Universal Serial Bus Device Class Definition for MIDI Devices

Release 1.0

Nov 1, 1999

Scope of this Revision

This document is the 1.0 release of this device class definition.

Contributors

Gal Ashour	IBM Corporation
Billy Brackenridge	Microsoft Corporation
Oren Tirosh	Altec Lansing
Mike Kent	Roland Corporation E-mail: mikekent@compuserve.com
Geert Knapen	Philips ITCL-USA 1000 West Maude Avenue Sunnyvale, CA 94086-2810 Phone: +1 (408) 617-4677 Fax: +1 (408) 617-7721 E-mail: gknapen@pmc.philips.com

Revision History

Revision	Date	Filename	Authors	Description
0.6	Jun. 1, 97	MIDI06.doc	Mike Kent Geert Knapen	Initial version
0.7	Sep. 1, 97	MIDI07.doc	Mike Kent Geert Knapen	Reworked the architecture
0.7a	Dec. 1, 97	MIDI07a.doc	Mike Kent Geert Knapen	Introduced multiple output Elements. Removed Appliance.
0.7b	Mar. 1, 98	MIDI07b.doc	Mike Kent Geert Knapen	Minor changes and clean-ups.
0.7c	May. 1, 98	MIDI07c.doc	Mike Kent Geert Knapen	Introduced the Event Packet structure. Removed all references to ‘F5’ message.
0.7d	Nov. 1, 98	MIDI07d.doc	Mike Kent Geert Knapen	Small editorial changes. Removed all references to time stamps.
0.8	Apr. 1, 99	MIDI08.doc	Mike Kent Geert Knapen	Minor changes. Added Management Overview. Made provisions for future extensions. Changed bit definitions for the Element.

Revision	Date	Filename	Authors	Description
0.9	Aug. 1, 99	MIDI09.doc	Mike Kent Geert Knapen	Deleted Synthesizer example. Updated a number of references. Small editorial changes.
1.0	Nov. 1, 99	MIDI10.doc	Mike Kent Geert Knapen	No Changes, except changing revision number from 0.9 to 1.0

Please send comments via electronic mail to usbhtech@mailbag.intel.com

Table of Contents

List of Tables

List of Figures

1 Introduction

Following is the USB Audio Device Class Definition for MIDI Devices. It is designed to cover the widest range of possible MIDI applications and products. This document must be considered an integral part of the USB Audio Device Class Definition.

1.1 Background

MIDI, introduced in 1983, is a mature standard with many existing products and applications. It is a standard that defines not only the data protocol for exchange of musical control information but also the hardware connection, used to physically exchange the data. Therefore, transferring MIDI data over another hardware connection like USB is not really conforming to the MIDI specification and will be called USB-MIDI from here on.

1.2 Purpose

This specification for MIDI exchange over USB is designed to be an elegant method to enable a wide range of MIDI system configurations, from the simplest to the most complex. Furthermore, this specification expands on MIDI, allowing applications not possible with non-USB MIDI configurations.

The first two figures show simple USB-MIDI systems that form the base of any MIDI related system on USB and the third example shows a very complex USB-MIDI system combining variations of the first two.

Figure 1 shows a simple USB-MIDI interface, which would allow many existing non-USB MIDI devices to connect to USB.

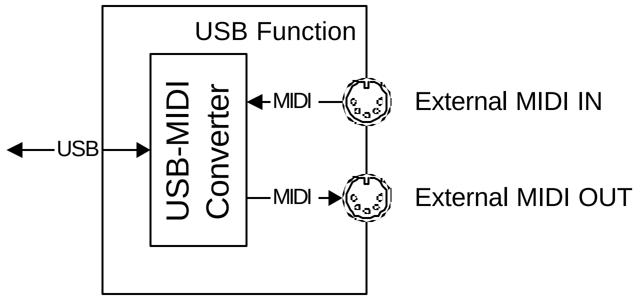

Figure 1: Simple USB-MIDI Interface

Figure 2 shows a simple USB-MIDI synthesizer receiving MIDI data via USB and supplying its output audio back to the Host for routing to USB speakers.

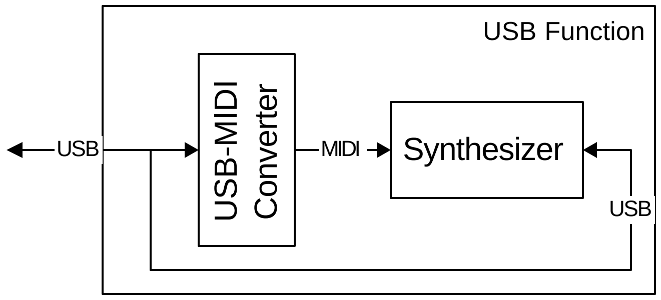

Figure 2: USB-MIDI Synthesizer

Figure 3 shows a very complex USB-MIDI system including the most challenging configurations.

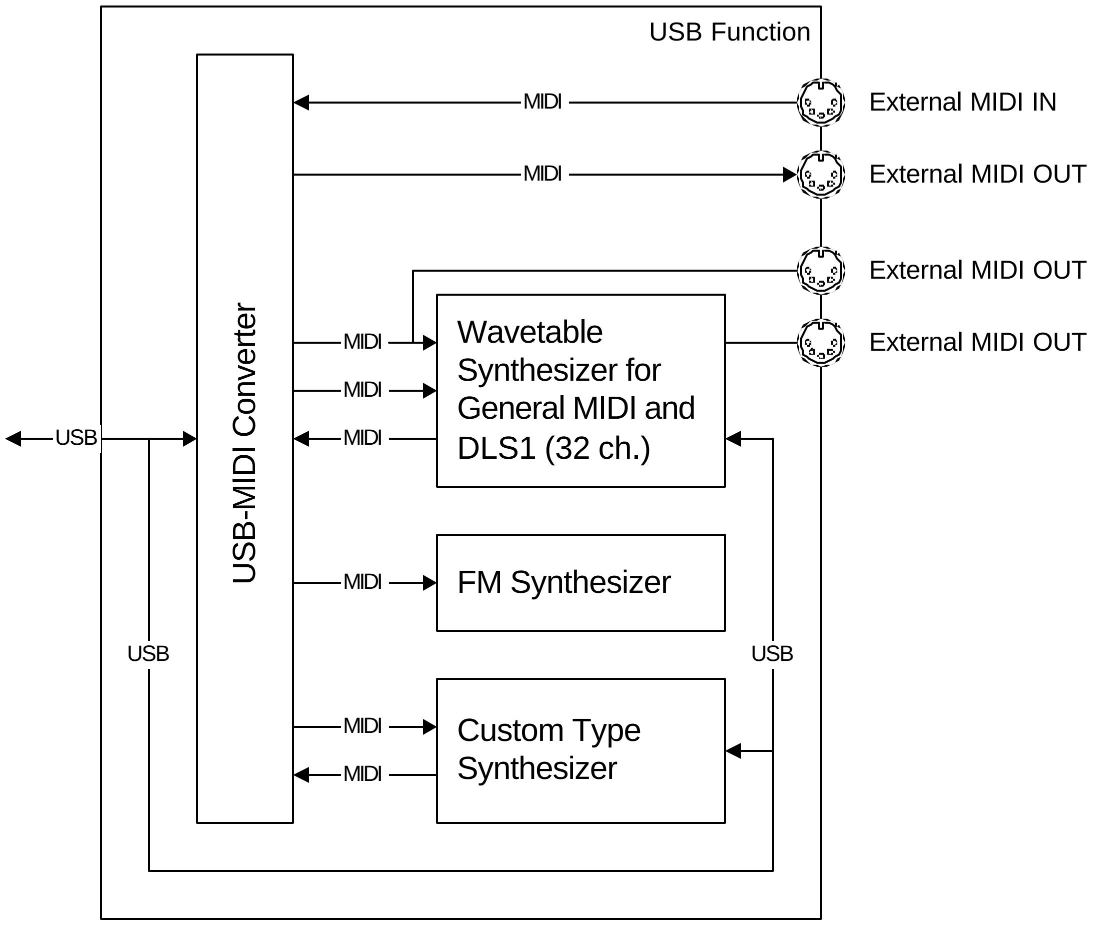

Figure 3: Complex USB-MIDI Device

1.3 Related Documents

- *Universal Serial Bus Specification*, 1.0 final draft revision (also referred to as the *USB Specification*). In particular, see Chapter 9, "USB Device Framework".
- *Universal Serial Bus Device Class Definition for Audio Devices* (referred to in this document as *USB Audio Devices*).
- *Universal Serial Bus Device Class Definition for Audio Data Formats* (referred to in this document as *USB Audio Data Formats*).
- *Universal Serial Bus Device Class Definition for Terminal Types* (referred to in this document as *USB Audio Terminal Types*).
- *Complete MIDI 1.0 Detailed Specification* as defined by the *MIDI Manufacturers Association.*
- *General MIDI System Level 1* as defined by the *MIDI Manufacturers Association.*

1.4 Terms and Abbreviations

See Section 8, "Glossary".

2 Management Overview

The USB is well suited for connecting MIDI Interfaces and MIDI instruments to computers. MIDI is a recognized protocol for music control that is serving the marketplace very well. The USB builds on the strengths of MIDI by adding higher speed of transfer and increased MIDI channels through its multiple "virtual" cable support.

In principle, a versatile bus like the USB provides many ways to handle MIDI data. For the industry, however, it is very important that MIDI transport mechanisms be well defined and standardized on the USB. Only in this way can we predict interoperability, guarantee reliable performance, and maintain the good market image of MIDI itself. Standardized MIDI transport mechanisms also help keep software drivers as generic as possible. The MIDIStreaming Interface Class described in this document satisfies those requirements. It is written and revised by experts in the MIDI field. Other device classes that address MIDI in some way should refer to this document for their MIDI specification.

