

TUSBAudio

USB Audio 2.0 Class Driver for Windows

User Manual

Version: 2.29.0
Date: 13 February 2015

Thesycon Systemsoftware & Consulting GmbH
Werner-von-Siemens-Str. 2
D-98693 Ilmenau
Germany

Tel: +49 3677 8462 0
Fax: +49 3677 8462 18

<http://www.thesycon.de>

Contents

Table of contents	7
1 Introduction	11
2 Overview	12
2.1 Feature Summary	12
General Features	12
ASIO Features	12
WDM/DirectX Features	12
DSD Support	13
MIDI Features	13
Driver Control Panel	13
Firmware Upgrade	13
Driver Installer	13
Driver Package Customization	13
2.2 Driver Components	14
2.2.1 tusbaudio kernel-mode driver	14
2.2.2 tusbaudioks kernel-mode driver	14
2.2.3 tusbaudioasio.dll	14
2.2.4 tusbaudioapi.dll	14
2.2.5 TUSBAudioCplApp.exe	14
2.2.6 TUSBAudioDfu.exe	15
2.2.7 dfucons.exe	15
2.2.8 setup.exe	15
2.3 Driver Customization	15
3 System Requirements	17
3.1 Supported Operating Systems	17
3.1.1 Windows on Apple Hardware	17
3.2 Minimum Hardware Requirements	17
3.3 Compatibility and Stability Issues	18
3.3.1 Low-latency applications	18
3.3.2 Normal-latency applications	18
3.4 Known Hardware Issues and Possible Solutions	18

3.4.1	USB Signal Quality	19
	USB cables	19
	PCB mounted USB ports	19
	Front panel mounted USB ports	19
3.5	Known Software Issues and Possible Solutions	19
3.5.1	Kernel Mode Processing (background info)	20
3.5.2	Misbehaving Kernel-Mode Components	20
	W-LAN or Ethernet device drivers	20
	On-access virus scanners or personal firewall software	21
	Windows in-box drivers	21
	WHQL signed third party drivers	21
3.5.3	DPC Latency Checker Utility	21
4	Driver Operation	22
4.1	USB Audio Class Compliance	22
4.1.1	Audio Function Models	22
	Audio Streaming Interface IN + OUT	22
	Audio Streaming Interface OUT only	22
	Audio Streaming Interface IN only	22
4.1.2	USB Terminals	22
4.1.3	Sample Clock Source	23
4.2	Buffer Settings	23
4.2.1	USB Streaming Mode	23
4.2.2	ASIO Buffer Size	24
4.2.3	WDM Buffers	24
5	Driver Customization	25
5.1	Overview	25
5.1.1	Required Customization Steps	25
5.1.2	Optional Customization Steps	25
5.1.3	Multi-language Support	25
5.1.4	Additional Customization	25
5.2	Digital Signature	26
5.3	Preparing Driver Package Builder	26
5.3.1	Install SignTools	26
5.3.2	Check your Code Signing Certificate	27

5.3.3	Configure Certificate Variables	27
5.3.4	Prepare for GUID Generation	29
5.4	Using Driver Package Builder	29
5.4.1	Create your Driver Package Directory	29
5.4.2	Configure your Driver Package	29
5.4.3	Build your Driver Package	30
5.5	Customization Parameters Reference	32
5.5.1	VENDOR_NAME	32
5.5.2	PRODUCT_NAME	32
5.5.3	DRIVER_NAME_BASE	32
5.5.4	DEFAULT_INSTALL_DIR	32
5.5.5	DRIVER_INTERFACE_GUID	32
5.5.6	ASIO_DRIVER_GUID	32
5.5.7	ASIO_DRIVER_NAME	32
5.5.8	INF_SETUP_CLASS_GUID	32
5.5.9	INF_SETUP_CLASS_NAME	33
5.5.10	INF_VID_PID_[1..16]	33
5.5.11	INF_VID_PID_[1..16]_DESCRIPTION	33
5.5.12	INF_VID_PID_[1..16]_DESCRIPTION_KS	33
5.5.13	SHORTCUT_FOLDER	33
5.5.14	CPL_EXE	33
5.5.15	CPL_SHORTCUT_NAME	33
5.5.16	CPL_SHORTCUT_PARAMS	33
5.5.17	CPL_AUTOSTART_SHORTCUT_NAME	34
5.5.18	CPL_AUTOSTART_SHORTCUT_PARAMS	34
5.5.19	SPY_EXE	34
5.5.20	COPYFILE_[1..5]	34
5.5.21	COPYFILE_[1..5]_SHORTCUT_NAME	34
5.5.22	COPYFILE_[1..5]_SHORTCUT_PARAMS	34
5.5.23	LICENSE_TEXT_FILE	34
5.5.24	CUSTOM_BMP	34
5.5.25	API_DLL	35
6	Control Panel Customization	36
6.1	Overview	36

6.2	Driver Interface	36
6.3	User Interface Text Strings	36
6.4	Multi-language Support	37
6.5	Taskbar Notification Area Support	37
6.6	User Interface Pages	37
7	DFU Wizard Customization	39
7.1	Overview	39
7.2	Driver Interface	39
7.3	Supported Devices and Corresponding Firmware Files	39
7.4	Firmware Version Check	40
7.5	User Interface Text Strings	41
7.6	Multi-language Support	41
8	Driver Installation	42
8.1	Driver Package Contents	42
8.2	Driver Package Delivery	43
8.2.1	Why a self-extracting .exe should be used?	43
8.3	Driver Installation requires a connected device on some Windows versions	43
8.4	Driver Installation in Silent Mode	43
8.4.1	Command Line Parameters	44
8.4.2	Exit Codes	44
8.5	Driver Uninstallation	46
8.5.1	Command Line Parameters	47
8.5.2	Exit Codes	48
9	Known Issues	50
9.1	Intel 5/6/7 Series Chipsets: System hangs after device unplug	50
9.2	XMOS USB descriptors not compatible with USB 3.0 controllers	51
9.3	Sample rates greater than 200 kHz not supported by Windows	52
9.4	Installation problems on Windows XP	53
9.5	Driver does not work with Fresco Logic USB 3.0 host controllers	54
9.6	Windows does not display the 24 bit sample format for 88200 and 176400 Hz	55
9.7	Multiple USB audio devices on Intel USB 3.0 host controller not working	56
9.8	Windows 7 does not display the 32 bit sample format	57

10 Analyzing Problems	58
10.1 Monitoring Driver Statistics	58
10.2 Checking for Event Messages	58
10.3 Capturing Driver Log Messages	58
10.3.1 Installing and Configuring DebugView	58
11 XMOS Evaluation Kits	59
11.1 Device Firmware Upgrade	59
11.1.1 Preconditions	59
Installing the Factory Image	59
Generating a Firmware Upgrade Image	60
11.1.2 DFU Console	60
11.1.3 DFU API	60

References

- [1] Thesycon, TUSBAudio Reference Manual,
TUSBAudio_ReferenceManual.pdf
- [2] Universal Serial Bus Device Class Definition for Audio Devices Release 2.0,
http://www.usb.org/developers/devclass_docs/Audio2.0_final.zip
- [3] Universal Serial Bus Device Class Definition for Audio Data Formats Release 2.0,
http://www.usb.org/developers/devclass_docs/Audio2.0_final.zip
- [4] Universal Serial Bus Device Class Definition for Terminal Types Release 2.0,
http://www.usb.org/developers/devclass_docs/Audio2.0_final.zip
- [5] Universal Serial Bus Device Class Specification for Device Firmware Upgrade (DFU) 1.1
http://www.usb.org/developers/devclass_docs/DFU_1.1.pdf

1 Introduction

This manual describes properties and features of Thesycon's TUSBAudio device driver. The manual enables manufacturers of USB audio devices to configure the driver so that it works with their devices. The manual also provides some information (such as system requirements) that are useful for end users of those devices. However, this manual should not be passed to end user directly.

To develop a custom application based on the device driver, the TUSBAudio software development kit (SDK) is needed. The SDK must be licensed from Thesycon separately. The SDK includes a reference manual with information about the programming interface and sample application source code.

2 Overview

2.1 Feature Summary

General Features

- supports Audio class 1.0 and Audio class 2.0 devices
- supports standard sampling rates (depending on device capabilities):
44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz, 176.4 kHz, 192 kHz, 352.8 kHz, 384 kHz
- supports USB Audio Type I sample formats:
PCM 16 bit, PCM 24 bit, PCM 32 bit, FLOAT 32 bit
- supports stereo and multi-channel configurations with as many channels as the device implements

ASIO Features

- ASIO 2.2 compliant driver DLL
- sample formats (depending on device capabilities): PCM 24 bit, PCM 32 bit, Float 32 bit
- bit-perfect playback and recording through ASIO
- playback mix (simultaneous ASIO and WDM playback)
- supports both 32-bit and 64-bit ASIO host applications
- multi-client support (multiple ASIO applications in parallel)
- configuration of ASIO buffer depth via driver control panel
- ASIO DSD mode supported (playback and recording)
- DSD sample rates: DSD64 (2.8MHz), DSD128 (5.6MHz), DSD256 (11.3MHz), DSD512 (22.6MHz)

WDM/DirectX Features

- standard Windows sound interfaces: MME, DirectSound, WASAPI
- stereo and multi-channel playback and recording sound devices (depending on device capabilities)
- flexible sound device configuration, for example: 8-channel unit can either be exposed as 7.1 or 4 x stereo playback
- bit-perfect playback and recording through WASAPI
- volume and mute control through Windows standard GUI (depending on device capabilities)
- jack sensing (depending on device capabilities)
- PCM 16 bit, 24 bit, and 32 bit sample format (depending on device capabilities)

DSD Support

- native DSD mode through ASIO (see above)
- DSD over PCM (DoP) supported through ASIO and WDM

MIDI Features

- Windows compliant MIDI input and output ports (depending on device capabilities)
- Multiple applications can share a MIDI input port.
- Optionally, multiple applications can share a MIDI output port.

