Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol

With MIDI 1.0 Protocol in UMP Format

MIDI Association Document: M2-104-UM

Document Version 1.1.2
Draft Date 2023-10-27

Published 2023-11-10

Developed and Published By
The MIDI Association
and
Association of Musical Electronics Industry (AMEI)

PREFACE

MIDI Association Document M2-104-UM
Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol

The Universal MIDI Packet, defined in this specification, provides a standardized modern packet format for all MIDI messages, both MIDI 1.0 Protocol and MIDI 2.0 Protocol. This specification also defines messages in the MIDI 2.0 Protocol. Some messages in the MIDI 2.0 Protocol may be used by MIDI 1.0 devices which communicate using the Universal MIDI Packet.

http://www.amei.or.jp https://www.midi.org

Version History

Table 1 Version History

Publication Date	Version	Changes
February 20, 2020	1.0	Initial release
May 11, 2023	1.1	Allowed Message Types to have no Group Field; implemented on MT= 0x0 and 0xF. Allocated MT = 0xD and 0xF. Added UMP Endpoint discovery and Protocol selection mechanisms. Added many new messages including some required for Standard MIDI Files v2.
July 15, 2023	1.1.1	Increased maximum size of Product Instance Id value from 16 bytes to 42 bytes. Changed Melisma events from a “Continue” packet to a Complete packet and clarified rules for Lyric Data and Ruby Data. Minor editorial improvements addressing errata in v1.1: Figures 32, 75, 76, 78 Section 7.2.3.2 Tables 22, 26 Appendix G
November 10, 2023	1.1.2	Minor fixes or improvements in Sections 2.1.2, 7.1.9, and Appendix D.3.1.

Contents

Figures

Tables

1 Introduction

1.1 Executive Summary

The Universal MIDI Packet, defined in this specification, provides a standardized modern packet format for all MIDI messages, both MIDI 1.0 Protocol and MIDI 2.0 Protocol.

To deliver an unprecedented level of nuanced musical and artistic expressiveness, the MIDI 2.0 Protocol re-imagines the role of performance controllers, the aspect of MIDI that translates human performance gestures to data computers can understand. Controllers are now easier to use, and there are more of them: over 32,000 controllers, including controls for individual notes. Enhanced, 32-bit resolution gives controls a smooth, continuous, "analog" feel.

Note-On capabilities in MIDI 2.0 enable further articulation control, precise note pitch. In addition, dynamic response (velocity) has been upgraded.

The Universal MIDI Packet is intended to be the packet format for all new transports defined by the MIDI Association. For example, UMP serves as the basis for the USB Class Specification for MIDI Devices, v2.0.

1.2 Background

This Specification defines two major extensions to the MIDI 1.0 Protocol:

Universal MIDI Packet (UMP) Format

UMP can contain all MIDI 1.0 Protocol messages and all MIDI 2.0 Protocol messages in a single, common container definition with a payload format which is intended to be usable in (or easily adaptable for) any standardized or proprietary data transport.

The UMP Format adds 16 Groups to MIDI addressing. Each Group contains an independent set of System Messages, and 16 Channels that are equivalent to the MIDI 1.0 Protocol's 16 MIDI Channels.

The UMP Format also adds a per-packet Jitter Reduction (JR) Timestamp mechanism: a JR Timestamp can be prepended to UMPs to improve timing accuracy.

MIDI 2.0 Protocol

The MIDI 2.0 Protocol is an extension of the MIDI 1.0 Protocol. Architectural concepts and semantics remain the same as MIDI 1.0. Compatibility for translation to/from the MIDI 1.0 Protocol is given high priority in the design of the MIDI 2.0 Protocol.

Compared to the MIDI 1.0 Protocol, MIDI 2.0 Protocol messages have extended data resolution for all Channel Voice Messages. New properties have been added to some Channel Voice Messages, and new Channel Voice Messages have been added with greatly improved Per-Note control and much more musical expression.

In addition, some functions that require the use of multiple MIDI Messages in the MIDI 1.0 Protocol (for example: Bank and Program Change, RPN, and NRPN) are easier to use in the MIDI 2.0 Protocol, as they are now implemented as a single, unified message.

A set of new Data Messages has been added, including System Exclusive 8 Messages (very similar to MIDI 1.0 Protocol System Exclusive message, but allowing use of all 8 data bits per byte) and Mixed Data Set Messages (for transfer of large data sets, including non-MIDI data).

Both the UMP Format and the MIDI 2.0 Protocol include a large, reserved space for future extensibility.

1.3 Reliance Upon Other Specifications

Implementers should understand that this Specification is not a stand-alone document, in the following regards:

The UMP Format sections describe a transport-independent payload format, not necessarily the low-level data format that will actually be used "on the wire" or "over the air" for any particular standardized transport (such as

USB, UDP, Bluetooth, Wi-Fi, etc.). MMA/AMEI expect that for every standardized transport that uses the UMP Format, a separate specification will exist to define how to carry UMP payload data for that standardized transport. See also Section ***2.1.1***.