The USB transfers MIDI data at rate hundreds of times faster than the original MIDI 1.0 hardware specification. In addition, this specification takes advantage of the USB's bandwidth and flexible data handling to enable the transfer of many "virtual" cables worth of MIDI data.

Multiport MIDI interfaces have become more commonplace today and they need a connection to the computer that can handle multiple MIDI connections on one cable. USB is very well suited to this task.

Synthesizers and other MIDI instruments have increased in abilities. The bandwidth of a traditional MIDI connection can be more easily consumed when trying to serve the high polyphony and increasing number of MIDI message types commonly used. Typical synthesizers now also use the 16 MIDI channels available on a MIDI bus in one instrument alone, requiring multiple MIDI busses in a typical setup with more than one MIDI instrument. Additionally, timing accuracy is essential in music.

USB can easily handle heavy loads of MIDI data while preserving the timing integrity of the data. Hundreds of MIDI note messages can be sent all at the same time. In addition, by handling multiple "virtual" cables the USB offers a solution to go beyond MIDI's 16-channel limit.

This document contains all necessary information for a designer to build a USB-compliant device that incorporates MIDI functionality. It specifies the standard and class specific descriptors that must be present in each USB MIDI Function. It further explains the transfer of MIDI events, parsed into 32 bit messages for standardized transfer over the USB and for easy handling by MIDI devices. The MIDI data itself is transferred transparently, without any changes. Furthermore, if the MIDI 1.0 specification is updated, new MIDI events or definitions are fully supported.

3 Functional Characteristics

As is the case for all audio functionality, USB-MIDI functionality resides at the interface level in the USB Device Framework hierarchy. The MIDIStreaming interface represents the entire functionality of the USB-MIDI function. It is defined as a subclass of the Audio Interface Class.

Audio functions are addressed through their audio interfaces. Each audio function has a single AudioControl interface and can have several AudioStreaming and MIDIStreaming interfaces. The AudioControl (AC) interface is used to access the audio Controls of the function whereas the AudioStreaming (AS) interfaces are used to transport audio streams into and out of the function. The MIDIStreaming (MS) interfaces are used to transport USB-MIDI data streams into and out of the audio function. The collection of the single AudioControl interface and the AudioStreaming and MIDIStreaming interfaces that belong to the same audio function is called the Audio Interface Collection (AIC). Refer to the *Universal Serial Bus Device Class Definition for Audio Devices* document for further details.

3.1 USB-MIDI Function Topology

USB-MIDI functions may contain several building blocks. All USB-MIDI functions must contain a USB-MIDI Converter, many may have some Embedded or External MIDI Jacks and some may contain one or more Elements. Elements and MIDI Jacks are generically called Entities and they are connected to each other to implement the desired MIDI functionality as shown in the following diagram (Figure 4).

Entities provide the basic building blocks to fully describe most USB-MIDI functions.

An Element is the representation of an engine that either interprets MIDI data streams and transforms them into audio streams or accepts audio streams and transforms them into MIDI data streams. Some Elements may even accept MIDI data streams and transform them into other MIDI data streams. Elements are uniquely identified by their ElementID. An Element can have one or more Input Pins and one or more Output Pins. Each Pin carries a single MIDI data stream. Furthermore, this specification provides the necessary concepts to allow asynchronous transfers of larger data sets between the Host and an Element. This can be used to implement DLS. Dedicated Transfer bulk endpoints are used for this purpose.

In addition, the concept of a MIDI Jack is introduced. There are two types of MIDI Jacks. A MIDI IN Jack is an Entity that represents a starting point for a MIDI data stream inside the USB-MIDI function. MIDI data streams enter the USB-MIDI function through a MIDI IN Jack. A MIDI OUT Jack represents an ending point for MIDI data streams. MIDI data streams leave the USB-MIDI function through a MIDI OUT Jack. From the USB-MIDI function's perspective, a USB endpoint is a typical example of a MIDI IN or MIDI OUT Jack. It either provides MIDI streams to the USB-MIDI function or consumes MIDI streams coming from the USB-MIDI function. Such MIDI Jacks, representing a USB endpoint are called Embedded MIDI Jacks. Likewise, all the physical MIDI connections, built into a USB-MIDI function are represented by External MIDI Jack Entities. Connection to a MIDI IN Jack is made through its single Output Pin. A MIDI OUT Jack can have multiple Input Pins. The MIDI OUT Jack will merge the MIDI data streams, received over its Input Pins, effectively transforming them into a single MIDI data stream. MIDI Jacks are uniquely identified by their JackID.

Entities are wired together by connecting their I/O Pins according to the required topology.

Input Pins of an Entity are numbered starting from one up to the total number of Input Pins on the Entity. Likewise, Output Pins are numbered starting from one up to the total number of Output Pins on the Entity.

Every Entity in the USB-MIDI function is fully described by its associated Entity Descriptor. The Entity Descriptor contains all necessary fields to identify and describe the Entity.

Each Entity within the USB-MIDI function is assigned a unique identification number, the EntityID, contained in the **bJackID** or **bElementID** field of the descriptor. The value 0x00 is reserved for undefined ID, effectively restricting the total number of addressable Entities in the USB-MIDI function (both Jacks and Elements) to 255.

Besides uniquely identifying all addressable Entities in an USB-MIDI function, the IDs also serve to describe the topology of the function; i.e. the **baSourceID()** array of a Jack or Element descriptor indicates to which other Entities this Entity's Input Pins are connected. In addition, the **baSourcePin()** array of a Jack or Element descriptor further qualifies the connection by indicating to which Output Pin of the other Entities this Entity's Input Pins are connected.

The ensemble of Element and MIDI Jack descriptors provide a full description of the USB-MIDI function to the Host. A generic MIDI driver should be able to fully control the USB-MIDI function.

The descriptors are further detailed in Section 6, "Descriptors" of this document.

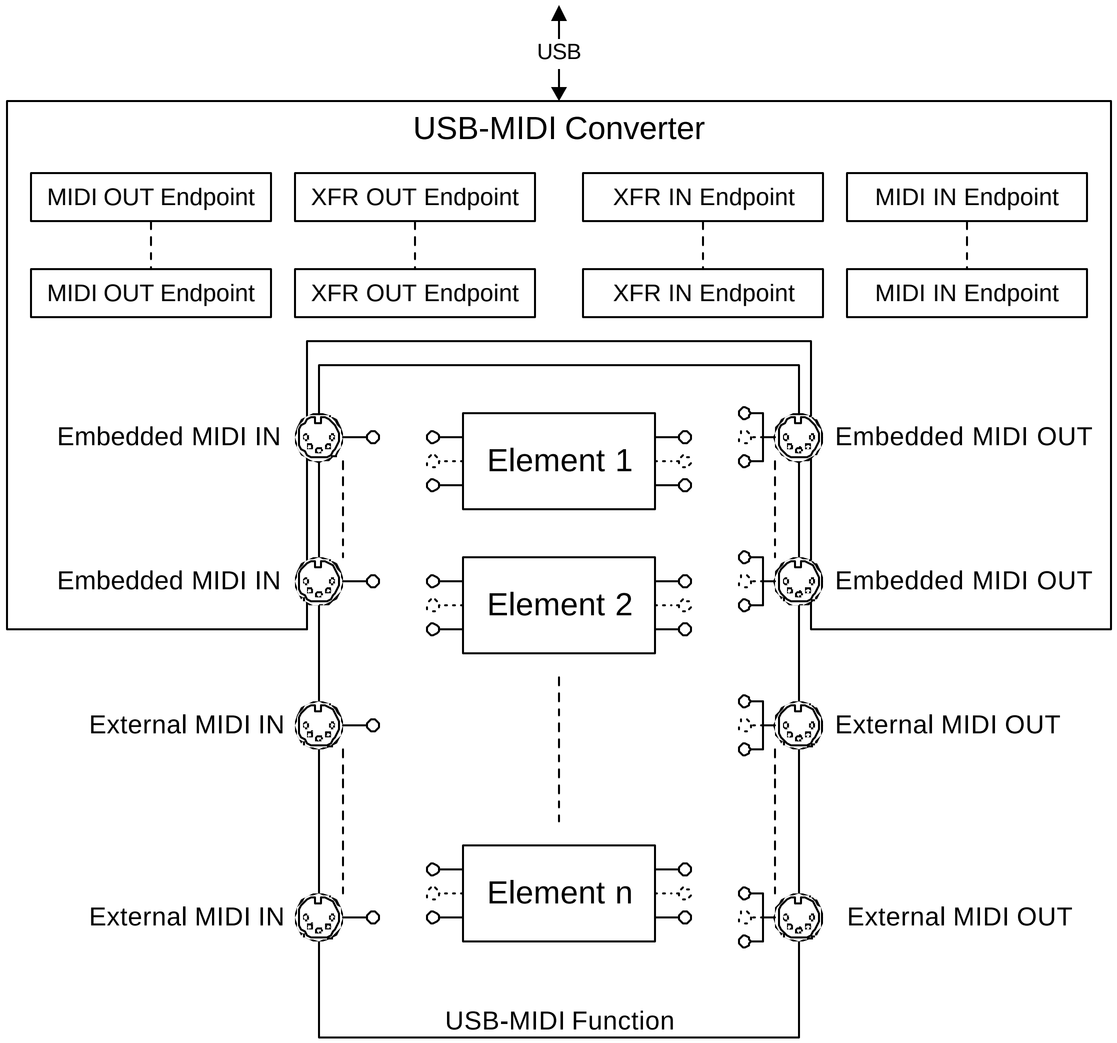

Figure 4: USB-MIDI Function Topology

3.2 USB-MIDI Converter

The USB-MIDI Converter is the heart of every USB-MIDI function since it provides the link between the Host and the USB-MIDI function. Therefore, it is a mandatory building block. On one side, it interfaces with the USB pipes that are used to exchange MIDI data streams between the Host and the USB-MIDI function's MIDI endpoints. On the other side, it presents a number of Embedded MIDI Jacks. The Embedded MIDI Jack is a logical concept to represent true MIDI connectivity within the USB-MIDI function. The USB-MIDI Converter provides the link between a MIDI OUT endpoint and the associated Embedded MIDI IN Jack. Likewise, it converts between an Embedded MIDI OUT Jack and the corresponding MIDI IN endpoint.