Driver Control Panel

- driver status and control via private programming interface (DLL)
- direct access to custom firmware extensions
- control panel source code available as part of SDK

Firmware Upgrade

- firmware upgrade according to DFU device class
- customizable firmware upgrade utility included

Driver Installer

- customizable driver setup included
- wizard style user interface for interactive use
- command line interface for silent mode (for integration into overall software setup)

Driver Package Customization

- automated customization procedure via scripts
- customization of USB VID/PID, file names, text strings, etc.
- code signing certificate for Microsoft Authenticode required

2.2 Driver Components

The TUSBAudio driver solution consists of various components which are described in more detail below.

2.2.1 tusbaudio kernel-mode driver

The tusbaudio kernel-mode driver consists of the files tusbaudio.sys, tusbaudio_x64.sys, tusbaudio.inf, and tusbaudio.cat. The kernel-mode driver implements core functionality such as USB communication and provides programming interfaces to other components.

2.2.2 tusbaudioks kernel-mode driver

The tusbaudioks kernel-mode driver consists of the files tusbaudioks.sys, tusbaudioks_x64.sys, tusbaudioks.inf, and tusbaudioks.cat. This kernel-mode driver is based upon tusbaudio and implements the interface to Microsoft's kernel streaming (KS). This way, the tusbaudioks driver supports all audio or MIDI related programming interfaces under Windows (MME, DirectSound, DirectKS, WASAPI).

2.2.3 tusbaudioasio.dll

This DLL is based upon tusbaudio and implements the ASIO driver interface. ASIO based applications (Cubase, Sonar, etc.) load this DLL to access the audio device according to the ASIO 2.2 specification.

tusbaudioasio.dll loads configuration information from custom.ini, which must be present in the same directory as the DLL itself.

2.2.4 tusbaudioapi.dll

This DLL provides the private programming interface supported by the tusbaudio kernel-mode driver. The programming interface consists of a set of C functions. To use the programming interface in custom applications such as a vendor-specific control panel, the TUSBAudio software development kit (SDK) must be licensed from Thesycon.

tusbaudioapi.dll loads configuration information from custom.ini which must be present in the same directory as the DLL itself.

2.2.5 TUSBAudioCplApp.exe

This is Thesycon's driver control panel. The control panel application is based on the programming interface exposed by tusbaudioapi.dll. The application is implemented in C++ and uses Visual C++ and ATL/WTL.

The control panel application is customizable through an XML file. Customers can ship this application to end users, or decide to create a vendor or product specific control panel according to their own requirements. In order to do this, the TUSBAudio software development kit (SDK) is

required. The SDK contains the complete C++ source code of the TUSBAudioCplApp application which may be used as a starting point.

2.2.6 TUSBAudioDfu.exe

This is a GUI-based application which implements Device Firmware Upgrade (DFU) functionality based on the the programming interface exposed by tusbaudioapi.dll. The application is implemented in C++ and uses Visual C++ and ATL/WTL.

The DFU application is customizable through an XML file. Customers can ship this application to end users, or decide to create a vendor or product specific variant according to their own requirements. In order to do this, the TUSBAudio software development kit (SDK) is required. The SDK contains the complete C++ source code of the TUSBAudioDfu application which may be used as a starting point.

2.2.7 dfucons.exe

This is a command line based utility which provides a front end to the Device Firmware Upgrade (DFU) functionality that is supported by the driver. The utility is based upon the programming interface exposed by tusbaudioapi.dll and is implemented in C++ using Visual C++. The C++ source code of dfucons is included in the TUSBAudio software development kit (SDK).

The dfucons utility is intended to be used by developers and support staff, and is not targeted to end users. Customers should not provide the utility to end users. A more user-friendly DFU application is TUSBAudioDfu (see above).

2.2.8 setup.exe

This is the driver installer Thesycon provides for TUSBAudio. The installer is configurable by means of setup.ini, which must be present in the same directory as setup.exe. For more information about the installer, refer to section 8.

Customers can implement their own installer, or integrate driver installation into an overall application installer.

2.3 Driver Customization

Thesycon's TUSBAudio driver is generic with respect to concrete products. The driver can be used (and shipped together) with many different products from various vendors. Any generic driver must be customized. Customization includes modification of USB VID and PID, choice of unique file names, assignment of unique identifiers (GUIDs) and modification of display names.

If no (or an incomplete) customization is done, the following situation can occur in the field:

1. User buys product A and installs it. Product A works.
2. Same user buys product B (from another vendor) and installs it. B ships with the same (non-customized) driver as A.

This situation results in a couple of potential problems:

- B driver .sys files overwrite A driver .sys files which reside in the Windows drivers directory. This can cause product A to stop working e.g. because a different driver version gets loaded now.
- Product A control panel detects a product B device and tries to work with it.
- Product B control panel detects a product A device and tries to work with it.
- Uninstallation of product A removes .sys files (and possibly other files) which causes product B to become non-functional.
- Uninstallation of product B removes .sys files (and possibly other files) which causes product A to become non-functional.

Important: Before the driver is shipped to end users, a customized driver package must be created. Never ship the original driver package provided by Thesycon to end users! Driver packages included in the TUSBAudio kits provided by Thesycon are sample packages that have been created to install and evaluate the driver together with evaluation boards.

For detailed information on the steps required to create a customized driver package, refer to section 5.

3 System Requirements

3.1 Supported Operating Systems

The TUSBAudio driver supports the Windows operating system versions listed below. The driver and its installer have been tested on these systems.

- Windows XP with SP3 (32 bit only)
- Windows Vista with SP2 (32 and 64 bit)
- Windows 7 (32 and 64 bit)
- Windows 8 (32 and 64 bit)
- Windows 8.1 (32 and 64 bit)

The server version corresponding to each of the listed operating systems is supported as well.

The minimum supported OS and service pack level is Windows XP SP3. Any older Windows version or service pack level is not supported, as before XP SP3 there are issues in the Microsoft USB bus driver stack that can cause problems.

Thesycon will test compatibility of the driver with any future service pack releases for one of the operating systems listed above and will release an updated driver, if required.

3.1.1 Windows on Apple Hardware

Thesycon supports Windows installed on PC hardware only. While it is possible to install Windows on a (Intel based) Mac this is considered a very specific use case that is not tested by Thesycon.

3.2 Minimum Hardware Requirements

The USB Audio class uses isochronous transfer mode to transfer the audio signals through USB. Early USB implementations and chip sets did not support isochronous transfer mode correctly and had a couple of other issues. Therefore, it is necessary to define minimum chip set requirements. But most users don't know which chip set is inside their PC or laptop and it may not be easy find this out. Hence, it makes sense to define a limit for the age of the computer.

Based on this, Thesycon defines the minimum PC hardware requirements as follows:

- PC or laptop manufactured after January 2006
- Intel Core 2 @1.6 GHz, or AMD equivalent
- 1 GB main memory

However, it is important to note that there is no guarantee that USB based audio streaming will work on any PC that fulfills the above requirements. Because there are so many different PC configurations, there is always a risk that hardware and/or software issues will cause problems. Some examples are given in the next section.

3.3 Compatibility and Stability Issues

USB audio devices have stronger requirements for USB hardware and software layers than other USB devices. A faulty hardware component, e.g. USB cable or USB port, may not have an impact on standard USB devices such as a FLASH drive but can be catastrophic for a USB audio device. USB hardware requirements are discussed in more detail in section 3.4.

Due to the real-time nature of USB audio streams there are also requirements for the time characteristics of the operating system and third party software components installed on the system. Software components that make real-time behavior of the operating system worse are not compatible with audio streaming applications in general. However, in this context it is important to note that real-time requirements depend directly on audio latency requirements. If low audio latency is required (e.g. to create a monitor mix in the PC and route it back to speakers) then the operating system must be able to fulfill resulting timing requirements of the driver. If audio latency is not critical (e.g. in case of music playback) then timing requirements of the driver are relaxed, which increases compatibility with other applications and drivers significantly.

Thesycon's TUSBAudio driver allows users to adjust internal buffer depths to find a trade off between audio latency and compatibility with a given system. Below, two typical scenarios are discussed. For details about buffer size configuration, see also section 4.2.

3.3.1 Low-latency applications

Audio latency is critical if an audio signal is routed into the PC, processed in the PC and then routed back to speakers. A typical scenario is a monitor mix created by a multi-track recording application such as Cubase. For such applications the driver's USB streaming buffer depth should be set to 4 milliseconds or less (see also section 4.2). In this configuration a couple of issues can occur because the given system may not be able to handle the resulting real-time requirements. A discussion of possible problems and incompatibilities is given in section 3.5.

3.3.2 Normal-latency applications

In case of a playback-only or recording-only scenario, audio latency is not critical. For such applications the driver's USB streaming buffer depth should be set to 16 milliseconds or more (see also section 4.2). This will reduce the risk of audio distortions caused by other software components in the system significantly. Issues discussed in section 3.5 may still occur but this will happen only in extreme cases. If the driver is configured in this way, it will be compatible with a larger number of Windows systems. Normally, it will not be necessary to optimize a system for audio streaming according to the hints given in section 3.5.

3.4 Known Hardware Issues and Possible Solutions

This section discusses some hardware-related issues Thesycon has discovered in the past and provides some tips on problem analysis and possible solutions or workarounds.

3.4.1 USB Signal Quality

Isochronous transfer mode uses error checking but no retransmission in case of CRC errors. Electrical noise on USB signals causes CRC errors and thus data loss. This leads to audio signal distortions (clicks). This means that a USB audio device can work only if USB signal quality is good and no CRC errors occur. Most other USB device types (e.g. FLASH drive, printer) are based on bulk transfer mode, which uses automatic retransmission in case of errors. These kind of devices are much more tolerant with respect to USB signal distortion. For this reason, it is possible that a mouse, keyboard, FLASH drive, printer etc. works well on a given USB port using a given cable while an audio device does not work with the same port or cable.

Below is a list of possible sources for this kind of problems.

USB cables

Quite often, the USB cable (or its connectors) is the cause for USB signal distortions. Some cables available in the market are not suited for USB 2.0 high-speed communication (480 Mbps). Also the maximum allowed cable length of 5 meters should not be exceeded.