The UMP Format and MIDI 2.0 Protocol descriptions are written as extensions of the MIDI 1.0 Protocol. Therefore, understanding this document and the technical design of the UMP Format requires comprehensive knowledge of the MIDI 1.0 Specification ***[MA01]***.

1.4 Changes in this Version of UMP and MIDI 2.0 Protocol

This version contains all errata and feature changes or additions since UMP and MIDI 2.0 Protocol version 1.0. The most significant changes include:

- Added the notion of Groupless messages. Some UMP Message Types are not sent to a specific Group. These messages are intended to be processed by the UMP Endpoint.
- Utility messages (Message Type 0x0) are now Groupless. The former Group field is changed to Reserved.
- Added Section ***6***, Function Blocks.
    - A Device may have one or more Function Blocks. A Function Block describes a single functional component or application that operates on a set of one or more Groups.
- Added UMP Endpoint Messages (Message Type 0xF)
    - These messages are used to discover details about a UMP Endpoint and its Function Blocks. Discovery of Max System Exclusive 8 Messages has been moved to these messages. These messages are Groupless.
- Added Flex Data Messages (Message Type 0xD)
    - Flex Data Messages are used to send messages to a Channel or a Group. New Flex Data Messages include Lyric and Text messages as well messages useful to the MIDI Clip Specification.
- Added Delta Clockstamps Messages for use in the MIDI Clip File specification.
- Deprecated MIDI-CI Protocol Negotiation
    - These mechanisms have been replaced by UMP Endpoint mechanisms and the Stream Configuration Request and Stream Configuration Notification Messages.
- Added a Registered Controller Message to set Per-Note Pitch Sensitivity
- Clarified specific CC/RPN Message translation between MIDI 1.0 and MIDI 2.0 Protocol
- Added MIDI 2.0 Addressing Appendix to help clarify if messages are meant for a Channel, Group, Function Block or UMP Endpoint.

1.5 References

1.5.1 Normative References

[COMM01]	***CommonMark Spec***, Version 0.28, *https://spec.commonmark.org/0.28/*
[ECMA01]	***The JSON Data Interchange Syntax***, ECMA-404, *https://www.ecma-international.org/publications/standards/Ecma-404.htm*
[MA01]	***Complete MIDI 1.0 Detailed Specification***, Document Version 96.1, Third Edition, Association of Musical Electronics Industry, *http://www.amei.or.jp/*, and The MIDI Association, *https://www.midi.org/*
[MA02]	***M2-100-U MIDI 2.0 Specification Overview***, Version 1.1, Association of Musical Electronics Industry, *http://www.amei.or.jp/*, and The MIDI Association, *https://www.midi.org/*
[MA03]	***M2-101-UM MIDI Capability Inquiry (MIDI-CI)***, Version 1.2, Association of Musical Electronics Industry, *http://www.amei.or.jp/*, and The MIDI Association, *https://www.midi.org/*
[MA04]	***M2-102-U Common Rules for MIDI-CI Profiles***, Version 1.1, Association of Musical Electronics Industry, *http://www.amei.or.jp/*, and The MIDI Association, *https://www.midi.org/*
[MA05]	***M2-103-UM Common Rules for Property Exchange***, Version 1.1, Association of Musical Electronics Industry, *http://www.amei.or.jp/*, and The MIDI Association, *https://www.midi.org/*
[MA06]	***M2-104-UM Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol***, Version 1.0, Association of Musical Electronics Industry, *http://www.amei.or.jp/*, and The MIDI Association, *https://www.midi.org/*
[MA07]	***M1-100-UM MIDI Polyphonic Expression***, Version 1.1, Association of Musical Electronics Industry, *http://www.amei.or.jp/*, and The MIDI Association, *https://www.midi.org/*
[MA08]	***M2-115-MIDI 2.0 Bit Scaling and Resolution***, Version 1.0, Association of Musical Electronics Industry, *http://www.amei.or.jp/*, and The MIDI Association, *https://www.midi.org/*
[MA09]	***M2-116-MIDI Clip File***, Version 1.0, Association of Musical Electronics Industry, *http://www.amei.or.jp/*, and The MIDI Association, *https://www.midi.org/*
[USBIF01]	***USB Class Specification for MIDI Devices***, Version 1.0, USB Implementers Forum, *https://www.usb.org/*
[USBIF02]	***USB Class Specification for MIDI Devices***, Version 2.0, USB Implementers Forum, *https://www.usb.org/*
[UNIC01]	***Unicode Standard***, Annex #15, Unicode Consortium, *https://www.unicode.org/reports/tr15/*

1.6 Terminology

1.6.1 Definitions

100-Cent Unit (HCU): A unit of measure for musical intervals, corresponding to one-twelfth of an octave measured logarithmically. This term is preferred over "semitone" which may refer to various intervals.

AMEI: Association of Musical Electronics Industry. Authority for MIDI Specifications in Japan.

Clock: An expression of musical progression, as measured in bars and beats (and further subdivisions).

Device: An entity, whether hardware or software, which can send and/or receive MIDI messages.

Function Block: A single logical entity which describes the functional components available on a UMP Endpoint of a Device, A Function Block operates on a set of one or more Groups.