3.2.1 MIDI Endpoints and Embedded MIDI Jacks

The USB-MIDI Converter typically contains one or more MIDI IN and/or MIDI OUT endpoints. These endpoints use bulk transfers to exchange data with the Host. Consequently, a large quantity of USB-MIDI data can simultaneously be sent by an application without missing any MIDI events. Therefore, music applications can perform complex MIDI operations, including sending many MIDI Note On messages at the same time to more smoothly play the most complex music.

The information flowing from the Host to a MIDI OUT endpoint is routed to the USB-MIDI function through one or more Embedded MIDI IN Jacks, associated with that endpoint. Information going to the Host leaves the USB-MIDI function through one or more Embedded MIDI OUT Jacks and flows through the MIDI IN endpoint to which the Embedded MIDI Out Jacks are associated.

USB-MIDI converters can connect to multiple Embedded MIDI Jacks. Each MIDI Endpoint in a USB-MIDI converter can be connected to up to 16 Embedded MIDI Jacks. Each Embedded MIDI Jack connected to one MIDI Endpoint is assigned a number from 0 to 15. MIDI Data is transferred over the USB in 32-bit USB-MIDI Event Packets, with the first 4 bits used to designate the appropriate Embedded MIDI Jack.

A 32-bit USB-MIDI Event Packet is adopted to construct multiplexed MIDI streams (MUX MIDI) that can be sent or received by each MIDI Endpoint. At the sending end, multiple individual MIDI streams are placed into constant sized packets (with cable number) and are interleaved into a single MUX MIDI stream. At the receiving end, the multiplexed stream is properly demultiplexed, the data is extracted from the 32-bit USB-MIDI Event Packets, and each original MIDI stream is routed to the indicated virtual MIDI port. In this way, one endpoint can have multiple Embedded MIDI Jacks logically assigned. This method makes economical use of few endpoints but requires a mux/demux process on both ends of the pipe. For more information, see Section 4, “USB-MIDI Event Packets”.

3.2.2 Transfer Endpoints

The USB-MIDI Converter can contains one or more XFR IN and/or XFR OUT endpoints. These endpoints use bulk transfers to exchange data sets between the Host and any of the Elements within the USB-MIDI function. A mechanism of dynamic association is used to link a Transfer endpoint to an Element whenever that Element needs out-of-band data sets exchanged with the Host. A typical application for this type of endpoint is the transfer of DownLoadable Samples to a Synthesizer Element. The same technique could be used to download program code to an Element that contains a programmable DSP core.

3.3 External MIDI Jack

External MIDI Jacks represent physical MIDI Jacks, as defined by the MIDI Specification. They are the physical connections used by the USB-MIDI function to interface with external MIDI-compliant devices. Inside the USB-MIDI function, they are connected to Embedded MIDI Jacks, to other External MIDI Jacks or to Elements. The use of External MIDI Jacks is optional.

3.3.1 PARALLEL OUT

If multiple External MIDI OUT Jacks are linked to the same Element’s or MIDI IN Jack’s Output Pin (Embedded or External), these MIDI OUT Jacks output the same raw MIDI message streams, effectively implementing MIDI PARALLEL OUT.

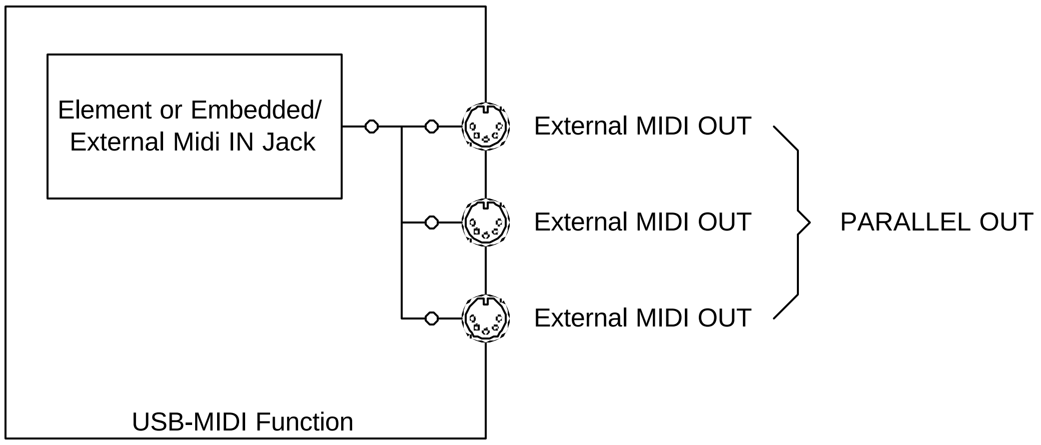

Figure 5: PARALLEL OUT

3.3.2 MIDI Through

If an External MIDI OUT Jack is linked to an External MIDI IN Jack, the raw MIDI message stream which is input from the External MIDI IN Jack is duplicated on the External MIDI OUT Jack. This effectively implements MIDI THROUGH.

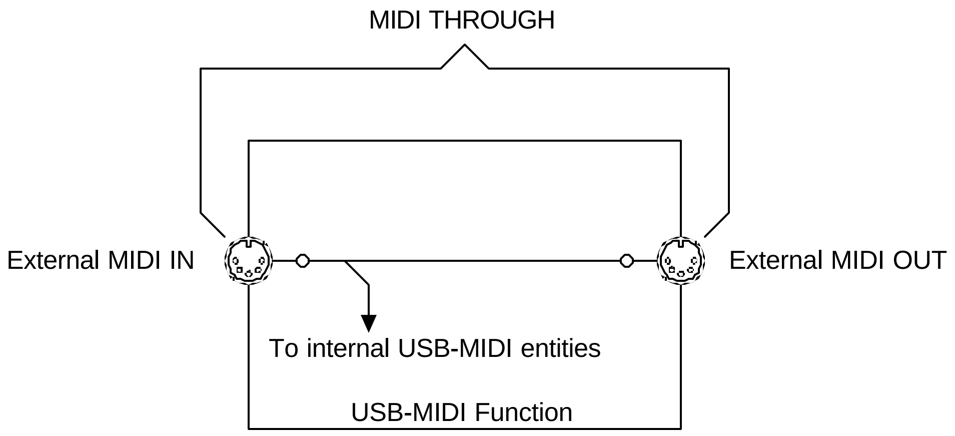

Figure 6: MIDI THROUGH

3.4 Element

A USB-MIDI function contains one or more processing engines, called Elements. Typical Elements may include:

- Synthesizer engines
- External Time Code to MIDI Time Code (MTC) converters
- MIDI controlled audio effects processors
- other MIDI controlled engines

The Element is typically connected to one or more of the Embedded MIDI Jacks or External MIDI Jacks. Commonly, when an Element is a Synthesizer, it creates an audio output stream, based on the MIDI data received from the USB-MIDI converter. This audio output is then brought into the audio function through an Input Terminal, representing the Synthesizer.

An Element may also want to send raw MIDI message streams to the Host. A typical case is the ‘bulk dump’ of settings or status from an Element (such as a Synthesizer) within a USB-MIDI function. Another examp le could be MTC messages, converted from External Time Code such as LTC (SMPTE) signals, coming from an external SMPTE Jack on the USB Device. The use of a USB-MIDI Element is optional.

3.4.1 Element Capability

There are numerous different types of Elements with MIDI capabilities, which may be reported to the Host. Some specifications for Synthesizer capabilities are defined. These capabilities may be defined by an industry standard (such as General MIDI or DownLoadableSounds) or a de-facto standard (such as GS or XG). The capabilities may be vendor-specific. Some other types of Elements may have other defined capabilities including support for MMC (MIDI Machine Control) or Sync Features (such as SMPTE-MTC conversion). Any one Element may have one or several of these capabilities.

Note: See Section 8, “Glossary” for more information about some of these de-facto standard or industry standard specifications.

3.4.2 Link to the Audio Function

Elements that either create or consume audio streams must be represented in the audio function through Input or Output Terminals respectively. The audio streams can then further be processed using all the facilities, provided by the Audio Device Class. By using the class-specific AudioControl Requests as defined by the USB Audio Device Class, Host applications can control a Synthesizer's or Audio Effect Processor’s output/input levels, equalizer settings, mixing levels with other audio streams, etc. Refer to the USB Audio Device Class documentation for more information about implementing these non-MIDI, audio features. Figure 7 illustrates the relationship between a USB-MIDI Element and its associated Input Terminal.

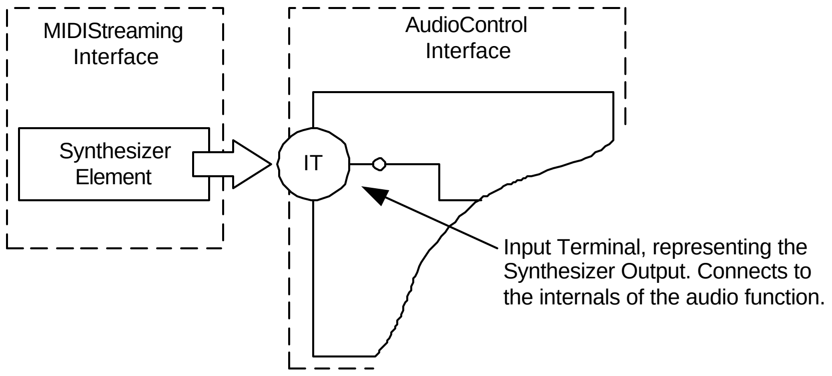
MIDIStreaming Interface
Synthesizer Element
AudioControl Interface
IT
Input Terminal, representing the Synthesizer Output. Connects to the internals of the audio function.