Solution: Try using a different cable. Try a shorter cable (less than 2 meters).

Tip: Stay away from special USB cable offerings optimized for audio, or cables which include additional functionality such as status LEDs.

PCB mounted USB ports

Theyson found that on some PC main boards (or laptops) signal quality of some USB ports is insufficient for isochronous streaming. The cause could be that on the PCB USB signals are routed close to a switching voltage regulator, for example.

Solution: Try using a different USB port to connect the audio device.

Front panel mounted USB ports

External USB ports (mounted on a front panel or elsewhere in the PC case) are a possible source of USB signal distortion. Quality of cables or connectors used to connect the external USB port with the main board could be insufficient, or internal cables are placed close to the power supply or other sources of electrical noise.

Solution: Try using a USB port that is mounted directly on the main board.

3.5 Known Software Issues and Possible Solutions

This section discusses some software-related issues Theyson has discovered in the past and provides some tips on problem analysis and possible solutions or workarounds.

Note that the discussion in this section applies to scenarios where the driver has been configured for low audio latency (see also section 3.3).

3.5.1 Kernel Mode Processing (background info)

A USB audio driver must be able to process incoming and outgoing isochronous data streams in real time. The driver is called periodically by the operating system kernel to perform its processing. The period of the calls is determined by the buffer size the driver uses to exchange data with the USB. This buffer size is adjustable and is referred to as "USB Streaming Mode", see also section 4.2.1.

If one of the processing calls is delayed by more than one buffer size interval, data loss will occur. Delayed processing calls are the most common source for audible signal distortions. This problem is referred to as DPC latency issue. Deferred procedure call (DPC) is Microsoft's term for processing callbacks issued by the kernel. In this context, "DPC latency" is not to be confused with audio latency which is not directly related.

Because Windows is not a real-time operating system, the Windows kernel cannot guarantee that a driver will be called in time. Any kernel-mode software component, including other device drivers, can monopolize the CPU, which prevents the kernel to perform its periodic processing in time. A typical situation is that a device driver is called by the kernel (e.g. from a timer service) to check the state of its hardware. The driver keeps the CPU busy for several milliseconds by spinning in a loop and polling hardware status, for example. The kernel cannot call any other driver before this driver returns control. If the misbehaving driver keeps the CPU busy for an interval that exceeds the streaming buffer depth of the audio driver, the audio driver experiences a delayed processing call which leads to an interruption in the audio data stream.

Note that at this level of kernel-mode processing there is no priority based multi-threading. The Windows kernel executes calls into various device drivers sequentially which may lead to delayed calls as described above. The design is based on the assumption that no driver stalls the CPU and any call into a driver returns quickly. Microsoft documentation states that drivers should spend no more than 100 microseconds in any processing call issued by the kernel. While most device drivers fulfill this requirement there are a couple of misbehaving drivers in the market. If such a driver is used in parallel with an audio driver then audio dropouts can occur.

Note also that a multi-core CPU is not a solution to the problem. This is because the kernel (and bus drivers) arbitrarily assign kernel processing calls (DPCs) to CPU cores. A device driver cannot reserve a CPU core for its own (time-critical) processing.

3.5.2 Misbehaving Kernel-Mode Components

This section lists some potential sources of DPC latency issues. The information is based on Thesycon's own experience and on user feedback.

W-LAN or Ethernet device drivers

Quite often it can be observed that device drivers for W-LAN adapters monopolize the CPU in kernel mode as described in the previous section. A few Ethernet drivers also have such issues. Thesycon's `dpclat` utility (see next section) can be used to analyze the system behavior when W-LAN is enabled or disabled.

Solution: Try to find an updated W-LAN driver, or try an older version of the driver. If no suited driver can be found, disable W-LAN (or Ethernet) while audio streaming is running.

On-access virus scanners or personal firewall software

Normally, this kind of software includes some kernel-mode component to perform filtering or scanning work in the kernel. Often, such components keep the CPU busy for long periods which causes kernel processing calls (DPCs) to be delayed. Thesycon's dpclat utility (see next section) can be used to analyze the system behavior when on-access scanning (or filtering) is enabled or disabled.

Solution: Disable or uninstall the software. Try using a different product with similar functionality.

Windows in-box drivers

Generally it can be stated that the set of device drivers that ships with Windows behaves well. If DPC latency problems occur then they are typically caused by third party drivers that are to be installed for hardware which Windows does not support by default. So if a hardware component is supported by an in-box driver then this driver is to be preferred and a third party driver should not be installed.

WHQL signed third party drivers

Many vendors submit their drivers to Microsoft's Windows Hardware Quality Lab (WHQL) to get them certified and signed. All drivers available on Windows Update servers are WHQL signed. But the fact that a given third party device driver is WHQL certified is not a guarantee for correct DPC runtime behavior. WHQL tests currently do not enforce proper DPC behavior.

3.5.3 DPC Latency Checker Utility

Thesycon provides a free utility that enables users to check the behavior of a given system. The utility performs a statistical analysis of kernel-mode processing calls which are called deferred procedure calls (DPC). For more information about the utility checkout <http://www.thesycon.de/dpclat>

The dpclat utility should be executed when the audio device is not connected to the system. The utility shows the maximum delay of processing calls the audio driver may experience on the given system. If excessive DPC latencies are observed then try to identify the culprit by using a trial and error approach. Disable/Enable devices in Device Manager (one at a time) and check whether the results shown by dpclat change.

Note that the dpclat utility does not support Windows 8 or newer operating systems. For these systems, Thesycon recommends to use the LatencyMon utility which provides similar functionality. For more information on LatencyMon checkout <http://www.resplendence.com/latencymon>

4 Driver Operation

4.1 USB Audio Class Compliance

The TUSBAudio device driver works with devices that implement the USB Audio Class 2.0 specification [2]. The class specification provides a very sophisticated framework and allows to build arbitrarily complex audio devices.

The TUSBAudio driver is designed to work well with high-quality audio devices typically used by audio professionals or HIFI enthusiasts. The driver supports a subset of the functionality available in the USB Audio Class specification. Some basic assumptions and restrictions are documented in the following subsections.

4.1.1 Audio Function Models

Depending on the capabilities of the audio function the TUSBAudio driver supports various modes of operation.

Audio Streaming Interface IN + OUT

The device provides both input (record) and output (playback) channels. The driver receives an isochronous input stream and sends an isochronous output stream. The driver derives the sample clock from the isochronous input stream. A feedback endpoint, if present, will not be activated.

Audio Streaming Interface OUT only

The device provides output channels only (playback only). The driver sends an isochronous output stream to the device. A feedback endpoint is required. The driver uses the feedback data to recover the sample clock.

Audio Streaming Interface IN only

The device provides input channels only (record only). The driver receives an isochronous input stream from the device. The driver derives the sample clock from the isochronous input stream.

Note: Because most ASIO based applications such as Cubase require at least two output channels, in this mode the driver provides two dummy output ASIO channels. Samples passed to these channels will be discarded and not be sent to the device.

4.1.2 USB Terminals

The TUSBAudio driver supports up to one USB input terminal and up to one USB output terminal. If a device exposes more than one USB terminal (or Audio Steaming Interface respectively) per direction then those additional terminals will be ignored.

This restriction results from the fact that under Windows it is very difficult (or impossible) to set up multiple isochronous endpoints that run in parallel and are fully synchronized to each other.

Because ASIO requires all channels to be fully synchronized, all input or output channels must be mapped to one USB isochronous endpoint.

4.1.3 Sample Clock Source

The TUSBAudio driver supports clock entities as defined in [2], namely Clock Source Entity and Clock Selector Entity. The driver supports clock source switching through its private programming interface.

However, a basic assumption is that all input and all output channels are driven by the same clock source. In other words, both the USB input terminal and the USB output terminal must be connected to the same clock entity, which is typically a clock selector. This assumption is based on the fact that ASIO requires all input and output channels be synchronized to the same sample clock source.

4.2 Buffer Settings

The driver has two layers of buffering: USB streaming and ASIO. The configuration of these buffer layers is described in more detail in the next subsections.

4.2.1 USB Streaming Mode

The USB streaming layer is configurable through a setting called "USB Streaming Mode". The USB Streaming Mode defines the buffer depth in the USB streaming layer in terms of milliseconds, not samples. Consequently, the buffer depth does not depend on the current sample rate.

The USB Streaming Mode can be set to one of the following values:

- **Minimum Latency** = 1 millisecond
- **Low Latency** = 2 milliseconds
- **Standard** = 4 milliseconds
- **Relaxed** = 8 milliseconds
- **Reliable** = 12 milliseconds
- **Safe** = 16 milliseconds
- **Extra Safe** = 32 milliseconds

A buffer depth of 1 millisecond is the smallest value supported by the underlying Microsoft bus drivers.

The given PC must be able to process kernel-mode DPCs within the configured buffer depth interval. Otherwise dropouts can occur. Therefore, the USB streaming buffer depth must be adjusted according to the capabilities of the given Windows system. For more background information on kernel-mode DPCs, refer to section 3.5.

4.2.2 ASIO Buffer Size

The ASIO buffer is used to exchange sample data between the driver and an application (DAW). The ASIO buffer size is adjustable. However, because the ASIO buffer layer is driven by the USB streaming layer there is a dependency on the USB Streaming Mode setting. ASIO can work without dropouts only if the following condition is met:

ASIO buffer size (in ms) \geq USB streaming buffer depth (in ms)

By convention the ASIO buffer depth is specified in terms of samples which creates another dependency on the current sample rate. For example, if USB Streaming Mode is set to Standard then the minimum ASIO buffer depth is 4 milliseconds which corresponds to 176 samples at 44100 Hz and 384 samples at 96000 Hz. Usually, an ASIO buffer size (in terms of samples) that is a power of two is preferred. In most DAWs sample processing is more efficient if such an "even" number is chosen. So in the above example we round up to the next power of two and end up with 256 samples at 44100 Hz and 512 samples at 96000 Hz.