Group: A field in the UMP Format addressing some UMP Format MIDI message (and some UMP comprising any given MIDI message) to one of 16 Groups. See the M2-104-UM Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol specification ***[MA06]***.

HCU: See 100-Cent-Unit

JSON: JavaScript Object Notation as defined in ***[ECMA01]***.

MA: See MIDI Association.

MIDI 1.0 Protocol: Version 1.0 of the MIDI Protocol as originally specified in ***[MA01]*** and extended by MA and AMEI with numerous additional MIDI message definitions and Recommended Practices. The native format for the MIDI 1.0 Protocol is a byte stream, but it has been adapted for many different transports. MIDI 1.0 messages can be carried in UMP packets. See Section ***3***.

MIDI 1.0 Specification: Complete MIDI 1.0 Detailed Specification, Document Version 96.1, Third Edition ***[MA01]***.

MIDI 2.0: The MIDI environment that encompasses all of MIDI 1.0, MIDI-CI, Universal MIDI Packet (UMP), MIDI 2.0 Protocol, MIDI 2.0 messages, and other extensions to MIDI as described in AMEI and MA specifications.

MIDI 2.0 Protocol: Version 2.0 of the MIDI Protocol. The native format for MIDI 2.0 Protocol messages is UMP Format. See Section ***3***.

MIDI Association: Authority for MIDI specifications worldwide except Japan. See also MMA.

MIDI-CI: MIDI Capability Inquiry ***[MA03]***, a specification published by The MIDI Association and AMEI.

MIDI Endpoint: A Device which is an original source of MIDI messages or final consumer of MIDI messages.

MIDI In: A hardware or software MIDI connection used by a MIDI Device to receive MIDI messages from a MIDI Transport.

MIDI Manufacturers Association: A California nonprofit 501(c)6 trade organization, and the legal entity name of the MIDI Association.

MIDI Out: A hardware or software MIDI connection used by a MIDI Device to transmit MIDI messages to a MIDI Transport.

MIDI Port: A hardware or software connector associated with a MIDI Endpoint using messages in MIDI 1.0 data format.

MIDI Thru: A hardware or software MIDI connection used by a MIDI Device to retransmit MIDI messages the device has received from a MIDI In.

MIDI Transport: A hardware or software MIDI connection used by a Device to transmit and/or receive MIDI messages to and/or from another Device.

MMA: See MIDI Manufacturers Association.

Port: See MIDI Port.

Profile: An MA/AMEI specification that includes a set of MIDI messages and defined responses to those messages. A Profile is controlled by MIDI-CI Profile Configuration Transactions. A Profile may have a defined minimum set of mandatory messages and features, along with some optional or recommended messages and features. See the MIDI-CI specification ***[MA03]*** and the Common Rules for MIDI-CI Profiles ***[MA04]***.

Property Exchange: A set of MIDI-CI Transactions by which one device may access properties from another device. See the MIDI-CI specification ***[MA03]*** and the Common Rules for Property Exchange [MA05].

Protocol: There are two defined MIDI Protocols: the MIDI 1.0 Protocol and the MIDI 2.0 Protocol, each with a data structure that defines the semantics for MIDI messages. See ***[MA01]*** and ***[MA06]***.

Receiver: A MIDI Device which has a MIDI Transport connected to its MIDI In.

Sender: A MIDI Device which transmits MIDI messages to a MIDI Transport which is connected to its MIDI Out or to its MIDI Thru Port.

Tempo: The rate at which a passage of music is or should be played, declared as and measured in a number of Clocks per a unit of Time (typically beats per minute).

Time: An expression of time as measured in hours, minutes, and seconds (and further subdivisions).

Transaction: An exchange of MIDI messages between two MIDI Devices with a bidirectional connection. All the MIDI messages in a single Transaction are associated and work together to accomplish one function. The simplest Transaction generally consists of an inquiry sent by one MIDI Device and an associated reply returned by a second MIDI Device. A Transaction may also consist of an inquiry from one MIDI Device and several associated replies from a second MIDI Device. A Transaction may be a more complex set of message exchanges, started by an initial inquiry from one MIDI Device and multiple, associated replies exchanged between the first MIDI Device and a second MIDI Device.

UMP: Universal MIDI Packet, see ***[MA06]***.

UMP Endpoint: A MIDI Endpoint which uses the UMP Format.

UMP Format: Data format for fields and messages in the Universal MIDI Packet, see ***[MA06]***.

UMP MIDI 1.0 Device: any Device that sends or receives MIDI 1.0 Protocol messages using the UMP ***[MA06]***. Such Devices may use UMP Message Types that extend the functionality beyond Non-UMP MIDI 1.0 Systems.

Universal MIDI Packet (UMP): The Universal MIDI Packet is a data container which defines the data format for all MIDI 1.0 Protocol messages and all MIDI 2.0 Protocol messages. UMP is intended to be universally applicable, i.e., technically suitable for use in any transport where MA/AMEI elects to officially support UMP. For detailed definition see M2-104-UM Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol specification ***[MA06]***.