Figure 7: Synthesizer – Input Terminal Relationship

4 USB-MIDI Event Packets

MIDI data is transferred over USB using 32-bit USB-MIDI Event Packets. These packets provide an efficient method to transfer multiple MIDI streams with fixed length messages. The 32-bit USB-MIDI Event Packet allows multiple "virtual MIDI cables" routed over the same USB endpoint. This approach minimizes the number of required endpoints. It also makes parsing MIDI events easier by packetizing the separate bytes of a MIDI event into one parsed USB-MIDI event.

The first byte in each 32-bit USB-MIDI Event Packet is a Packet Header contains a Cable Number (4 bits) followed by a Code Index Number (4 bits). The remaining three bytes contain the actual MIDI event. Most typical parsed MIDI events are two or three bytes in length. Unused bytes must be padded with zeros (in the case of a one- or two-byte MIDI event) to preserve the 32-bit fixed length of the USB-MIDI Event Packet. They are reserved for future use. Figure 8 illustrates the layout of the packet.

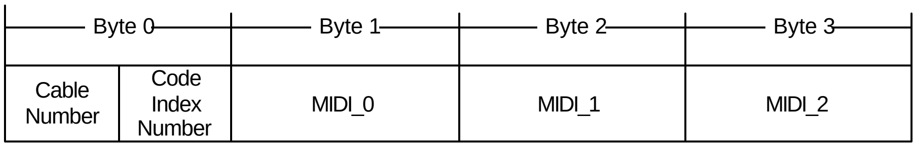

Figure 8: 32-bit USB-MIDI Event Packet

The Cable Number (CN) is a value ranging from 0x0 to 0xF indicating the number assignment of the Embedded MIDI Jack associated with the endpoint that is transferring the data. The Code Index Number (CIN) indicates the classification of the bytes in the MIDI_x fields. The following table summarizes these classifications.

Table 4-1: Code Index Number Classifications

CIN	MIDI_x Size	Description
0x0	1, 2 or 3	Miscellaneous function codes. Reserved for future extensions.
0x1	1, 2 or 3	Cable events. Reserved for future expansion.
0x2	2	Two-byte System Common messages like MTC, SongSelect, etc.
0x3	3	Three-byte System Common messages like SPP, etc.
0x4	3	SysEx starts or continues
0x5	1	Single-byte System Common Message or SysEx ends with following single byte.
0x6	2	SysEx ends with following two bytes.
0x7	3	SysEx ends with following three bytes.
0x8	3	Note-off
0x9	3	Note-on
0xA	3	Poly-KeyPress

CIN	MIDI_x Size	Description
0xB	3	Control Change
0xC	2	Program Change
0xD	2	Channel Pressure
0xE	3	PitchBend Change
0xF	1	Single Byte

Note1: F4 and F5 messages are undefined by the MIDI Specification 1.0. If they are defined in the future, the length should be two or three bytes. In this case they would be categorized as CIN=0x2 or CIN=0x3.

Note2: CIN=0xF, Single Byte: in some special cases, an application may prefer not to use parsed MIDI events. Using CIN=0xF, a MIDI data stream may be transferred by placing each individual byte in one 32 Bit USB-MIDI Event Packet. This way, any MIDI data may be transferred without being parsed.

The following table presents a number of examples on the use of the 32-bit USB-MIDI Event Packets and their relationship with the original MIDI 1.0 event.

Table 4-2: Examples of Parsed MIDI Events in 32-bit USB-MIDI Event Packets

Description	MIDI_ver. 1.0	Event Packet
Note-on message on virtual cable 1 (CN=0x1; CIN=0x9)	9n kk vv	19 9n kk vv
Control change message on cable 10 (CN=0xA; CIN=0xB)	Bn pp vv	AB Bn pp vv
Real-time message F8 on cable 3 (CN=0x3; CIN=0xF)	F8 xx xx	3F F8 xx xx
SysEx message on cable p (CN=0xp). Start of SysEx: CIN=0x4. End of SysEx: CIN=0x5	F0 00 01 F7	p4 F0 00 01 p5 F7 00 00
SysEx message on cable p (CN=0xp). Start of SysEx: CIN=0x4. End of SysEx: CIN=0x6	F0 00 01 02 F7	p4 F0 00 01 p6 02 F7 00
SysEx message on cable p (CN=0xp). Start of SysEx: CIN=0x4. End of SysEx: CIN=0x7	F0 00 01 02 03 F7	p4 F0 00 01 p7 02 03 F7
Special case: two-byte SysEx on cable p (CN=0xp; CIN=0x6)	F0 F7	p6 F0 F7 00
Special case: three-byte SysEx on cable p (CN=0xp; CIN=0x7)	F0 mm F7	p7 F0 mm F7

5 Operational Model

The USB-MIDI function exposes a single MIDIStreaming interface that is used by Host software to interact and control the entire MIDI functionality of the function. Since all control messages are performed in-band (buried into the MIDI data stream) there is no need for a separate control and status interface. The MIDIStreaming interface contains one or more MIDI endpoints, which all use bulk transfers to exchange MIDI data with the Host. In addition, one or more Transfer bulk endpoints can be present to provide a high bandwidth path to some of the Elements inside the USB-MIDI function.

As already mentioned before, MUX MIDI data streams can be used to optimize communication between Host and USB-MIDI function. The following paragraphs describe the data paths involved when communication is taking place between multiple MIDI applications on the Host and multiple USB-MIDI Elements or external MIDI appliances, connected to the USB-MIDI function. However, it must be clearly understood that the part of the discussion involving Host behavior is merely there to complete the full picture and is not imposing any implementation rules on Host software. This specification is limited to specifying USB-MIDI function behavior only.

5.1 Communication from Host to USB-MIDI Function

Multiple applications may want to send MIDI data streams to different parts of the USB-MIDI function. For efficiency reasons, it is desirable not to have a different USB pipe for every communication path to be established although this specification does not prevent implementations that desire to do so. In general, multiple MIDI data streams, possibly originating from different applications will be multiplexed into a single MUX MIDI data stream by some MIDI Mixer entity in Host software. The format of the MUX MIDI data stream is based upon the use of 32 Bit USB-MIDI Event Packets with cable numbers and is further detailed in Section 4, “USB-MIDI Event Packets”. This MUX MIDI data stream is sent over a USB pipe to a MIDI OUT Endpoint, residing in the USB-MIDI Converter. The USB-MIDI Converter inspects the incoming MUX-MIDI USB-MIDI Data Packets. It extracts the MIDI data from the packets and routes the original MIDI data streams to the proper Embedded MIDI IN jacks, according to the cable number embedded in the USB-MIDI Data Packets. There is a one to one relationship between cable numbers and JackIDs of the Embedded MIDI IN Jacks, associated with the MIDI OUT endpoint. Once present at the Embedded MIDI IN Jacks, the MIDI data streams follow the routing as dictated by the implemented topology.

5.2 Communication from USB-MIDI Function to Host

Multiple Entities in the USB-MIDI function may want to send MIDI data streams to different applications in the Host. For efficiency reasons, it is desirable not to have a different USB pipe for every communication path to be established although this specification does not prevent implementations that desire to do so. In general, multiple MIDI data streams, originating from different Entities within the USB-MIDI function arrive at the different Embedded MIDI OUT Jacks, associated with a MIDI IN endpoint in the USB-MIDI Converter. The USB-MIDI Converter converts each MIDI data stream into 32 Bit USB-MIDI Event Packets with the appropriate cable number message and multiplexes all the MIDI data streams into a single MUX MIDI data stream. This MUX MIDI data stream is sent over a USB pipe to the Host. The Host software then inspects the incoming MUX-MIDI USB-MIDI Data Packets, extracts the MIDI data from the packets and routes the original MIDI data streams to the proper applications, according to the cable number embedded in the USB-MIDI Data Packets.

5.2.1 High Bandwidth Data Transfer Mechanism

To efficiently implement features like DownLoadable Samples, there is a need for a high bandwidth data transfer mechanism between the Host and one or more Elements within the USB-MIDI function. The most straightforward way to implement such a facility would be to associate a separate bulk endpoint to each Element that requires this high bandwidth path. However, a more conservative approach has been chosen where a Transfer bulk endpoint can be dynamically allocated to any Element that needs the facility at a certain moment in time. Typically, a USB-MIDI function would have one Transfer bulk IN and one Transfer bulk OUT endpoint to serve all the Elements inside the USB-MIDI function. However, if necessary, multiple Transfer bulk IN and OUT endpoints may be implemented to increase the available bandwidth of the transfer mechanism. A class-specific Request is provided to control the allocation of a Transfer bulk endpoint (See Section 7.2.2.2.3.1, “Association Control”). This specification makes no assumptions as to which data formats are used over the Transfer bulk endpoint. The Transfer mechanism merely provides a high bandwidth data path between the Host and the currently associated Entity.

6 Descriptors

The MIDIStreaming (MS) interface and endpoint descriptors contain all relevant information to fully characterize the corresponding USB-MIDI function. The standard interface descriptor characterizes the interface itself, whereas the class-specific interface descriptor provides pertinent information concerning the internals of the USB-MIDI function. It specifies revision level information and lists the capabilities of each Jack and Element.

6.1 MIDIStreaming Interface Descriptors

6.1.1 Standard MS Interface Descriptor

The standard MIDIStreaming (MS) interface descriptor is identical to the standard interface descriptor defined in Section 9.6.3 of the *USB Specification*, except that some fields have now dedicated values.