The driver internally does not perform buffer size checks and does not enforce an ASIO buffer size that still works with the current USB Streaming Mode setting according to the condition defined above. This logic is implemented in the control panel. The control panel allows the user to pick one of the power-of-two numbers between 64 and 4096 and then checks if this works with the current USB Streaming Mode setting (The driver provides an API function which performs this check internally). If the selected ASIO buffer is too small for the current setting then the control panel displays a warning message. In this case the user should pick a larger ASIO buffer size value.

As a consequence of the convention of specifying the ASIO buffer size in terms of samples the user has to adjust the ASIO buffer size every time the sample rate in a DAW is changed.

4.2.3 WDM Buffers

Windows uses a separate buffer layer to exchange sample data with the driver. This is done through the so-called WDM interface which is the basis for all other Windows sound programming interfaces such as MME, DirectSound and WASAPI. The size of the WDM buffers is defined by Windows or the application that is using one of the above programming interfaces. The driver does not provide a mean to adjust the WDM buffer size.

Thus, the latency that is achievable through WDM depends on the USB Streaming Mode setting (see section 4.2.1 above) and the WDM buffer size selected by Windows or the application itself.

5 Driver Customization

5.1 Overview

The driver supports numerous features that enable a licensee to create a customized device driver package. A driver package which is shipped to end users **must be** customized. This is required in order to avoid potential conflicts with other products of other vendors that are also using this driver. See also section 2.3 for a discussion of the reasons why a full driver customization is absolutely required.

5.1.1 Required Customization Steps

Required customization steps are:

- Modification of product name shown in the installer,
- Modification of file names of all driver executables and .inf files,
- Modification of text strings shown in the Windows user interface, e.g. Device Manager,
- Definition of various unique software interface identifiers (GUIDs).

5.1.2 Optional Customization Steps

Optionally, the following customization steps can be implemented by the licensee:

- Adaptation of driver behavior for a specific device,
- Custom icon shown in the driver control panel,
- Custom icon shown in the firmware upgrade wizard,
- Adaptation of behavior of the firmware upgrade wizard,
- Custom bitmap shown in the driver installer.

5.1.3 Multi-language Support

Optionally, the following modifications can be implemented by the licensee:

- Multi-language support in the driver control panel (by default English is implemented only),
- Multi-language support in the firmware upgrade wizard (by default English is implemented only).

5.1.4 Additional Customization

The following modifications can be implemented by Thesycon only:

- Multi-language support in the driver installer.

5.2 Digital Signature

Windows Vista and higher Windows versions have a new feature to verify the vendor of a software component. The vendor adds a digital signature to a software component to identify itself. This signature grants that the software was signed by the vendor and that the software was not modified after it was signed.

Any 64-bit version of Windows Vista or higher **requires** that a kernel-mode device driver is digitally signed using a so called code signing certificate. Windows x64 will not load un-signed drivers. For background information about code signing, please refer to the document "Kernel-Mode Code Signing Walkthrough" available on the Microsoft web site.

To add a digital signature to a software component, the vendor must own a certificate that supports the Microsoft Authenticode technology (sometimes called Microsoft Authenticode Digital ID). Such a certificate can be purchased from several certification authorities (CA), for example VeriSign and GlobalSign. For details you may refer to

<http://www.symantec.com/code-signing/microsoft-authenticode/>
or to

<https://www.globalsign.com/code-signing/>

Alternatively, search the Web for for

"VeriSign Code Signing Certificates for Microsoft Authenticode".

Note that you need a certificate that uses the SHA1 hashing algorithm. Windows 7 supports signatures created with the SHA1 hashing algorithm only and does not support the SHA256 hashing algorithm. Windows 8 supports both the SHA1 and the SHA256 hashing algorithm.

5.3 Preparing Driver Package Builder

Thesycon provides a set of batch scripts and tools that generate driver packages and all required customization files automatically. This tool set is called the Driver Package Builder.

Before the scripts can be used, the steps described in the following subsections must be executed on the machine on which Driver Package Builder will run.

5.3.1 Install SignTools

Together with the TUSBAudio customization kit, Thesycon provides a package which includes all tools required to digitally sign driver files.

To install the SignTools package, run `SignTools_v1.x.x.exe` (1.x.x stands for the current version), contained in the `signing` subdirectory of the customization kit. This will install all the required tools into a directory of your choice and set the `SIGNTOOLS_1_x_x` environment variable to point to this directory.

The SignTools package includes current versions of the following Microsoft tools:

- `signtools.exe` (part of WDK)
- `inf2cat.exe` (part of WDK)
- `certmgr.exe` (part of WDK)

- pvk2pfx.exe (part of WDK)
- INF checker (part of WDK)

5.3.2 Check your Code Signing Certificate

A code signing certificate is **required** to build a driver package that installs on Windows x64. Refer to section 5.2 for more information. You cannot use the Driver Package Builder scripts without a valid code signing certificate.

The certificate to be used for code signing needs to be present in the certificate store of the build machine. For information on how to get the certificate and how to import it into the certificate store, please refer to instructions published by your certificate provider.

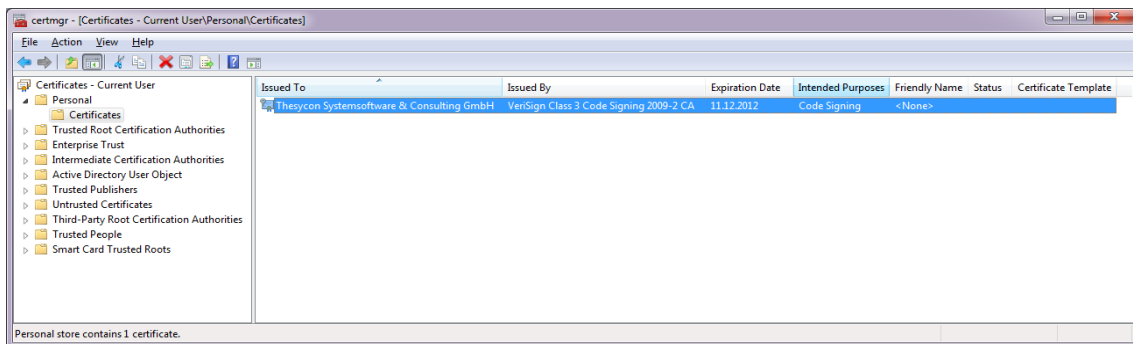


Figure 1: Certificates Microsoft Management Console

Before you continue, you should check if the certificate is present in the Personal folder of the certificate store of the machine in which you want to run Driver Package Builder. To do this, click Start - Run and enter `certmgr.msc`. This launches the Certificates Microsoft Management Console. The Personal folder should contain your certificate and should look similar to figure 1.

To verify that the certificate is valid for code signing, double-click on the certificate to open Details view. The Certificate Information dialog should look as shown in figure 2. For code signing, it is required that the private key that corresponds to the certificate is available. This is indicated by the key icon at the bottom of the General page and the statement "You have a private key that corresponds to this certificate.". If the key icon is missing, only the public key of the certificate is installed and code signing will not be possible.

You should also check that the field "Signature hash algorithm" on the Details tab is set to sha1. Windows 7 supports the SHA1 hashing algorithm only, see also section 5.2.

5.3.3 Configure Certificate Variables

NOTE: A code signing certificate is required to build a driver package that installs on Windows x64. Refer to section 5.2 for more information. You cannot use the Driver Package Builder scripts without a valid code signing certificate.

Edit the `set_vendor_certificate.cmd` script contained in the `signing` subdirectory of the customization kit. This script must refer to your certificate in the certificate store described in the previous step. In `set_vendor_certificate.cmd`, set the variables described below

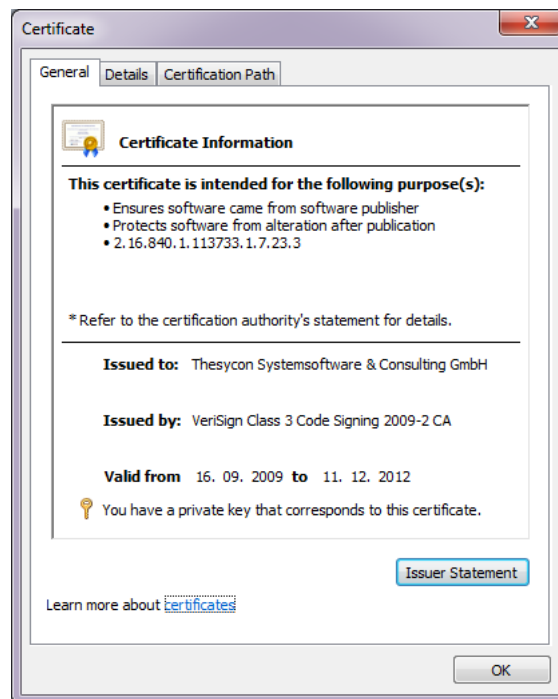


Figure 2: Certificate Information

according to your certificate. See also the comments and instructions given within the script itself. Note that alternatively you can add the variables to your build machine's environment.

- **VENDOR_CERTIFICATE**

This variable needs to be set to your certificate's name exactly as shown in `certmgr.msc`. Do not surround the string with quotation marks. If your certificate name includes special chars, quote each of them with a preceding escape character (^).

Example 1:

```
set VENDOR_CERTIFICATE=Best Audio Devices Inc.
```

Example 2:

```
set VENDOR_CERTIFICATE=Thesycon Systemsoftware ^& Consulting GmbH
```

- **VENDOR_CROSS_CERTIFICATE**

This variable specifies the full path and file name of the cross certificate file to be used. The cross certificate establishes a trust relationship between Microsoft and your certificate issuer. For latest information on cross certificates, see the article "Cross-Certificates for Kernel Mode Code Signing" in the MSDN library:

<http://msdn.microsoft.com/en-us/library/windows/hardware/gg487315.aspx>

Alternatively, ask your certificate provider which cross certificate is to be used.

In case of a VeriSign certificate, you can keep the default shown in `set_vendor_certificate.cmd`.

- `SIGNTOOL_TIMESTAMP_URL`

This variable specifies a trusted time stamp server to be used when the signature is created. Note that a signature should always be timestamped. In case of VeriSign, you can keep the default shown in `set_vendor_certificate.cmd`. If you are using a different certificate provider, refer to the documentation published by this provider.