USB Endpoint: The source or sink of data sent over USB, as defined in the USB core specifications.

USB-MIDI Endpoint: A USB Endpoint used to transfer MIDI Data as defined in the two USB Class Specification for MIDI Devices documents ***[USBIF01]*** and ***[USBIF02]***.

1.6.2 Reserved Words and Specification Conformance

In this document, the following words are used solely to distinguish what is required to conform to this specification, what is recommended but not required for conformance, and what is permitted but not required for conformance:

Table 2 Words Relating to Specification Conformance

Word	Reserved For	Relation to Specification Conformance
shall	Statements of requirement	**Mandatory** A conformant implementation conforms to all ‘shall’ statements.
should	Statements of recommendation	**Recommended but not mandatory.** An implementation that does not conform to some or all ‘should’ statements is still conformant, providing all ‘shall’ statements are conformed to.
may	Statements of permission	**Optional** An implementation that does not conform to some or all ‘may’ statements is still conformant, providing that all ‘shall’ statements are conformed to.

By contrast, in this document, the following words are never used for specification conformance statements; they are used solely for descriptive and explanatory purposes:

Table 3 Words Not Relating to Specification Conformance

Word	Reserved For	Relation to Specification Conformance
must	Statements of unavoidability	Describes an action to be taken that, while not required (or at least not directly required) by this specification, is unavoidable. Not used for statements of conformance requirement (see ‘shall’ above).
will	Statements of fact	Describes a condition that as a question of fact is necessarily going to be true, or an action that as a question of fact is necessarily going to occur, but not as a requirement (or at least not as a direct requirement) of this specification. Not used for statements of conformance requirements (see ‘shall’ above).
can	Statements of capability	Describes a condition or action that a system element is capable of possessing or taking. Not used for statements of conformance permission (see ‘may’ above).
might	Statements of possibility	Describes a condition or action that a system element is capable of electing to possess or take. Not used for statements of conformance permission (see ‘may’ above).

1.7 Bit Scaling and Resolution

For critical information on understanding resolution of various fields in MIDI messages in the UMP Format, see the MIDI 2.0 Bit Scaling and Resolution specification ***[MA08]***. That document defines recommended practices for scaling values, handling of stepped/enumerated values and translating values between MIDI 1.0 Protocol and MIDI 2.0 Protocol (see also ***Appendix D***: Translation: MIDI 1.0 and MIDI 2.0 Messages).

1.8 Unicode in Message Fields

In many MIDI messages Unicode text can be exchanged. Often, these texts will have to be displayed on a device display. The Unicode standard specifies a lot of character sets. In some cases, there are even multiple ways to encode the same character.

MIDI messages only use normalized UTF-8 encoded Unicode characters, following the NFC (Normalization Form C) standard, as specified in the Unicode Standard Annex #15 ***[UNIC01]***.

> *Note: Some messages may define a different encoding for a unique purpose. For example, MIDI-CI messages use pure ASCII characters converted to UTF-16, and then escaped using "\u" as defined in the JSON standard **[ECMA01]**.*

Unicode text in MIDI messages shall not include a Byte Order Mark.

2 Universal MIDI Packet (UMP) Format

Using the format defined in Section 2.1, the Universal MIDI Packet (UMP) Format supports:

- All MIDI 1.0 Protocol Channel Voice Messages
- All MIDI 2.0 Protocol Channel Voice Messages
- The System Common, System Real Time, System Exclusive, System Exclusive 8, Mixed Data Set, Flex Data, Utility, and UMP Stream messages.

See also see ***Appendix F: All Defined UMP Formats***.

2.1 UMP Basic Packet and Message Format

Each UMP shall be one, two, three, or four 32-bit words long.

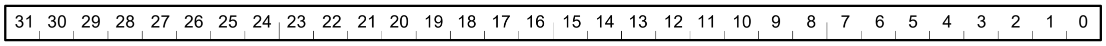
Each UMP shall contain one entire MIDI Message, or (in the sole case of Data Messages longer than 128 bits) part of one MIDI Message, and no additional data.

A MIDI Message that is longer than a single UMP allows will span multiple UMPs.

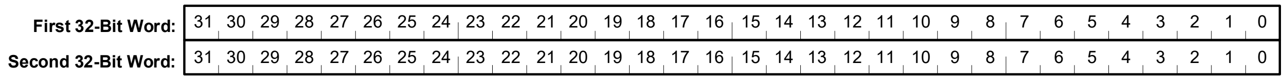
2.1.1 Bit, Byte, and Word Order in UMP Format Diagrams

In this specification, for clarity UMP Format diagrams present one 32-bit word per line. The leftmost bits are the most significant bits, for each 32-bit word and for each field within each 32-bit word.