Table 6-1: Standard MIDIStreaming Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor, in bytes: 9
1	bDescriptorType	1	Constant	INTERFACE descriptor type
2	bInterfaceNumber	1	Number	Number of interface. A zero-based value identifying the index in the array of concurrent interfaces supported by this configuration.
3	bAlternateSetting	1	Number	Value used to select an alternate setting for the interface identified in the prior field.
4	bNumEndpoints	1	Number	Number of MIDI endpoints used by this interface (excluding endpoint 0).
5	bInterfaceClass	1	Class	AUDIO. Audio Interface Class code (assigned by the USB). See Appendices in the *Universal Serial Bus Device Class Definition for Audio Devices* document.
6	bInterfaceSubClass	1	Subclass	MIDISTREAMING. Audio Interface Subclass code. Assigned by this specification. See Appendices in the *Universal Serial Bus Device Class Definition for Audio Devices* document.
7	bInterfaceProtocol	1	Protocol	Not used. Must be set to 0.
8	iInterface	1	Index	Index of a string descriptor that describes this interface.

6.1.2 Class-Specific MS Interface Descriptor

The class-specific MS interface descriptor is a concatenation of all the descriptors that are used to fully describe the USB-MIDI function, i.e. all MIDI IN and MIDI OUT Jack Descriptors (JDs) and Element Descriptors (EDs).

6.1.2.1 Class-Specific MS Interface Header Descriptor

The total length of the class-specific MS interface descriptor depends on the number of Jacks and Elements in the USB-MIDI function. Therefore, the descriptor starts with a header that reflects the total length in bytes of the entire class-specific MS interface descriptor in the **wTotalLength** field. The **bcdMSC** field identifies the release of the MIDIStreaming SubClass Specification with which this USB-MIDI function and its descriptors are compliant. The order in which the Jack and Element descriptors are reported is not important since every descriptor can be identified through its **bDescriptorType** and **bDescriptorSubtype** field. The **bDescriptorType** field identifies the descriptor as being a class-specific interface descriptor. The **bDescriptorSubtype** field further qualifies the exact nature of the descriptor.

The following table defines the class-specific MS interface header descriptor.

Table 6-2: Class-Specific MS Interface Header Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor, in bytes: 7
1	bDescriptorType	1	Constant	CS_INTERFACE descriptor type.
2	bDescriptorSubtype	1	Constant	MS_HEADER descriptor subtype.
3	bcdMSC	2	BCD	MIDIStreaming SubClass Specification Release Number in Binary-Coded Decimal. Currently 01.00.
5	wTotalLength	2	Number	Total number of bytes returned for the class-specific MIDIStreaming interface descriptor. Includes the combined length of this descriptor header and all Jack and Element descriptors.

This header is followed by one or more Jack and/or Element Descriptors. The layout of the descriptors depends on the type of Jack or Element they represent. The first four fields are common for all Jack and Element Descriptors. They contain the Descriptor Length, Descriptor Type, Descriptor Subtype and Jack or Element ID.

Each Jack and Element within the USB-MIDI function is assigned a unique identification number, the JackID or ElementID, contained in the **bJackID** or **bElementID** field of the descriptor. The value 0x00 is reserved for undefined ID, effectively restricting the total number of addressable Entities in the USB-MIDI function (both Jacks and Elements) to 255.

6.1.2.2 MIDI IN Jack Descriptor

Table 6-3: MIDI IN Jack Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor, in bytes: 6
1	bDescriptorType	1	Constant	CS_INTERFACE descriptor type.
2	bDescriptorSubtype	1	Constant	MIDI_IN_JACK descriptor subtype.
3	bJackType	1	Constant	EMBEDDED or EXTERNAL
4	bJackID	1	Constant	Constant uniquely identifying the MIDI IN Jack within the USB-MIDI function.
5	iJack	1	Index	Index of a string descriptor, describing the MIDI IN Jack.

6.1.2.3 MIDI OUT Jack Descriptor

Table 6-4: MIDI OUT Jack Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor, in bytes: 6+2*p
1	bDescriptorType	1	Constant	CS_INTERFACE descriptor type.
2	bDescriptorSubtype	1	Constant	MIDI_OUT_JACK descriptor subtype.
3	bJackType	1	Constant	EMBEDDED or EXTERNAL
4	bJackID	1	Constant	Constant uniquely identifying the MIDI OUT Jack within the USB-MIDI function.
5	bNrInputPins	1	Number	Number of Input Pins of this MIDI OUT Jack: p
6	baSourceID(1)	1	Number	ID of the Entity to which the first Input Pin of this MIDI OUT Jack is connected.
7	BaSourcePin(1)	1	Number	Output Pin number of the Entity to which the first Input Pin of this MIDI OUT Jack is connected.
…	…	…	…	…
6+2*(p-1)	baSourceID (p)	1	Number	ID of the Entity to which the last Input Pin of this MIDI OUT Jack is connected.
6+2*(p-1)+1	BaSourcePin(p)	1	Number	Output Pin number of the Entity to which the last Input Pin of this MIDI OUT Jack is connected.

Offset	Field	Size	Value	Description
5+2*p	iJack	1	Index	Index of a string descriptor, describing the MIDI OUT Jack.

6.1.2.4 Element Descriptor

The **bInTerminalLink** field contains the unique Terminal ID of the Input Terminal to which this Element is connected. If there is no link to an Input Terminal, then the **bInTerminalLink** field must be set to zero.

The **bOutTerminalLink** field contains the unique Terminal ID of the Output Terminal to which this Element is connected, effectively implementing a Sampler, an Effector controlled by MIDI, and so on. If there is no link to an Output Terminal, then the **bOutTerminalLink** field must be set to zero.

The **bElCapsSize** field represents the size in bytes of the **bmElementCaps** field. Currently, this field is set to 1. However, this field is here to accommodate future extensions.

Each bit in the **bmElementCaps** field represents some capability of a USB-MIDI Element. When a bit is set, it indicates that the corresponding capability is supported by the Element. At least one bit should be set (bit D7 if no other applicable).

D0: CUSTOM UNDEFINED	The Element has unique, undefined features. A typical example would be a unique synthesizer type or MIDI controlled audio effects processor.
D1: MIDI CLOCK	MIDI CLOCK messages are supported. Typical example Elements include drum machines and MIDI CLOCK to FSK converters.
D2: MIDI TIME CODE (MTC)	Synchronization features are supported. Typical example Elements include MTC to SMPTE converters.
D3: MIDI MACHINE CONTROL (MMC)	MMC messages are supported.
D4: GM1	General MIDI System Level 1 compatibility as defined by the MIDI Manufacturers Association.
D5: GM2	General MIDI System Level 2 compatibility as defined by the MIDI Manufacturers Association.
D6: GS	GS Format compatibility as defined by Roland Corporation.
D7: XG	XG compatibility as defined by Yamaha Corporation.
D8: EFX	The Element provides an audio effects processor controlled by USB.
D9: MIDI Patch Bay	The Element provides an internal MIDI Patcher or Router.
D10: DLS1	DownLoadable Sounds Standard Level 1 compatibility as defined by the MIDI Manufacturers Association.
D11: DLS2	DownLoadable Sounds Standard Level 2 compatibility as defined by the MIDI Manufacturers Association.
D12 and higher	Represents future possible common defined MIDI type such as DLS3, GM3 and so on.

iElement represents the Index of a string descriptor, describing the Element.

Table 6-5: MIDI Element Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor, in bytes: 10+2*p+n
1	bDescriptorType	1	Constant	CS_INTERFACE descriptor type.
2	bDescriptorSubtype	1	Constant	ELEMENT descriptor subtype.
3	bElementID	1	Constant	Constant uniquely identifying the MIDI OUT Jack within the USB-MIDI function.
4	bNrInputPins	1	Number	Number of Input Pins of this Element: p
5	baSourceID(1)	1	Number	ID of the Entity to which the first Input Pin of this Element is connected.
6	BaSourcePin(1)	1	Number	Output Pin number of the Entity to which the first Input Pin of this Element is connected.
…	…	…	…	…
5+2*(p-1)	baSourceID (p)	1	Number	ID of the Entity to which the last Input Pin of this Element is connected.
5+2*(p-1)+1	BaSourcePin(p)	1	Number	Output Pin number of the Entity to which the last Input Pin of this Element is connected.
5+2*p	bNrOutputPins	1	Number	Number of Output Pins of this Element: q
6+2*p	bInTerminalLink	1	Constant	The Terminal ID of the Input Terminal to which this Element is connected
7+2*p	bOutTerminalLink	1	Constant	The Terminal ID of the Output Terminal to which this Element is connected
8+2*p	bElCapsSize	1	Number	Size, in bytes of the bmElementCaps field.

Offset	Field	Size	Value	Description
9+2*p	bmElementCaps	n	BitMap	D0: Custom Undefined Type D1: MIDI CLOCK D2: MTC D3: MMC D4: GM1 D5: GM2 D6: GS D7: XG D8: EFX D9: MIDI Patch Bay D10: DLS1 D11: DLS2 D12..Dx: Reserved for future common defined MIDI types
9+2*p+n	iElement	1	Index	Index of a string descriptor, describing the Element.

6.2 MIDIStreaming Endpoint Descriptors

The following paragraphs outline the descriptors that fully characterize the endpoint(s) used for transporting MIDI data streams to and from the USB-MIDI function. In addition, the descriptors of the Transfer bulk endpoint(s) are also described.