5.3.4 Prepare for GUID Generation

Various Globally Unique Identifiers (GUID) are needed for the customization procedure. A GUID can be created easily by using Microsoft's `guidgen.exe` utility. This utility is part of the Microsoft Visual Studio 2005 and Visual Studio 2008 distribution. Note that it is not included in Visual Studio 2010.

The `guidgen.exe` utility can also be downloaded from this URL:

<http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=17252>

5.4 Using Driver Package Builder

The Driver Package Builder batch scripts are located in the `DriverPackageBuilder` subdirectory of the customization kit. To create your own customized driver package, please follow the steps described in the following subsections.

IMPORTANT: These steps are mandatory and must be executed. Otherwise name collisions and other problems can arise in the field.

5.4.1 Create your Driver Package Directory

Define a driver package identifier, e.g. `MyProductUSBAudioDriver`, and create a subdirectory with that name under the `DriverPackageBuilder` directory. Copy all files contained in the `XMOS_EVAL_KITS` directory to the new directory.

5.4.2 Configure your Driver Package

The driver package is configured by means of a set of variables defined in `setvars.cmd`. To configure your driver package, enter the subdirectory created in the previous step and edit the file `setvars.cmd`.

The following step-by-step procedure includes the major customizations steps. Make sure that you check the value of each variable that is contained in `setvars.cmd`. For a detailed description of each variable, refer to section 5.5.

1. Set vendor and product strings

Adapt the following strings to match your company and product:

```
VENDOR_NAME=  
PRODUCT_NAME=  
DRIVER_NAME_BASE=
```

```
DEFAULT_INSTALL_DIR=  
ASIO_DRIVER_NAME=
```

2. Generate unique software interface identifiers

By using the `guidgen.exe` utility (see section 5.3.4), generate a fresh GUID **for each** of the following variables:

```
DRIVER_INTERFACE_GUID=  
ASIO_DRIVER_GUID=  
INF_SETUP_CLASS_GUID=
```

3. Adapt USB VIDs and PIDs

Each product (device model) is unambiguously identified by its USB vendor ID (VID) and USB product ID (PID). The `setvars.cmd` script allows you to define up to 16 models which will all be supported by the same driver package. Set one or more of the `INF_VID_PID_x` variables to match with your device's VID(s) and PID(s). Be careful to get the special format of the `INF_VID_PID_x` string right. See also the comments and examples within `setvars.cmd`.

Set `INF_VID_PID_x_DESCRIPTION` and `INF_VID_PID_x_DESCRIPTION_KS` to your product's name. `INF_VID_PID_x_DESCRIPTION` will be shown in Device Manager, `INF_VID_PID_x_DESCRIPTION_KS` will be shown in Device Manager and in Windows 7 Playback/Recording devices panel.

Clear all unused `INF_VID_PID_x` variables, e.g.:

```
set INF_VID_PID_5=  
set INF_VID_PID_5_DESCRIPTION=  
set INF_VID_PID_5_DESCRIPTION_KS=
```

4. Remove Thesycon's applications

Thesycon ships together with the driver a sample control panel and a Spy utility. These should be removed from the driver package by clearing these variables:

```
set CPL_EXE=  
set SPY_EXE=
```

5. Include your own control panel (optional)

If you want to include your own control panel (or other applications) in the driver package then set `CPL_XXX` variables accordingly. Any executables referenced by these variables need to be copied manually to the `\bin\release` and `\bin\debug` directories under `DriverPackageBuilder`.

5.4.3 Build your Driver Package

Open a Windows console window (`cmd.exe`) in the `DriverPackageBuilder` directory. From the command line, run the `create_drvpackages.cmd` script passing your driver package identifier as an argument. The driver package identifier is the name of the subdirectory created in the first step (section 5.4.1 above).

Example:

```
create_drvpackages.cmd MyProductUSBAudioDriver
```

Your customized driver package will be created under the `DriverPackages` subdirectory of the customization kit. This subdirectory contains the driver installation package in uncompressed form. In addition, a self-extracting and digitally signed archive (.exe) will be created. You can either ship the driver installation package in uncompressed form or as a single self-extracting .exe. Refer to section [8.2](#) for more information.

5.5 Customization Parameters Reference

5.5.1 VENDOR_NAME

The vendor of the driver package. This string is used as a provider and manufacturer string (.inf file) and to create the default installation directory.

5.5.2 PRODUCT_NAME

The name of the product that uses the driver package.

5.5.3 DRIVER_NAME_BASE

The common part of the name of the driver package files.

This string must not include spaces or special characters.

5.5.4 DEFAULT_INSTALL_DIR

The default destination directory on the user's system. The path is relative to the Program Files directory. Your company name and product name should be included in the path, for example `MyCompany\MyProduct`.

5.5.5 DRIVER_INTERFACE_GUID

Unique identifier for the driver interface. Create a fresh GUID using `guidgen.exe` (see section 5.3.4).

It is very important to use a fresh GUID here.

5.5.6 ASIO_DRIVER_GUID

GUID that is used for ASIO driver registration. Create a fresh GUID using `guidgen.exe` (see section 5.3.4).

It is very important to use a fresh GUID here.

5.5.7 ASIO_DRIVER_NAME

The friendly name of the ASIO driver. This name is displayed at the user interface.

5.5.8 INF_SETUP_CLASS_GUID

Unique identifier for the device setup class in the .inf file. Create a fresh GUID using `guidgen.exe` (see section 5.3.4).

It is very important to use a fresh GUID here.

5.5.9 INF_SETUP_CLASS_NAME

Name of the device setup class for the USB devices that are supported by the driver package. This name will be shown in Windows Device Manager as a device group name.

5.5.10 INF_VID_PID_[1..16]

USB Vendor ID and Product ID of the USB devices (models) that are supported by the driver package.

Format: VID_XXXX&PID_YYYY, XXXX specifies the VID in hex format, YYYY specifies the PID in hex format.

5.5.11 INF_VID_PID_[1..16]_DESCRIPTION

Display name of the respective device model. This name is displayed in Windows Device Manager.

5.5.12 INF_VID_PID_[1..16]_DESCRIPTION_KS

Display name of the WDM audio node of the respective device model. This name is displayed in Windows Device Manager and in Windows 7 Playback/Recording devices panel.

5.5.13 SHORTCUT_FOLDER

The path for all shortcuts that will be created (except for the autostart shortcuts). This path is relative to Start Menu\Programs folder of all users.

5.5.14 CPL_EXE

The name of the control panel executable (without the .exe extension). The control panel application is expected in the bin\release and bin\debug subdirectories of the DriverPackageBuilder directory. During installation of the driver package it will be placed into the root of the installation directory. Set this variable only if a control panel application is to be included in the driver package, otherwise clear it by specifying CPL_EXE=.

5.5.15 CPL_SHORTCUT_NAME

The name of the shortcut to be created for the control panel application. The shortcut is created in the folder specified by SHORTCUT_FOLDER. If no shortcut is to be created, clear this variable by specifying CPL_SHORTCUT_NAME=.

5.5.16 CPL_SHORTCUT_PARAMS

Command line parameters for the control panel application shortcut.

5.5.17 CPL_AUTOSTART_SHORTCUT_NAME

The name of the autostart shortcut to be created for the control panel application. The shortcut is created in the `Start Menu\Startup` folder of all users. If no autostart shortcut is to be created, clear this variable by specifying `CPL_AUTOSTART_SHORTCUT_NAME=`.

5.5.18 CPL_AUTOSTART_SHORTCUT_PARAMS

Command line parameters for the control panel application autostart shortcut.

5.5.19 SPY_EXE

The name of the Spy utility executable (without the `.exe` extension). The Spy application is expected in the `bin\release` and `bin\debug` subdirectories of the `DriverPackageBuilder` directory. During installation of the driver package it will be placed into the root of the installation directory. Set this variable only if the Spy application is to be included in the driver package, otherwise clear it by specifying `SPY_EXE=`.

5.5.20 COPYFILE_[1..5]

Each variable specifies the name of an additional file that is to be included in the driver package. The file is expected in the `bin\release` and `bin\debug` subdirectories of the `DriverPackageBuilder` directory. During installation of the driver package it will be placed into the root of the target directory.

5.5.21 COPYFILE_[1..5]_SHORTCUT_NAME

The name of the shortcut to be created for the corresponding additional file.

5.5.22 COPYFILE_[1..5]_SHORTCUT_PARAMS

Command line parameters in the shortcut for the corresponding additional file.

5.5.23 LICENSE_TEXT_FILE

ASCII text file that contains licensing information to be displayed during installation of the driver package. The file is expected to be in your driver package directory, i.e. where `setvars.cmd` resides.

5.5.24 CUSTOM_BMP

The name of a custom bitmap file that contains the bitmap to be displayed by the driver installer. The bitmap dimension is fixed and must be 164x314 pixels. The file is expected to be in your driver package directory, i.e. where `setvars.cmd` resides. If this variable is not set (`CUSTOM_BMP=`) then a default bitmap will be displayed.

5.5.25 API_DLL

The name of the driver API function library without the .dll extension. If this parameter is not set, no API DLL is included in the driver package. Note that the API DLL is required by the control panel.

6 Control Panel Customization

6.1 Overview

The control panel allows a licensee to customize its appearance and behavior via .xml configuration file. The name of this file must be equal to the name of the control panel executable, except for the extension .xml. For example, the configuration file of a control panel named 'MyCpl.exe' must be named 'MyCpl.xml'. The .xml file must be located in the same folder as the control panel executable.

Customization includes:

- Definition of the driver interface to be used,
- Modification of text strings shown in the user interface,
- Multi-language support,
- Support of the system's taskbar notification area and
- Definition of the visibility of individual dialog pages.

The XML element hierarchy correlates to the control panel's visual tree. Most elements are self-explanatory. The attributes `FormatVersionMajor` and `FormatVersionMinor` of the `<ControlPanel>` element specify the XML format version. The major format version number will be incremented if changes are made to the XML format, that are incompatible with older versions of the control panel. The minor format version number will be incremented if changes are made to the XML format, that are still compatible with older versions of the control panel, e.g. additional items, that will simply be ignored by existing control panel versions.