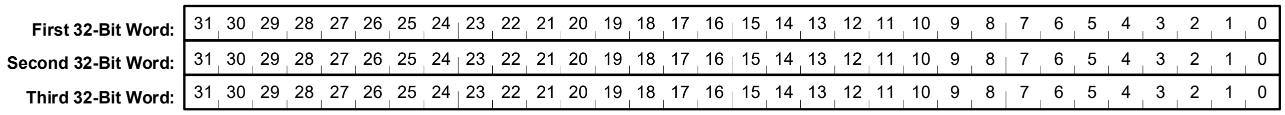
Example Diagram 1: 32-Bit Message in a Single 32-Bit UMP



Example Diagram 2: 64-Bit Message in a Single 64-Bit UMP



Example Diagram 3: 96-Bit Message in a Single 96-Bit UMP



Example Diagram 4: 128-Bit Message in a Single 128-Bit UMP

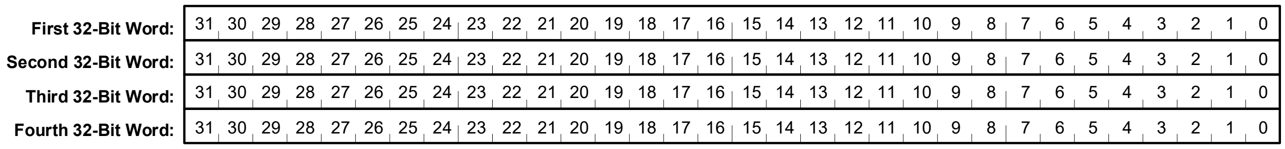


Figure 1 Example UMP Format Diagrams

Scope of Bit, Byte, and Word Order Guidance

Although UMP 32-bit words can be converted to and from byte streams for storage or transmission, the formats of such byte streams, including the byte order to be used for such transport and storage, are outside the scope of this specification. Per Section ***1.3,*** separate transport specifications will define formats and byte orders for each particular transport, and separate file format specifications addressing the UMP Format will define byte orders for each particular file format.

For the internals of any given implementation, a device or system may use any desired format, including native-endian 32-bit words.

2.1.2 UMP Format Commonly Used Fields

The following fields have common meanings across multiple Message Types.

Message Type

The most significant 4 bits in every UMP shall contain the Message Type field, detailed in Section ***2.1.4***. This field indicates the message's general functional area (e.g., Utility, MIDI 1.0 Channel Voice Messages, MIDI 2.0 Channel Voice Messages), as well as the UMP's size, and the size of the Status field.

Group

Group field is 4-bit value to address a UMP MIDI Message to one of 16 Groups.

Group field values 0-15 = Groups 1-16 in specifications and when presented to users.

Each Group's set of 16 MIDI Channels shall be separate and independent from any other Group's set of MIDI Channels, allowing up to 256 MIDI Channels (i.e., 16 Groups x 16 MIDI Channels) per UMP-based MIDI connection for Channel-based MIDI Messages. UMPs addressed to different Groups may be freely interleaved (i.e., transmitted in any order).

Within a given Group, MIDI Messages that do not support a MIDI Channel field (i.e., System Messages and Data Messages) shall apply to, and shall affect, all MIDI Channels within that Group. Groups are treated in the same manner as Channels for addressing purposes. Communication requires a Sender and Receiver to be on the same Group.

Messages without a Group Field

Messages of Message Type = 0x0 and Message Type = 0xF do not have a Group field. Other Message Types which are currently reserved may be defined in the future with or without a Group field.

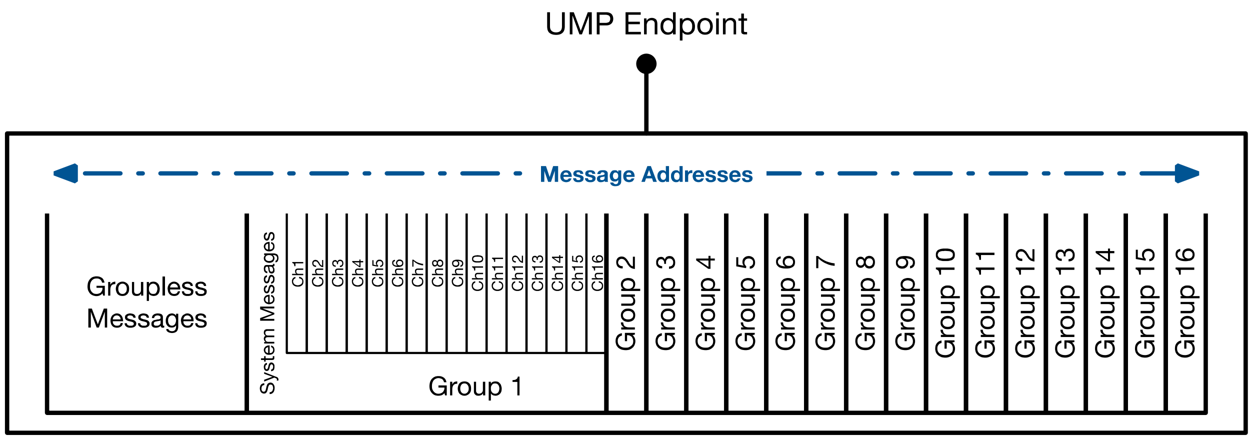


Figure 2 UMP Stream and Addressing

Status

Within each Message Type multiple messages are defined. Each message in the Message Type has its own Status value. As detailed in the UMP Format for each MIDI Message, the size in bits of the Status field depends upon the value of the Message Type.