6.2.1 Standard MS Bulk Data Endpoint Descriptor

The standard MS bulk MIDI data endpoint descriptor is identical to the standard endpoint descriptor defined in Section 9.6.4, “Endpoint,” of the *USB Specification* and further expanded as defined in the *Universal Serial Bus Class Specification*. D7 of the **bEndpointAddress** field indicates whether the endpoint is a MIDI data source (D7 = 1) or a MIDI data sink (D7 = 0). The **bmAttributes** Field bits are set to reflect the bulk type of the endpoint. The synchronization type is indicated by D3..2 and must be set to None.

Table 6-6: Standard MS Bulk Data Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor, in bytes: 9
1	bDescriptorType	1	Constant	ENDPOINT descriptor type
2	bEndpointAddress	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: D7: Direction. 0 = OUT endpoint 1 = IN endpoint D6..4: Reserved, reset to zero D3..0: The endpoint number, determined by the designer.

Offset	Field	Size	Value	Description
3	bmAttributes	1	Bit Map	D7..4: Reserved D3..2: Synchronization type 00 = None D1..0: Transfer type 10 = Bulk
4	wMaxPacketSize	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.
6	bInterval	1	Number	Interval for polling endpoint for data transfers expressed in milliseconds. This field is ignored for bulk endpoints. Must be reset to 0.
7	bRefresh	1	Number	Reset to 0.
8	bSynchAddress	1	Endpoint	The address of the endpoint used to communicate synchronization information if required by this endpoint. Reset to zero.

6.2.2 Class-Specific MS Bulk Data Endpoint Descriptor

The **bNumEmbMIDIJack** field contains the number of Embedded MIDI Jacks , associated with this MS bulk data endpoint. In case of a data IN endpoint, these would be Embedded MIDI OUT Jacks. If it is a data OUT endpoint, these would be Embedded MIDI IN Jacks. The **baAssocJacks()** array contains the JackID's of these Embedded MIDI Jacks.

Table 6-7: Class-specific MS Bulk Data Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor, in bytes: 4+n
1	bDescriptorType	1	Constant	CS_ENDPOINT
2	bDescriptorSubTyp e	1	Constant	MS_GENERAL
3	bNumEmbMIDIJack	1	Number	Number of Embedded MIDI Jacks: n.
4	baAssocJackID(1)	1	Constant	ID of the first Embedded Jack that is associated with this endpoint.
...	...	...	...	...
4+(n-1)	baAssocJackID(n)	1	Constant	ID of the last Embedded Jack that is associated with this endpoint.

6.2.3 Standard MS Transfer Bulk Data Endpoint Descriptor

The standard MS Transfer bulk MIDI data endpoint descriptor is identical to the standard endpoint descriptor defined in Section 9.6.4, "Endpoint," of the *USB Specification* and further expanded as defined in the *Universal Serial Bus Class Specification*. D7 of the **bEndpointAddress** field indicates whether the endpoint is a Transfer IN (D7 = 1) or a Transfer OUT (D7 = 0) endpoint. The **bmAttributes** Field bits are set to reflect the bulk type of the endpoint. The synchronization type is indicated by D3..2 and must be set to None.

Table 6-8: Standard MS Transfer Bulk Data Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor, in bytes: 9
1	bDescriptorType	1	Constant	ENDPOINT descriptor type
2	bEndpointAddress	1	Endpoint	The address of the endpoint on the USB device described by this descriptor. The address is encoded as follows: D7: Direction. 0 = OUT endpoint 1 = IN endpoint D6..4: Reserved, reset to zero D3..0: The endpoint number, determined by the designer.
3	bmAttributes	1	Bit Map	D7..4: Reserved D3..2: Synchronization type 00 = None D1..0: Transfer type 10 = Bulk
4	wMaxPacketSize	2	Number	Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.
6	bInterval	1	Number	Interval for polling endpoint for data transfers expressed in milliseconds. This field is ignored for bulk endpoints. Must be reset to 0.
7	bRefresh	1	Number	Reset to 0.
8	bSynchAddress	1	Endpoint	The address of the endpoint used to communicate synchronization information if required by this endpoint. Reset to zero.

6.2.4 Class-Specific MS Transfer Bulk Data Endpoint Descriptor

There is no class-specific MS Transfer bulk data endpoint descriptor.

7 Requests

7.1 Standard Requests

The Audio Device Class supports the standard requests described in Section 9, "USB Device Framework," of the *USB Specification*. The MIDIStreaming SubClass places no specific requirements on the values for the standard requests.

7.2 Class-Specific Requests

Most class-specific requests are used to set and get MIDI related Controls. At this time, only the Set and Get Association Control for Transfer bulk endpoints is defined, but a more general framework for MIDIStreaming Control requests is presented for future use.

The following request types are defined:

- **MIDIStreaming Requests**. Control of the class-specific behavior of a MIDIStreaming interface is performed through manipulation of either interface Controls or endpoint Controls. These can be either standard Controls, as defined in this specification or vendor-specific. In either case, the same request layout can be used. MIDIStreaming requests are directed to the recipient where the Control resides. This can be either the interface or its associated bulk endpoints.

The MIDIStreaming SubClass supports additional class-specific request:

- **Memory Requests**. Every addressable Entity in the USB-MIDI function (Element, MIDI Jack) can expose a memory-mapped interface that provides the means to generically manipulate the Entity. Vendor-specific Control implementations could be based on this type of request.
- The **Get Status** request is a general query to an Entity in a MIDIStreaming interface and does not manipulate Controls.

In principle, all requests are optional. If a USB-MIDI function does not support a certain request, it must indicate this by stalling the control pipe when that request is issued to the function. However, if a certain Set request is supported, the associated Get request must also be supported. Get requests may be supported without the associated Set request being supported.

The rest of this section describes the class-specific requests used to manipulate both interface Controls and endpoint Controls.

7.2.1 Request Layout

The following paragraphs describe the general structure of the Set and Get requests. Subsequent paragraphs detail the use of the Set/Get requests for the different request types.

7.2.1.1 Set Request

This request is used to set an attribute of a Control inside an Entity of the USB-MIDI function. Additionally, the memory space attribute of an Entity itself can be set through this request.

Table 7-1: Set Request Values

bmRequest Type	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_CUR SET_MIN SET_MAX SET_RES SET_MEM	See following paragraphs	Entity ID and Interface	Length of parameter block	Parameter block
00100010B			Endpoint		

The **bmRequestType** field specifies that this is a SET request (D7=0b0). It is a class-specific request (D6..5=0b01), directed to either a MIDIStreaming interface of the USB-MIDI function (D4..0=0b00001) or a bulk endpoint of an MIDIStreaming interface (D4..0=0b00010).

The **bRequest** field contains a constant, identifying which attribute of the addressed Control or Entity is to be modified. Possible attributes for a Control are its:

• Current setting attribute	(SET_CUR)
• Minimum setting attribute	(SET_MIN)
• Maximum setting attribute	(SET_MAX)
• Resolution attribute	(SET_RES)

Possible attributes for an Entity are its:

• Memory space attribute	(SET_MEM)

If the addressed Control or Entity does not support modification of a certain attribute, the control pipe must indicate a stall when an attempt is made to modify that attribute. In most cases, only the CUR and MEM attribute will be supported for the Set request. However, this specification does not prevent a designer from making other attributes programmable. For the list of Request constants, refer to Section A.4, "*Class-Specific Request Codes*".

The **wValue** field interpretation is qualified by the value in the **wIndex** field. Depending on what Entity is addressed, the layout of the **wValue** field changes. The following paragraphs describe the contents of the **wValue** field for each Entity separately. In most cases, the **wValue** field contains the Control Selector (CS) in the high byte. It is used to address a particular Control within Entities that can contain multiple Controls. If the Entity only contains a single Control, there is no need to specify a Control Selector and the **wValue** field can be used to pass additional parameters.

The **wIndex** field specifies the interface or endpoint to be addressed in the low byte and the Entity ID or zero in the high byte. In case an interface is addressed, the virtual Entity 'interface' can be addressed by specifying zero in the high byte. The values in **wIndex** must be appropriate to the recipient. Only existing Entities in the USB-MIDI function can be addressed and only appropriate interface or endpoint numbers may be used. If the request specifies an unknown or non-Entity ID or an unknown interface or endpoint number, the control pipe must indicate a stall.

The actual parameter(s) for the Set request are passed in the data stage of the control transfer. The length of the parameter block is indicated in the **wLength** field of the request. The layout of the parameter block is qualified by both the **bRequest** and **wIndex** fields. Refer to the following sections for a detailed description of the parameter block layout for all possible Entities.

7.2.1.2 Get Request

This request returns the attribute setting of a specific Control inside an Entity of the USB-MIDI function. Additionally, the memory space attribute of an Entity itself can be returned through this request.

Table 7-2: Get Request Values

bmRequest Type	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_CUR GET_MIN GET_MAX GET_RES GET_MEM	See following paragraphs	Entity ID and Interface	Length of parameter block	Parameter block
10100010B			Endpoint		

The **bmRequestType** field specifies that this is a GET request (D7=0b1). It is a class-specific request (D6..5=0b01), directed to either a MIDIStreaming interface of the USB-MIDI function (D4..0=0b00001) or a bulk endpoint of a MIDIStreaming interface (D4..0=0b00010).

The **bRequest** field contains a constant, identifying which attribute of the addressed Control or Entity is to be returned. Possible attributes for a Control are its:

• Current setting attribute	(GET_CUR)
• Minimum setting attribute	(GET_MIN)
• Maximum setting attribute	(GET_MAX)
• Resolution attribute	(GET_RES)

Possible attributes for an Entity are its:

• Memory space attribute	(GET_MEM)

If the addressed Control or Entity does not support readout of a certain attribute, the control pipe must indicate a stall when an attempt is made to read that attribute. For the list of Request constants, refer to Section A.4, "*Class-Specific Request Codes*".