Note: The XML configuration file must use **UTF-8** encoding! If you are using Windows Notepad for editing, choose Encoding = "UTF-8" in the "File - Save As" dialog.

6.2 Driver Interface

The configuration file has to define the driver interface to be used by the application. This is done through the XML element `<ControlPanel><DriverInterfaceGUID>` which must contain the driver interface GUID. The GUID must be equal to the driver interface GUID used to customize the driver (see section 5.5.5).

6.3 User Interface Text Strings

Apart from the multi-language support, text strings displayed by the control panel's user interface can be customized. Usually this is not required. However, it may be desired to adapt them for a specific product.

Note: The user interface was designed to allow text strings of variable length. Nevertheless, the variability is limited. So you should always check the results.

6.4 Multi-language Support

The control panel can be configured to support multiple languages. Each displayed text has a default string provided by the corresponding `<Default>` element in the configuration file. By means of optional elements of type `<Locale LCID="0x...">` the same text can be provided in additional languages. The attribute `LCID` specifies the local identifier of the language, e.g. `LCID="0x0407"` for German (Germany). Please refer to the Microsoft Developer Network (MSDN) for specific identifier values, e.g. the "National Language Support (NLS) API Reference" at <http://msdn.microsoft.com/en-us/global/bb896001.aspx>.

You can support multiple languages by adding several elements of type `<Locale LCID="0x...">` to each text element, one for each language. The application chooses the language to display by asking the system for the locale identifier that corresponds to the UI language for the current user. Then it checks separately for each string in the `.xml` file whether it is available for this locale identifier (LCID). If it is not present it looks for a default string. If there is no default string a hard-coded internal string is used instead.

Note again, that the user interface was designed to allow text strings of variable length. Nevertheless, the variability is limited. So you should always check the results.

6.5 Taskbar Notification Area Support

The control panel can be configured to display an icon in the notification area of the Windows task bar. This gives the user fast access to the application. To enable this feature, set the value of the element `<ControlPanel><NotificationArea><Enable>` to `True`, otherwise set it to `False`. If the icon is enabled, the application is minimized to the tray whenever it is closed, i.e. the application runs in the background. A left click on the icon restores the control panel. A right click on the icon opens a menu whose text strings can also be adapted via `.xml` file.

6.6 User Interface Pages

The control panel is divided into pages. Apart from the Status page which is always visible additional pages are only displayed if needed, depending on the features of the currently connected device. The configuration file allows to overwrite this default behavior for some pages by means of the element `<Visibility>`. Allowed values are `Hidden`, `Visible` and `Auto`, for some pages only `Hidden` and `Visible` apply (see comments in the configuration file itself).

Hidden

The page is never shown, regardless of the features of the connected device.

Visible

The page is always shown, regardless from the features of the connected device. The elements of the page are disabled if the corresponding device features are not available.

Auto

The page is shown if the current device supports the corresponding features, otherwise the page is hidden. This is the default behavior of the application.

7 DFU Wizard Customization

7.1 Overview

The DFU wizard allows a licensee to customize its appearance and behavior via .xml configuration file. The name of this file must be equal to the name of the wizard executable, except for the extension .xml. For example, the configuration file of an executable named 'MyDfuWizard.exe' must be named 'MyDfuWizard.xml'. The .xml file must be located in the same folder as the application executable.

Customization includes:

- Definition of the driver interface to be used,
- Definition of supported devices and corresponding firmware files,
- Firmware version check,
- Modification of text strings shown in the user interface and
- Multi-language support.

The XML element hierarchy correlates to the DFU wizards's visual tree. Most elements are self-explanatory. The attributes `FormatVersionMajor` and `FormatVersionMinor` of the `<DFU>` element specify the XML format version. The major format version number will be incremented if changes are made to the XML format, that are incompatible with older versions of the DFU wizard. The minor format version number will be incremented if changes are made to the XML format, that are still compatible with older versions of the DFU wizard, e.g. additional items, that will simply be ignored by existing wizard versions.

Note: The XML configuration file must use **UTF-8** encoding! If you are using Windows Notepad for editing, choose Encoding = "UTF-8" in the "File - Save As" dialog.

7.2 Driver Interface

The configuration file has to define the driver interface to be used by the application. This is done through the XML element `<DFU><DriverInterfaceGUID>` which must contain the driver interface GUID. The GUID must be equal to the driver interface GUID used to customize the driver (see also section 5.5.5).

7.3 Supported Devices and Corresponding Firmware Files

By default, the DFU wizard allows upgrading the firmware of all devices supported by the underlying driver. The driver is identified by a GUID, see section 7.2). The user can browse for a firmware file to be loaded onto the connected device. However, this feature should be used by advanced users only. There is always a risk that the user selects and loads the wrong file which can damage the device. Therefore, it is recommended that the DFU wizard is delivered to end users with a specification of all supported devices and their corresponding firmware files. In this case, the user is not required to select a file (and is not able to do so). To do the firmware upgrade, the user just needs to click the `Start` button.

Supported devices are specified by means of the element `<DFU><SupportedDevices>` in the XML file. Here is an example:

```
<SupportedDevices>
  <Device VID="0x1234" PID="0xABCD">
    <Firmware MajorVersion="2" MinorVersion="11">FirmwareForPidABCD.bin</Firmware>
  </Device>
  <Device VID="0x1234" PID="0x0815" Revision="0x0200">
    <Firmware MajorVersion="5" MinorVersion="11">FirmwareForPid0815v2.bin</Firmware>
  </Device>
</SupportedDevices>
```

The example lists two supported devices. Each device is identified by its VID and PID. Optionally, the revision number (bcdDevice field in the USB device descriptor) can be specified. VID, PID and revision number have to be provided in hexadecimal format (0x....). Device models not listed in the `<DFU><SupportedDevices>` section are not supported by the DFU application, even if these models are supported by the driver.

There should be no two `<DFU><SupportedDevices><Device>` elements with the same VID/PID and revision. In this case only the first element will be used. However, it is possible to add two elements for one model, one element with VID/PID/revision and another one with VID/PID only. In that case the application uses the most specific element, i.e. it first looks for an element that has the same VID/PID/revision as the connected device and if that is not available it looks for an element with matching VID/PID but no revision information. This way it is possible to download different firmware files depending on the device revision.

Each `<DFU><SupportedDevices><Device>` element provides information about an associated firmware file which is specified by its file name without a path specification. The firmware file is expected in the same folder as the DFU wizard executable. For each firmware file version information should be provided. Major and minor version numbers have to be specified in decimal format as shown in the above example.

7.4 Firmware Version Check

If supported devices are listed (see section 7.3) the DFU wizard knows the version of the corresponding firmware file. In this case it is able to compare the file version with the version of the firmware currently running in the device. The element `<DFU><CheckFirmwareVersion>` specifies the behavior of the DFU wizard with respect to firmware version checks. The following values are supported:

No No version check is performed. The firmware is always written to the device. This is the default behavior.

Warn If the version of the firmware file is equal to or older than the version running in the device, the user will be asked for confirmation before the download is started.

Block If the version of the firmware file is equal to or older than the version running in the device, the application will not perform a download. The user gets an appropriate error message.

Note: The value of `<DFU><CheckFirmwareVersion>` is ignored if the configuration file does not specify any supported devices in the `<DFU><SupportedDevices>` section.

7.5 User Interface Text Strings

Apart from the multi-language support, text strings displayed by the DFU wizard's user interface can be customized. Usually this is not required. However, it may be desired to adapt them for a specific product.

Note: The user interface was designed to allow text strings of variable length. Nevertheless, the variability is limited. So you should always check the results.

7.6 Multi-language Support

The DFU wizard can be configured to support multiple languages. Each displayed text has a default string provided by the corresponding `<Default>` element in the configuration file. By means of optional elements of type `<Locale LCID="0x...">` the same text can be provided in additional languages. The attribute `LCID` specifies the local identifier of the language, e.g. `LCID="0x0407"` for German (Germany). Please refer to the Microsoft Developer Network (MSDN) for specific identifier values, e.g. the "National Language Support (NLS) API Reference" at <http://msdn.microsoft.com/en-us/global/bb896001.aspx>.

You can support multiple languages by adding several elements of type `<Locale LCID="0x...">` to each text element, one for each language. The application chooses the language to display by asking the system for the locale identifier that corresponds to the UI language for the current user. Then it checks separately for each string in the .xml file whether it is available for this locale identifier (LCID). If it is not present it looks for a default string. If there is no default string a hard-coded internal string is used instead.

Note again, that the user interface was designed to allow text strings of variable length. Nevertheless, the variability is limited. So you should always check the results.

8 Driver Installation

For convenient and reliable installation of the TUSBAudio driver, a driver setup wizard is provided. This wizard should be customized to meet the specific requirements of the licensee. Please refer to section 5 for a detailed description of the customization procedure.

Important: It is not allowed to ship the driver setup wizard to end users without customization!

It is recommended to use the driver setup wizard to install only the driver files. For delivery, a product installer should be created that installs the (customized) driver setup wizard to disk and then runs setup.exe.

This way a vendor can

- create a product installer by using a preferred installation framework
- include more files that are required to be installed additionally
- adapt strings, logos, etc.

The driver setup wizard supports the silent (non-interactive) mode that is most qualified for this case. Please refer to section 8.4 for more details on running driver setup wizard in silent mode.

8.1 Driver Package Contents

The driver setup wizard consists of a set of files. All files are located in a single directory and must be delivered to the end user's system.