For example, Message Type 0x2 is “MIDI 1.0 Channel Voice Messages” which contains the MIDI 1.0 Note Off, MIDI 1.0 Note On, MIDI 1.0 Program Change, and other related messages; the Status field selects one particular message within that Message Type.

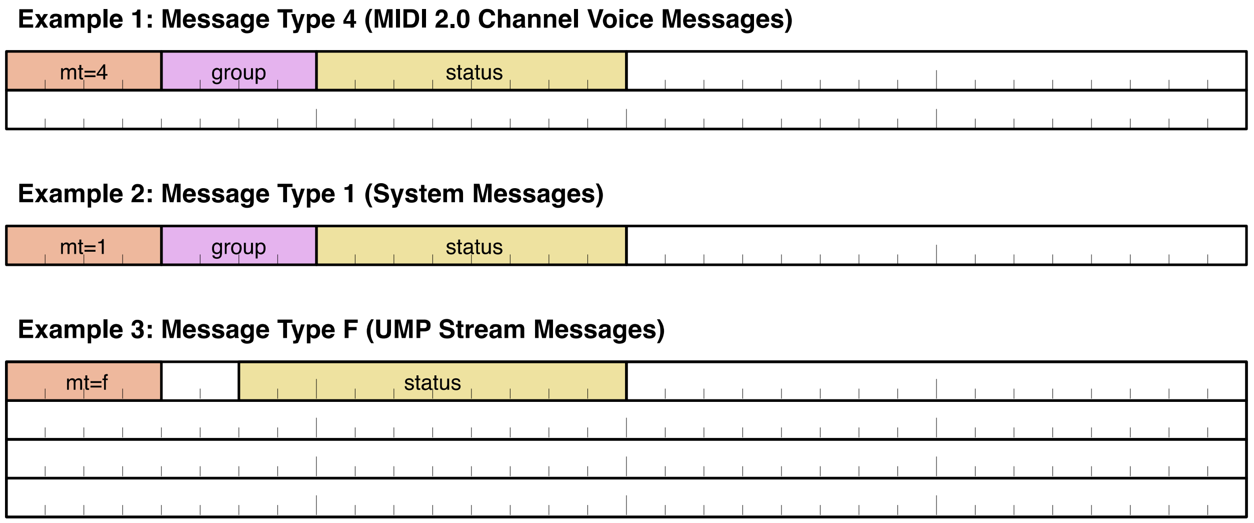

Example 1: Message Type 4 (MIDI 2.0 Channel Voice Messages)
mt=4
group
status
Example 2: Message Type 1 (System Messages)
mt=1
group
status
Example 3: Message Type F (UMP Stream Messages)
mt=f
status

Figure 3 Status Field Size Varies with Message Type Value

2.1.3 Reserved Items

In this specification, the term Reserved means reserved for future definition by MMA/AMEI.

In particular:

- Messages marked as Reserved shall not be used.
- Fields marked as Reserved shall be set to zero and shall not be used for any purpose.
- Bits marked “r” are reserved, shall be set to zero, and shall not be used for any purpose.
- Option flag bits that are undefined are reserved, shall be set to zero, and shall not be used for any purpose.
- Receivers, Translators, transports, or other MIDI system components shall not depend upon “r” bits or Reserved fields necessarily containing the value zero, to allow for future definitions with new uses for the reserved values.

2.1.4 Message Type (MT) Allocation

The most significant 4 bits of every message contain the Message Type (MT). The Message Type is used as a classification of message functions. All messages within a Message Type have the same UMP size.

Table 4 Message Type (MT) Allocation

MT	UMP Size	Description
0x0	32 bits	Utility Messages
0x1	32 bits	System Real Time and System Common Messages (except System Exclusive)
0x2	32 bits	MIDI 1.0 Channel Voice Messages
0x3	64 bits	Data Messages (including System Exclusive)
0x4	64 bits	MIDI 2.0 Channel Voice Messages

0x5	128 bits	Data Messages
0x6	32 bits	Reserved for future definition by MMA/AMEI
0x7	32 bits	
0x8	64 bits	
0x9	64 bits	
0xA	64 bits	
0xB	96 bits	
0xC	96 bits	
0xD	128 bits	Flex Data Messages
0xE	128 bits	Reserved for future definition by MMA/AMEI
0xF	128 bits	UMP Stream Messages