The **wValue** field interpretation is qualified by the value in the **wIndex** field. Depending on what Entity is addressed, the layout of the **wValue** field changes. The following paragraphs describe the contents of the **wValue** field for each Entity separately. In most cases, the **wValue** field contains the Control Selector (CS) in the high byte. It is used to address a particular Control within Entities that can contain multiple Controls. If the Entity only contains a single Control, there is no need to specify a Control Selector and the **wValue** field can be used to pass additional parameters.

The **wIndex** field specifies the interface or endpoint to be addressed in the low byte and the Entity ID or zero in the high byte. In case an interface is addressed, the virtual Entity 'interface' can be addressed by specifying zero in the high byte. The values in **wIndex** must be appropriate to the recipient. Only existing Entities in the USB-MIDI function can be addressed and only appropriate interface or endpoint numbers may be used. If the request specifies an unknown or non-Entity ID or an unknown interface or endpoint number, the control pipe must indicate a stall.

The actual parameter(s) for the Get request are returned in the data stage of the control transfer. The length of the parameter block to return is indicated in the **wLength** field of the request. If the parameter block is longer than what is indicated in the **wLength** field, only the initial bytes of the parameter block are returned. If the parameter block is shorter than what is indicated in the **wLength** field, the device indicates the end of the control transfer by sending a short packet when further data is requested. The layout of the parameter block is qualified by both the **bRequest** and **wIndex** fields. Refer to the following sections for a detailed description of the parameter block layout for all possible Entities.

7.2.2 MIDIStreaming Requests

MIDIStreaming requests can be directed either to the MIDIStreaming interface or to the associated bulk data endpoint(s), depending on the location of the Control to be manipulated.

7.2.2.1 Interface Control Requests

For now, this specification does not support interface Control Requests.

7.2.2.2 Endpoint Control Requests

The following sections describe the requests a USB-MIDI function can support for its MIDIStreaming bulk endpoints. The same layout of the parameter blocks is used for both the Set and Get requests.

7.2.2.2.1 Set Endpoint Control Request

This request is used to set an attribute of an endpoint Control inside a particular endpoint of the USB-MIDI function.

Table 7-3: Set Endpoint Control Request Values

bmRequest Type	bRequest	wValue	wIndex	wLength	Data
00100010B	SET_CUR SET_MIN SET_MAX SET_RES	CS	endpoint	Length of parameter block	Parameter block

The **bRequest** field indicates which attribute the request is manipulating. The MIN, MAX, and RES attributes are usually not supported for the Set request.

The **wValue** field specifies the Control Selector (CS) in the high byte and the low byte must be set to zero. The Control Selector indicates which type of Control this request is manipulating (Association, etc.) If the request specifies an unknown CS to that endpoint, the control pipe must indicate a stall.

For a description of the parameter block for the endpoint Control Selectors, see Section 7.2.2.2.3, "*Endpoint Controls*."

7.2.2.2.2 Get Endpoint Control Request

This request returns the attribute setting of a specific endpoint Control inside an endpoint of the USB-MIDI function.

Table 7-4: Get Endpoint Control Request Values

bmRequest Type	bRequest	wValue	wIndex	wLength	Data
10100010B	GET_CUR GET_MIN GET_MAX GET_RES	CS	endpoint	Length of parameter block	Parameter block

The **bRequest** field indicates which attribute the request is reading.

The **wValue** field specifies the Control Selector (CS) in the high byte and the low byte must be set to zero. The Control Selector indicates which type of Control this request is manipulating (Association, etc.) If the request specifies an unknown CS to that endpoint, the control pipe must indicate a stall.

For a description of the parameter block for the endpoint Control Selectors, see Section 7.2.2.2.3, "Endpoint Controls."

7.2.2.2.3 Endpoint Controls

7.2.2.2.3.1 Association Control

The Association Control is used to establish a link between the endpoint and the Entity who's ID is provided in the Parameter Block. Once the link is established, all data traveling over this endpoint will be delivered to or will originate from this Entity. The Association Control only supports the CUR attribute and must contain a valid EntityID. An association can only be established to Entities that support this feature. If a non-existent or invalid EntityID is specified, the control pipe must indicate a stall. One exception to this occurs when the **bEntityID** is set to 0 for a Set Association request. In this case, the current association is terminated and any subsequent data transfer to this endpoint results in a halt of the endpoint.

Table 7-5: Association Control Parameter Block

Control Selector		ASSOCIATION_CONTROL		
wLength		1		
Offset	**Field**	**Size**	**Value**	**Description**
0	bEntityID	1	Number	The ID of the Entity that is currently associated with this endpoint.

7.2.3 Additional Requests

7.2.3.1 Memory Requests

The Host can interact with an addressable Entity (Element, Jack, interface or endpoint) within the USB-MIDI function in a very generic way. The Entity presents a memory space to the Host whose layout depends on the implementation. The Memory request provides full access to this memory space.

7.2.3.1.1 Set Memory Request

This request is used to download a parameter block into a particular Entity of the USB-MIDI function.

Table 7-6: Set Memory Request Values

bmRequest Type	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_MEM	Offset	Entity ID and Interface	Length of parameter block	Parameter block
00100010B			Endpoint		

The **bRequest** field indicates that the MEM attribute of the Entity is addressed.

The **wValue** field specifies a zero-based offset value that can be used to access only parts of the Entity's memory space.

The layout of the parameter block is implementation dependent. A device is required to reevaluate its memory space at the end of each Set Memory request.

7.2.3.1.2 Get Memory Request

This request is used to upload a parameter block from a particular Entity of the USB-MIDI function.

Table 7-7: Get Memory Request Values

bmRequest Type	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_MEM	Offset	Entity ID and Interface	Length of parameter block	Parameter block
10100010B			Endpoint		

The **bRequest** field indicates that the MEM attribute of the Entity is addressed.

The **wValue** field specifies a zero-based offset value that can be used to access only parts of the Entity's parameter space.

The layout of the parameter block is implementation dependent.

7.2.3.2 Get Status Request

This request is used to retrieve status information from an Entity within the USB-MIDI function.

Table 7-8: Get Status Request Values

bmRequest Type	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_STAT	Zero	Entity ID and Interface	Status message length	Status message
10100010B			Endpoint		

The **bRequest** field contains the GET_STAT constant, defined in Section A.4, "*Class-Specific Request Codes*".

The **wValue** field is currently unused and must be set to zero.

The **wLength** field specifies the number of bytes to return. If the status message is longer than the **wLength** field, only the initial bytes of the status message are returned. If the status message is shorter than the **wLength** field, the function indicates the end of the control transfer by sending short packet when further data is requested.

The contents of the status message is reserved for future use. For the time being, a null packet should be returned in the data stage of the control transfer.

8 Glossary

8.1 MIDI: Musical Instrument Digital Interface

MIDI is a standard for musical instruments and audio devices to communicate with each other and with computers. For more information contact the MMA or AMEI at:

- **MIDI Manufacturers Association**

  P.O.Box 3173, La Habra, CA 90632-3173 USA
  Fax: (310) 947-4569
  E-mail: mma@midi.org
  http://www.midi.org/

- **Association of Musical Electronics Industry**

  Ito Bldg.4th Floor 2-16-9 Misaki-cho Chiyoda-ku Tokyo 101 Japan
  Fax:+81-3-5226-8549

8.2 GM: General MIDI

The General MIDI System is a universal set of specifications for sound generating devices. These specifications seek to allow for the creation of music data, which is not limited to equipment by a particular manufacturer, or to specific models. The General MIDI System defines the minimum number of voices that should be supported, the MIDI messages that should be recognized, which sounds correspond to which Program Change numbers, and the layout of rhythm sounds on the keyboard. Thanks to these specifications, any device that is equipped with sound sources supporting the General MIDI System, will be able to accurately reproduce General MIDI Scores (music data created for the General MIDI System), regardless of the manufacturer or model.

In September of 1991 the MIDI Manufacturers Association (MMA) and the Japan MIDI Standards Committee (JMSC) created the beginning of a new era in MIDI technology, by adopting the *General MIDI System Level 1* specification (GM). The specification is designed to provide a minimum level of performance compatibility among MIDI instruments, and has helped pave the way for MIDI in the growing consumer and multimedia markets.

The full specification document is part of the official *Complete MIDI 1.0 Detailed Specification* published by the MMA. Developers of General MIDI devices and compatible software (including MIDI files) should also obtain the new GM Developer Guidelines and Survey document to assist with understanding GM compatibility issues and additional recommendations.

8.3 Roland GS

The GS Format is Roland's set of specifications for standardizing the performance of sound generating devices. It includes support for everything defined by the General MIDI System. Furthermore, the highly compatible GS Format offers an expanded number of sounds, provides for the editing of sounds, and spells out many details for a wide range of extra features, including effects such as reverberation and chorus.

Designed with the future in mind, the GS Format can readily include new sounds and support new hardware features when they arrive. It is backward compatible with the General MIDI System. Therefore, Roland's GS Format is capable of reliably playing back GM Scores equally well as it performs GS Music Data (music data that has been created with the GS Format in mind).

8.4 Yamaha XG

The Yamaha XG format is a set of rules describing how a tone generator will respond to MIDI data. The current GM (General MIDI) format is a similar concept, allowing GM music data to be reproduced accurately on any GM tone generator from any manufacturer. GM, however, applies only to a limited set of parameters. XG significantly expands on the basic GM format, providing many more voices, voice editing capability, effects, external input, and other features that contribute to enhanced musical expression. Since XG is totally backward compatible with GM, GM data can be accurately reproduced on any XG tone generator.