The following files belong to a driver setup package (<MyDriver> is a placeholder for the driver name that is selected by the licensee):

- setup.exe, setup.ini
- <MyDriver>.sys, <MyDriver>_x64.sys, <MyDriver>.inf, <MyDriver>.cat
- <MyDriver>ks.sys, <MyDriver>ks_x64.sys, <MyDriver>ks.inf, <MyDriver>ks.cat
- <MyDriver>asio.dll, <MyDriver>asio_x64.dll
- custom.ini

The following files can be included optionally:

- Driver API function library (tusbaudioapi.dll)
(see API_DLL in section 5.5)
- Control Panel application
(see CPL_EXE in section 5.5)
- Driver Licensing Information (license.txt in ASCII format)
(see LICENSE_TEXT_FILE in section 5.5)
- Other files that have to be copied to the user's system during driver installation
(see COPYFILE_[1 . . 5] in section 5.5)

8.2 Driver Package Delivery

The driver package builder scripts produce the driver installation package in two variants: uncompressed form (single files) and self-extracting .exe. If you plan to integrate the driver setup into an overall software installation then you should use the uncompressed form and ship all required files using the installer tool of your choice. For more information on this kind of installer integration, see also the next section.

If you plan to ship the driver installer standalone then you should use the self-extracting .exe. This self-extracting archive is created using the freeware utility 7-zip. Of course, you can also create your own self-extracting archive using any tool of your choice. However, you should create a self-extracting .exe and digitally sign it using your code-signing certificate. You should not use a .zip archive.

8.2.1 Why a self-extracting .exe should be used?

A .zip file cannot be digitally signed. If a .zip file is downloaded from the web and then extracted on Windows 8.1, Windows remembers that all the extracted files came from the web. If the user launches one of the extracted .exe files then Windows 8.1 shows a big red warning message. This problem can be avoided by using a self-extracting .exe which is digitally signed. Windows trusts this kind of archive because of its digital signature.

8.3 Driver Installation requires a connected device on some Windows versions

Windows assigns one of the drivers already preinstalled on the system, when a device is connected. For audio devices this is usually Microsoft's USB Audio Class Driver. The driver installer preinstalls an alternative driver provided with its driver package. When the user connects the device after installation finished, the system determines by means of a specific ranking the best matching driver and assigns it to the device. Unfortunately, on all operating systems before Windows 7 Microsoft's USB Audio Class Driver is preferred by default, if the driver provided with the installer is not WHQL-signed.

As long as the installer is running it can force the system to assign its own driver to the device. Therefore it requires a connected device on these operating systems. On Windows 7 and newer the system will prefer the driver provided with the installation by default, i.e. a device connection is not required during installation. As a consequence the user guidance differs depending on the operating systems. If the driver installer is integrated into a product installer using the silent mode (see section 8.4), the product installer has to adapt its user guidance too, at least because of the different installer exit codes (see section 8.4.2).

8.4 Driver Installation in Silent Mode

The so-called silent mode of the driver setup wizard allows to integrate driver installation into a product installer. Silent mode can be enabled and configured by using specific command line options. Please refer to section 8.4.1 for a description on the available command line parameters.

When using silent mode, the following needs to be considered:

- Silent mode only suppresses the GUI of the setup application. It does not suppress any Windows setup dialog that may appear.
- The caller is responsible for user guidance which depends on the exit code of the setup (see section "Exit codes").
- The installer is not a console mode application, even when running in silent mode. When executed on the command line the command prompt returns immediately while the setup is still running. To get the setup exit code it should be started by an application that is able to wait for termination of a specific process.

8.4.1 Command Line Parameters

Parameter	Description
/S	Run installer or uninstaller in silent mode.
/DIR="target path"	Installation directory on the target system. If not specified the default installation directory is used.
/NOPCPL	Prevent the installer from adding uninstallation support to Windows Programs control panel.

8.4.2 Exit Codes

Status Code	Description	Solution
0	The setup finished successfully.	-
100	Setup aborted: Another PnP installation process is currently running on the system.	Inform the user to close all open hardware installation wizards. If no wizard is open, the system probably performs some installation steps in the background. Just wait some time and run setup again.
101	Setup finished successfully. A restart is required to complete driver installation.	Reboot the system.
102	Setup aborted: The current operating system is not supported by the setup.	Inform the user and abort.
103	Setup aborted: Unexpected error.	Inform the user and abort. The setup creates a log file in the user's temp directory. This file may contain additional information on the error.
104	Setup aborted: The current user has no administrator privileges which are required to proceed.	Inform the user and abort.

Status Code	Description	Solution
105 (only on operating systems before Windows 7)	Setup aborted: The installation requires connection of the corresponding device but no device has been found.	Inform the user to connect the device and to turn it on. If the user uninstalled the drivers before by means of the Windows programs control panel, the device may already be connected but the system cannot detect it anyway. In this case the user has to disconnect and reconnect the device (and turn it on). Since you don't know whether or not an uninstallation was performed by the user before, you should always advise the user of this case. After the user has been informed, run setup again.
109	Setup aborted: Invalid command line parameters were specified.	Call the setup with correct command line parameters.
111	Setup aborted: The installer or uninstaller is already running.	The user has to finish the pending (un-)installation first.
112	Setup aborted: The installation directory could not be created on the destination system (e.g. because of missing permissions).	Inform the user or choose another installation directory and run setup again.
113	Setup aborted: The uninstaller file could not be extracted to the installation directory (e.g. because of missing permissions to write to the installation directory).	Inform the user or choose another installation directory and run setup again.
114	Setup aborted: Not all files could be installed. Probably, one or more files already exist in the installation directory and could not be overwritten.	Inform the user or choose another installation directory and run setup again.
115	Setup aborted: A driver could not be pre-installed on the system. The most likely reason is that the user did not accept installation of unsigned drivers.	Inform the user that he should accept installation of unsigned drivers despite the system warnings and run setup again.
116 (only on operating systems before Windows 7)	Setup aborted: A driver could not be updated on the system. The most likely reason is that the user didn't accept installation of unsigned drivers.	Inform the user that he should accept installation of unsigned drivers despite the system warnings and run setup again.

Status Code	Description	Solution
118	Setup aborted: Self-registration of some modules failed.	Inform the user and abort.
123	Setup aborted: setup.ini is not present or is corrupt.	Provide a correct setup.ini file.
125	Setup interrupted: Setup cannot continue because of some locked resources. A restart is required.	Reboot the system and run setup again.

8.5 Driver Uninstallation

During installation, the driver setup places some helper files for uninstallation to the main installation folder of the driver package (e.g. `uninstall.exe`). These helper applications are not intended to be called directly from any enclosing setup application or by the user.

Uninstallation support is always installed. If the installation fails for some reason the uninstallation can be performed to cleanup the system. Uninstallation is supported in the following ways:

1. If an installation is performed and the setup detects an existing installation on the destination system that differs from the current one, the uninstaller of the existing installation is executed before the new installation starts. I.e., it is not required to explicitly cleanup the system before an installation. This is done implicitly by the installer.
2. The installer creates an entry in the programs control panel (e.g. 'Add or Remove Programs' under Windows XP, 'Programs and Features' under Windows Vista/7). The user may run the driver uninstallation by means of the control panel. A graphical user interface is displayed in this case. You could prevent the installer from creating such an entry by using the appropriate command line parameter. This may be useful if you provide an enclosing setup and you don't want to allow the user to uninstall the driver package separately.
3. The setup allows to perform an explicit silent uninstallation. This may be useful if the driver setup is enclosed by another setup (e.g. an application setup). The uninstallation of the enclosing setup should involve the uninstallation of the driver package. To execute such an uninstallation, call setup with the appropriate command line parameter. It is not required to use the same version of the setup application that installed the existing driver package, but it has to be a setup application of the same product. The setup application may be located anywhere on the system, but the corresponding `setup.ini` file has to be located in the same folder. The caller of the uninstallation is responsible for deletion of the setup application file, if required, because the application does not delete itself when uninstallation is finished. Always wait for the result of the uninstallation! The uninstallation may be aborted for reasons that requires some user interaction. Furthermore, any following installation will be aborted as long as the uninstallation is running.

Note: If the uninstallation is performed as described in 2) or 3) while the device is connected the device is not visible to the system anymore when uninstallation finished. For any following installation that requires that the device is plugged in the user has to unplug and replug it.

8.5.1 Command Line Parameters

Parameter	Description
/SU	Run uninstaller in silent mode.

8.5.2 Exit Codes

Status Code	Description	Solution
0	The uninstallation finished successfully. This means that the uninstallation is done as well as possible. The uninstaller may detect that the existing installation is corrupt (e.g., because the user deleted some installed files). This may prevent the uninstaller from performing all required steps or may even prevent the whole uninstallation in the worst case. Since neither the user nor any calling application can resolve such problems, no error code is returned.	-
100	Uninstallation aborted: Another PnP installation process is currently running on the system.	Inform the user to close all open hardware installation wizards. If no wizard is open, the system is probably performing some installation steps in the background. Just wait some time and run uninstaller again.
101	Uninstallation finished successfully. A restart is required to complete driver installation.	Reboot the system.
102	Uninstallation aborted: The current operating system is not supported by the setup.	Inform the user and abort.
103	Uninstallation aborted: Unexpected error.	Inform the user and abort. The setup will create a log file in the user's temp directory. This file may contain additional information on the error.
104	Uninstallation aborted: The current user has no administrator privileges which are required to proceed.	Inform the user and abort.
109	Uninstallation aborted: Invalid command line parameters were specified.	Call the uninstaller with correct command line parameters.
111	Uninstallation aborted: The installer or uninstaller is already running.	The user has to finish the pending (un-)installation first.

Status Code	Description	Solution
123	Setup aborted: setup.ini is not present or is corrupt.	Provide a correct setup.ini file.

9 Known Issues

9.1 Intel 5/6/7 Series Chipsets: System hangs after device unplug

Chip set: Intel 5/6/7 Series Chipsets with ICH10 (e.g. P55, P65)

CPU: CPU: Intel i3/i5/i7

Operating systems: Windows 7 32/64 bit, Windows 7 32/64 bit with SP1

Problem description:

When the device is unplugged while audio is playing there is a possibility of the system hanging afterwards. Also there is a possibility of a crash (BSOD).

Possible workaround:

Do not connect the device directly to the PC. Use an external USB 2.0 hub and connect your device to this hub.