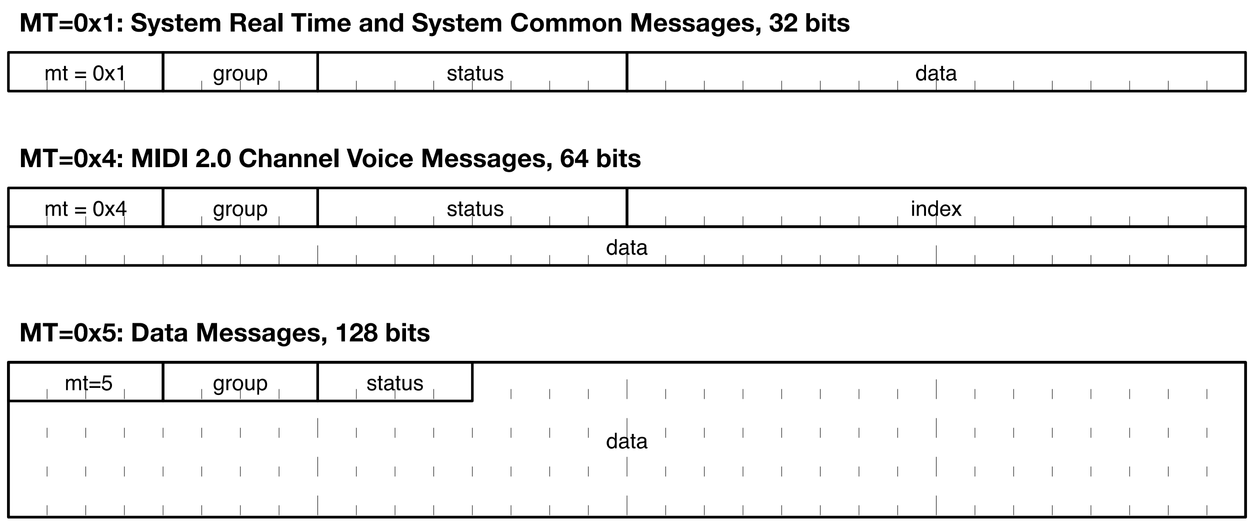

MT=0x1: System Real Time and System Common Messages, 32 bits
mt = 0x1
group
status
data
MT=0x4: MIDI 2.0 Channel Voice Messages, 64 bits
mt = 0x4
group
status
index
data
MT=0x5: Data Messages, 128 bits
mt=5
group
status
data

Figure 4 UMP Formats for Example Message Types

Reserved Message Types

Per Section ***2.1.3***, Message Types marked Reserved in ***Table 4*** are reserved for future definition by MMA/AMEI and shall not be used.

These reserved Message Types provide extensibility for future standardization. They have predefined sizes so that system components such as APIs, Transports, and Interfaces can be designed in advance to give basic support for those Message Types, even though the data within the messages are not yet defined.

3 MIDI Protocols in UMP Format

3.1 Overview

The UMP Format is capable of encoding multiple MIDI protocols. This version of the UMP Format specification defines support for the MIDI 1.0 Protocol and the MIDI 2.0 Protocol.

Stream Configuration Request (see Section *7.1.6.2*) is the defined method for selecting MIDI Protocols. Endpoint Discovery message (see Section *7.1.1*) is used to discover the Protocols available. Some devices, interfaces, APIs, or transports might have additional means for discovering or selecting protocols to fit the needs of a particular MIDI system.

3.2 MIDI 1.0 Protocol in UMP Format

MIDI 1.0 Protocol messages are carried in the UMP using several Message Types. UMP MIDI 1.0 Devices may use any of the Message Types listed in Section ***3.2.1.1***. UMP MIDI 1.0 Devices may also use messages from the Message Types listed in Section ***3.2.1.2*** within the same Group to add new functionality. But UMP MIDI 1.0 Devices shall not use any messages from Message Type 0x4, MIDI 2.0 Channel Voice Messages.

3.2.1 Message Types for MIDI 1.0 Protocol

There are two categories of UMP Message Types for the MIDI 1.0 Protocol: those that simply support traditional (i.e., pre-UMP) MIDI 1.0 Protocol functionality, and those that extend it.

3.2.1.1 Message Types for Traditional MIDI 1.0 Functionality

The following Message Types encapsulate all traditional MIDI 1.0 Protocol messages:

- Message Type 0x1 System Real Time and System Common Messages
- Message Type 0x2 Channel Voice Messages
- Message Type 0x3 Data Messages (for System Exclusive)

3.2.1.2 Message Types to Extend MIDI 1.0 Functionality

UMP MIDI 1.0 Devices may also use the following Message Types to add extended functionality:

- Message Type 0x0 Utility Messages
- Message Type 0x5 SysEx 8 and Mixed Data Set Messages
- Message Type 0xD Flex Data Messages
- Message Type 0xF UMP Stream Messages

Note: UMP MIDI 1.0 Devices shall NOT use any messages from Message Type 0x4, MIDI 2.0 Channel Voice Messages.

3.2.2 MIDI 1.0 Protocol and Future Expansion

Per Section ***2.1.3*** and Section ***2.1.4***, several Message Type values are reserved for future use, to be defined solely by MMA/AMEI. Whenever MMA/AMEI do define new messages that use these currently Reserved Message Types, it will be clearly specified whether UMP-based MIDI 1.0 Protocol Devices may (vs. shall not) use each of those messages.

3.3 MIDI 2.0 Protocol in UMP Format

The MIDI 2.0 Protocol expands on the architectural concepts and semantics of the MIDI 1.0 Protocol. The MIDI 2.0 Protocol increases the data resolution for all Channel Voice Messages, and makes some messages easier to use by aggregating some combination of multiple messages into a single, unified message. Some MIDI 2.0 Channel Voice Messages have additional properties which are not available in the corresponding MIDI 1.0 Protocol messages. Several new Channel Voice Messages are added to provide increased Per-Note control and musical expression.