Appendix A. Audio Device Class Codes: MIDIStreaming

A.1 MS Class-Specific Interface Descriptor Subtypes

Descriptor Subtype	Value
MS_DESCRIPTOR_UNDEFINED	0x00
MS_HEADER	0x01
MIDI_IN_JACK	0x02
MIDI_OUT_JACK	0x03
ELEMENT	0x04

A.2 MS Class-Specific Endpoint Descriptor Subtypes

Descriptor Subtype	Value
DESCRIPTOR_UNDEFINED	0x00
MS_GENERAL	0x01

A.3 MS MIDI IN and OUT Jack types

MIDI IN and OUT Jack type	Value
JACK_TYPE_UNDEFINED	0x00
EMBEDDED	0x01
EXTERNAL	0x02

A.4 Class-Specific Request Codes

For a complete list of class-specific Request Codes, refer to *the Universal Serial Bus Device Class Definition for Audio Devices.*

A.5 Control Selector Codes

A.5.1 Endpoint Control Selectors

Control Selector	Value
EP_CONTROL_UNDEFINED	0x00
ASSOCIATION_CONTROL	0x01

Appendix B. Example: Simple MIDI Adapter (Informative)

B.1 Device Descriptor

Table B-1: MIDI Adapter Device Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x12	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x01	DEVICE descriptor.
2	bcdUSB	2	0x0110	1.10 - current revision of USB specification.
4	bDeviceClass	1	0x00	Device defined at Interface level.
5	bDeviceSubClass	1	0x00	Unused.
6	bDeviceProtocol	1	0x00	Unused.
7	bMaxPacketSize0	1	0x08	8 bytes.
8	idVendor	2	0xXXXX	Vendor ID.
10	idProduct	2	0xXXXX	Product ID.
12	bcdDevice	2	0xXXXX	Device Release Code.
14	iManufacturer	1	0x01	Index to string descriptor that contains the string <Your Name> in Unicode.
15	iProduct	1	0x02	Index to string descriptor that contains the string <Your Product Name> in Unicode.
16	iSerialNumber	1	0x00	Unused.
17	bNumConfigurations	1	0x01	One configuration.

B.2 Configuration Descriptor

Table B-2: MIDI Adapter Configuration Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x02	CONFIGURATION descriptor.
2	wTotalLength	2	0x00XX	Length of the total configuration block, including this descriptor, in bytes.

Offset	Field	Size	Value	Description
4	bNumInterfaces	1	0x02	Two interfaces.
5	bConfigurationValu e	1	0x01	ID of this configuration.
6	iConfiguration	1	0x00	Unused.
7	bmAttributes	1	0x80	Bus Powered device, not Self Powered, no Remote wakeup capability.
8	MaxPower	1	0x32	100 mA Max. power consumption.

B.3 AudioControl Interface Descriptors

The AudioControl interface describes the device structure (audio function topology) and is used to manipulate the Audio Controls. This device has no audio function incorporated. However, the AudioControl interface is mandatory and therefore both the standard AC interface descriptor and the class-specific AC interface descriptor must be present. The class-specific AC interface descriptor only contains the header descriptor.

B.3.1 Standard AC Interface Descriptor

The AudioControl interface has no dedicated endpoints associated with it. It uses the default pipe (endpoint 0) for all communication purposes. Class-specific AudioControl Requests are sent using the default pipe. There is no Status Interrupt endpoint provided.

Table B-3: MIDI Adapter Standard AC Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x04	INTERFACE descriptor.
2	bInterfaceNumber	1	0x00	Index of this interface.
3	bAlternateSetting	1	0x00	Index of this setting.
4	bNumEndpoints	1	0x00	0 endpoints.
5	bInterfaceClass	1	0x01	AUDIO.
6	bInterfaceSubclass	1	0x01	AUDIO_CONTROL.
7	bInterfaceProtocol	1	0x00	Unused.
8	iInterface	1	0x00	Unused.

B.3.2 Class-specific AC Interface Descriptor

The Class-specific AC interface descriptor is always headed by a Header descriptor that contains general information about the AudioControl interface. It contains all the pointers needed to describe the Audio Interface Collection, associated with the described audio function. Only the Header descriptor is present in this device because it does not contain any audio functionality as such.

Table B-4: MIDI Adapter Class-specific AC Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x24	CS_INTERFACE.
2	bDescriptorSubtype	1	0x01	HEADER subtype.
3	bcdADC	2	0x0100	Revision of class specification - 1.0
5	wTotalLength	2	0x0009	Total size of class specific descriptors.
7	bInCollection	1	0x01	Number of streaming interfaces.
8	baInterfaceNr(1)	1	0x01	MIDIStreaming interface 1 belongs to this AudioControl interface.

B.4 MIDIStreaming Interface Descriptors

B.4.1 Standard MS Interface Descriptor

Table B-5: MIDI Adapter Standard MS Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x04	INTERFACE descriptor.
2	bInterfaceNumber	1	0x01	Index of this interface.
3	bAlternateSetting	1	0x00	Index of this alternate setting.
4	bNumEndpoints	1	0x02	2 endpoints.
5	bInterfaceClass	1	0x01	AUDIO.
6	bInterfaceSubclass	1	0x03	MIDISTREAMING.
7	bInterfaceProtocol	1	0x00	Unused.
8	iInterface	1	0x00	Unused.

B.4.2 Class-specific MS Interface Descriptor

Table B-6: MIDI Adapter Class-specific MS Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x07	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x24	CS_INTERFACE descriptor.
2	bDescriptorSubtype	1	0x01	MS_HEADER subtype.
3	BcdADC	2	0x0100	Revision of this class specification.
5	wTotalLength	2	0x0041	Total size of class-specific descriptors.

B.4.3 MIDI IN Jack Descriptor

Table B-7: MIDI Adapter MIDI IN Jack Descriptor (Embedded)

Offset	Field	Size	Value	Description
0	bLength	1	0x06	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x24	CS_INTERFACE descriptor.
2	bDescriptorSubtype	1	0x02	MIDI_IN_JACK subtype.
3	bJackType	1	0x01	EMBEDDED.
4	bJackID	1	0x01	ID of this Jack.
5	iJack	1	0x00	Unused.

Table B-8: MIDI Adapter MIDI IN Jack Descriptor (External)

Offset	Field	Size	Value	Description
0	bLength	1	0x06	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x24	CS_INTERFACE descriptor.
2	bDescriptorSubtype	1	0x02	MIDI_IN_JACK subtype.
3	bJackType	1	0x02	EXTERNAL.
4	bJackID	1	0x02	ID of this Jack.
5	iJack	1	0x00	Unused.

B.4.4 MIDI OUT Jack Descriptor

Table B-9: MIDI Adapter MIDI OUT Jack Descriptor (Embedded)

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x24	CS_INTERFACE descriptor.
2	bDescriptorSubtype	1	0x03	MIDI_OUT_JACK subtype.
3	bJackType	1	0x01	EMBEDDED.
4	bJackID	1	0x03	ID of this Jack.
5	bNrInputPins	1	0x01	Number of Input Pins of this Jack.
6	BaSourceID(1)	1	0x02	ID of the Entity to which this Pin is connected.
7	BaSourcePin(1)	1	0x01	Output Pin number of the Entity to which this Input Pin is connected.
8	iJack	1	0x00	Unused.

Table B-10: MIDI Adapter MIDI OUT Jack Descriptor (External)

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x24	CS_INTERFACE descriptor.
2	bDescriptorSubtype	1	0x03	MIDI_OUT_JACK subtype.
3	bJackType	1	0x02	EXTERNAL.
4	bJackID	1	0x04	ID of this Jack.
5	bNrInputPins	1	0x01	Number of Input Pins of this Jack.
6	BaSourceID(1)	1	0x01	ID of the Entity to which this Pin is connected.
7	BaSourcePin(1)	1	0x01	Output Pin number of the Entity to which this Input Pin is connected.
8	iJack	1	0x00	Unused.

B.5 Bulk OUT Endpoint Descriptors

B.5.1 Standard Bulk OUT Endpoint Descriptor

Table B-11: MIDI Adapter Standard Bulk OUT Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x05	ENDPOINT descriptor.
2	bEndpointAddress	1	0x01	OUT Endpoint 1.
3	bmAttributes	1	0x02	Bulk, not shared.
4	wMaxPacketSize	2	0x0040	64 bytes per packet.
6	bInterval	1	0x00	Ignored for Bulk. Set to zero.
7	bRefresh	1	0x00	Unused.
8	bSynchAddress	1	0x00	Unused.

B.5.2 Class-specific MS Bulk OUT Endpoint Descriptor

Table B-12: MIDI Adapter Class-specific Bulk OUT Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x05	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x25	CS_ENDPOINT descriptor
2	bDescriptorSubtype	1	0x01	MS_GENERAL subtype.
3	bNumEmbMIDIJack	1	0x01	Number of embedded MIDI IN Jacks.
4	BaAssocJackID(1)	1	0x01	ID of the Embedded MIDI IN Jack.

B.6 Bulk IN Endpoint Descriptors

B.6.1 Standard Bulk IN Endpoint Descriptor

Table B-13: MIDI Adapter Standard Bulk IN Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x09	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x05	ENDPOINT descriptor.

Offset	Field	Size	Value	Description
2	bEndpointAddress	1	0x81	IN Endpoint 1.
3	bmAttributes	1	0x02	Bulk, not shared.
4	wMaxPacketSize	2	0x0040	64 bytes per packet.
6	bInterval	1	0x00	Ignored for Bulk. Set to zero.
7	bRefresh	1	0x00	Unused.
8	bSynchAddress	1	0x00	Unused.

B.6.2 Class-specific MS Bulk IN Endpoint Descriptor

Table B-14: MIDI Adapter Class-specific Bulk IN Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	0x05	Size of this descriptor, in bytes.
1	bDescriptorType	1	0x25	CS_ENDPOINT descriptor
2	bDescriptorSubtype	1	0x01	MS_GENERAL subtype.
3	bNumEmbMIDIJack	1	0x01	Number of embedded MIDI OUT Jacks.
4	BaAssocJackID(1)	1	0x03	ID of the Embedded MIDI OUT Jack.