Additional information:

ICH10 has two internal EHCI compliant host controllers. Each of these controllers is connected with an internal USB hub. All USB connectors available on the PC main board are downstream ports of these hubs. So from a software perspective any USB device connected to such a PC sits behind a built-in hub. This architecture is required in order to support legacy USB 1.1 (full speed, low speed) devices. In ICH10 there is no USB 1.1 companion controller (UHCI).

Analysis has shown that the specific behavior implemented in the built-in hub causes issues in the Microsoft hub driver. In some cases the hub driver hangs when it received a disconnect event from the hub. A more detailed analysis is impossible because it is not possible to monitor the internal USB communication between EHCI and hub.

If the problem occurs, the TUSBAudio driver is not able to abort isochronous buffers pending in the bus driver. The bus driver completes the ABORT_PIPE request with STATUS_DEVICE_GONE and does not return pending buffers.

Also the bus driver does not issue an IRP_MN_SURPRISE_REMOVAL event.

9.2 XMOS USB descriptors not compatible with USB 3.0 controllers

Chip set: USB 3.0 host controllers from Renesas (NEC) and other vendors

CPU: any

Operating systems: Windows 7 32/64 bit, Windows XP

Problem description:

Older XMOS USB audio firmware uses `bInterval=8` in the endpoint descriptor of the feedback endpoint. The USB bus driver stacks provided by Microsoft and by vendors of USB 3.0 host controllers do not support `bInterval>4`. In particular, this applies to the Renesas driver which is widely used in the field.

The TUSBAudio driver fails to load if a device with this issue is connected to a USB3 host port. The system's event log contains the following error message:
"Unsupported bInterval on feedback endpoint (max 4)".

Solution:

To make the device work on USB3 host ports, the firmware needs to be changed to use `bInterval=4` in the endpoint descriptor of the feedback endpoint.

Additional information:

XMOS has fixed this issue in more recent versions of the USB audio reference designs. It has been fixed in firmware versions 3v3 for L1 and 5v3 for L2. See also firmware change log information provided by XMOS.

To check the descriptors of a given device, Thesycon's free utility TDD can be used. Download link: <http://www.thesycon.de/tdd/tdd.exe>

TUSBAudio driver versions before v1.45 had an internal workaround for the descriptor issue. This workaround has been removed in TUSBAudio v1.45 and above because it is not compatible with USB 3.0 host controllers. Consequently, TUSBAudio v1.45 and above requires a firmware version which includes the descriptor fix.

For a documentation of limitations of the Microsoft USB bus driver, check out

[http://msdn.microsoft.com/en-us/library/windows/hardware/ff539114\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff539114(v=vs.85).aspx)

9.3 Sample rates greater than 200 kHz not supported by Windows

Operating systems: Windows 7 32/64 bit, Windows XP

Problem description:

Microsoft Windows sound APIs (MME, DirectSound) do not support sample rates greater than 200 kHz. With the TUSBAudio driver higher sample rates (384 kHz, 352.8 kHz) can be used through ASIO only.

Additional information:

Article in Microsoft WDK documentation: "Policy for Mixing Audio Streams and Setting the Output Sample Rate"

[http://msdn.microsoft.com/en-us/library/windows/hardware/ff537756\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/hardware/ff537756(v=vs.85).aspx)

9.4 Installation problems on Windows XP

Operating systems: Windows XP

Problem description:

If the device's USB descriptors are designed in such a way that the Microsoft USB audio class 1.0 driver (usbaudio.sys) is installed automatically when the device is connected, this can cause a problem. Under certain conditions, the usbaudio.sys may hang and block further driver installations. It may not be possible to properly shutdown the system in this case.

Under this condition the TUSBAudio driver installer will not work correctly because the installer is not able to detect the device.

Possible workaround:

To avoid that the Microsoft USB composite driver (and USB audio class 1.0 driver) is loaded automatically, you can set the bNumConfigurations field in the USB device descriptor to 2. For a GET_DESCRIPTOR request for configuration index 1, simply return the same configuration descriptor as for configuration index 0.

9.5 Driver does not work with Fresco Logic USB 3.0 host controllers

Operating systems: Windows 7 32/64 bit, Windows XP

Problem description:

The TUSBAudio driver does not work if the audio device is connected to a Fresco Logic USB 3.0 host controller.

Additional information:

The USB host controller driver stack provided by Fresco Logic has issues. The current USB frame number reported by the bus driver is not correct. Some older versions of the driver report always zero as current frame number. Newer Fresco Logic host controller drivers (3.5.36.0) report a current frame number that is about 128 milliseconds in the future.

The internal synchronization logic in TUSBAudio relies on USB frame numbers and does not work if the current frame number is not available.

9.6 Windows does not display the 24 bit sample format for 88200 and 176400 Hz

Operating systems: Windows 7 32/64 bit, Windows 8 32/64 bit, Windows 8.1 32/64 bit

Problem description:

The Microsoft sound subsystem does not support the 24 bit PCM sample format for sample rates 88200 Hz and 176400 Hz. For these sample rates, Windows shows 16 bit formats only.

Additional information:

There is no known solution or workaround for this problem.

9.7 Multiple USB audio devices on Intel USB 3.0 host controller not working

Chip set: Intel Haswell chip set with integrated USB 3.0 host controller

Operating systems: Windows 7 32/64 bit

Problem description:

If more than one USB audio device is connected to an Intel Haswell based PC (with integrated xHCI USB 3.0 host controller) then a system crash (blue screen) can occur. The crash is caused by the Intel xHCI driver (iusb3xhc.sys). The driver does not correctly support isochronous streaming with multiple device instances in parallel.

The problem has been observed with the following driver versions:

Intel xHCI Driver version 2.5.1.28

Intel xHCI Driver version 2.5.3.34

Additional information:

There is no known solution or workaround for this problem.

9.8 Windows 7 does not display the 32 bit sample format

Operating systems: Windows 7 32/64 bit

Problem description:

The Microsoft sound subsystem does not support the 32 bit PCM sample format on Windows 7. Windows shows 16 bit and 24 bit formats only.

Additional information:

There is no known solution or workaround for this problem on Windows 7. Windows 8 shows the 32 bit PCM sample format correctly.

On any Windows version the 32-bit sample format can be used through ASIO.

10 Analyzing Problems

10.1 Monitoring Driver Statistics

The TUSBAudio driver provides a set of statistical values that are updated during driver operation. The TUSBAudio Spy Tool is part of the driver package and may be used to check these values.

10.2 Checking for Event Messages

The TUSBAudio driver writes an event message to the Windows System Event Log if it detects errors during driver and device initialization.

10.3 Capturing Driver Log Messages

The debug build variant of the TUSBAudio drivers outputs extensive log messages. To install a debug build variant of the driver, run setup.exe from the debug subdirectory of the driver package.

The trace messages are grouped into categories. Each category can be enabled separately through bit masks. The bit masks can be changed in the TUSBAudioSpy utility.

The messages are output to the Windows kernel debugger. To view and capture the messages, it is recommended that you use the DebugView utility.

10.3.1 Installing and Configuring DebugView

DebugView can be downloaded at
<http://technet.microsoft.com/en-us/sysinternals/bb896647>

After unpacking, some additional configuration steps are required for DebugView to work correctly.

- On Windows Vista and later operating system DebugView has to be run with administrator privileges. It is recommended to create a shortcut for DebugView with appropriate settings.
- Turn on the following options: 'Capture -> Capture Kernel' and 'Capture -> Enable Verbose Kernel Output'.
- Turn off the following option: 'Options -> Force Carriage Returns'

After that DebugView should show messages produced by the TUSBAudio driver when the device is connected to the system. Messages can be saved to a file and emailed to Thesycon for problem analysis.

11 XMOS Evaluation Kits

This section provides some information on how the TUSBAudio driver can be used together with Evaluation Kits from XMOS Ltd.

11.1 Device Firmware Upgrade

The driver provides an implementation for Device Firmware Upgrade (DFU) for USB devices using the USB standard DFU device class mechanism defined in [5].

Supported Functionality:

- Download a new firmware to the device
- Upload the existing firmware from the device
- Revert the device back to factory firmware image
- Automatic reboot of the device on firmware upgrade

An application accesses the DFU implementation by a set of API functions. Please refer to the TUSBAudio Reference Manual [1] for a detailed description of these API functions.

11.1.1 Preconditions

To perform a firmware upgrade the device has to conform to the following preconditions:

- the device must contain the latest DFU-enabled XMOS firmware
- the DFU-enabled firmware needs to be installed as a factory image on the device
- the new firmware needs to be available as a binary firmware image

In addition, it is required that

- the XMOS Development Tools 10.4 (or later) are used
- the TUSBAudio driver is already installed for the device.

Installing the Factory Image

To install the factory image at the device the XMOS Development tools have to be used.

Command:

```
xflash --boot-partition-size <partition size> <factory image name>.xe
```

This programs the factory default firmware image into the flash of the device and allows to install other firmware images using the DFU interface. Please refer to the XMOS Tools User Guide for details.

Generating a Firmware Upgrade Image

Before installing a new image to the device through the DFU interface it needs to be converted to binary format. The XMOS Development tools have to be used for this.

Command:

```
xflash --upgrade 1 <image name>.xe 0x20000 -o <image name>.bin
```

Please refer to the XMOS Tools User Guide for details.

11.1.2 DFU Console

The driver package includes a Windows console mode program `dfucons` that may be used to access the DFU interface. The full source code of this program is included in the TUSBAudio SDK.

The DFU console supports the following commands:

Device Information:

```
dfucons info
```

Firmware Upgrade:

```
dfucons download <firmware file> [-d<device index>]
```

Firmware Upload:

```
dfucons upload <firmware file> [-d<device index>]
```

Device Revert:

```
dfucons revertfactory [-d<device index>]
```

The device index is optional and defaults to zero. The index of each individual device instance is shown in the device information.

Note that Thesycon's sample control panel application does also implement device firmware upgrade functionality. The full source code of this sample control panel is included in the TUSBAudio SDK.

11.1.3 DFU API

The USB Audio 2.0 Class Driver for Windows provides a public DFU API that may be used to add DFU functionality to a Windows application. A reference documentation of the DFU API functions is included in the TUSBAudio SDK package.