trademarks or trademarks of Compuware Corporation in the United States and/or other countries.