MIDI 2.0 Protocol messages are carried in the UMP Format using several Message Types. MIDI 2.0 Protocol Devices may use any of these messages, and may also use messages from certain other defined Message Types within the same Group to add new functionality.

3.3.1 Message Types for MIDI 2.0 Protocol

The following Message Types contain all of the core MIDI 2.0 Protocol messages. MIDI 2.0 functionality may be implemented using these Message Types:

- Message Type 0x1 System Real Time and System Common Messages
- Message Type 0x4 MIDI 2.0 Channel Voice Messages
- Message Type 0x3 Data Messages (for System Exclusive)
- Message Type 0x0 Utility Messages
- Message Type 0x5 Data Messages
- Message Type 0xD Flex Data Messages
- Message Type 0xF UMP Stream Messages

MIDI 2.0 Protocol Devices shall not use any messages from Message Type 0x2, MIDI 1.0 Channel Voice Messages.

3.3.2 MIDI 2.0 Protocol and Future Expansion

Per Section ***2.1.3*** and Section ***2.1.4***,, several Message Type values are reserved for future use, to be defined solely by MMA/AMEI. Whenever MMA/AMEI do define new messages that use these currently Reserved Message Types, it will be clearly specified whether MIDI 2.0 Protocol Devices may (vs. shall not) use each of those messages.

4 Jitter Reduction (JR) Clock and Timestamps

4.1 Overview

The UMP Format provides a method of managing jitter for a UMP Stream. Jitter Reduction Timestamps are intended to help capture a performance with accurate timing. It may also be used for transmitting MIDI Messages with accurate timing over a system that is subject to jitter.

The Jitter Reduction mechanism is a simple, peer to peer mechanism and does not depend on system-wide synchronization, central clock, or explicit clock synchronization between Sender and Receiver. Each UMP Endpoint can choose if Jitter Reduction is one direction or bi-directional.

All messages from a Sender can be transmitted with a Jitter Reduction Timestamp prepended. Time is based on the Sender's notion of time. Therefore, the Sender also sends regular clock messages to declare its current time. For more details of the mechanism, see Section *7.2.2.1*.

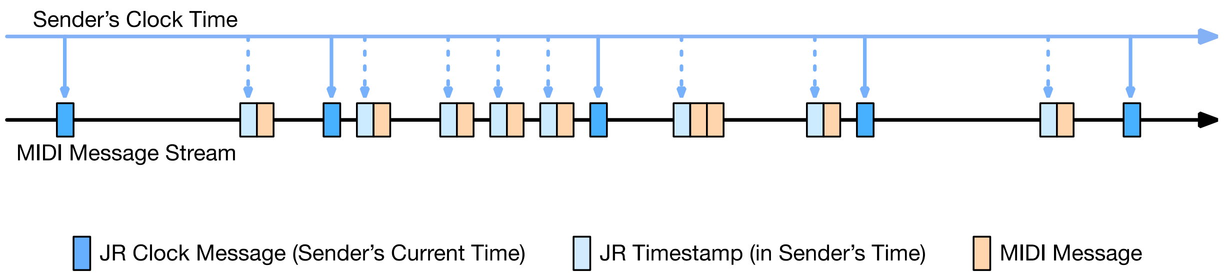

Sender's Clock Time
MIDI Message Stream
JR Clock Message (Sender's Current Time)
JR Timestamp (in Sender's Time)
MIDI Message

Figure 5 Sequence of MIDI Messages with JR Timestamps Prepended

Devices negotiate whether JR Timestamps will be used. Endpoint Discovery message (see Section *7.1.1*) is used to discover if Jitter Reduction Timestamps are supported by a Device. Stream Configuration Request (see Section *7.1.6.2*) and Stream Configuration Notification message (see Section *7.1.6.3*) are used to enable or disable JR Timestamps.

Note: There are two different sources of error for timing: Jitter (precision) and Latency (sync). The Jitter Reduction Timestamp mechanism only addresses the errors introduced by jitter. The problem of synchronization or time alignment across multiple devices in a system requires a measurement of latency. This is a complex problem and is not addressed by the JR Timestamping mechanism.

4.1.1 Translation to/from the MIDI 1.0 Protocol

JR Clock and JR Timestamps cannot be translated to Non-UMP MIDI 1.0 Systems, but they can be used by a Translator to improve timing. If a Translator understand JR Timestamps and receives messages with JR Timestamps, then when translating from a connection with JR Timestamps to a connection that does not support JR Timestamps, the Translator shall schedule the MIDI 1.0 Protocol messages according to the received JR Timestamps. When translating from a connection that does not support JR Timestamps to a connection with JR Timestamps, the Translator may generate JR Timestamps based on the time of reception.

5 Device Discovery

The UMP Format defines mechanisms for Devices to discover fundamental properties of other Devices to connect, communicate and address messages. Discoverable properties include:

1. Device Identifiers: Name, Manufacturer, Model, Version, and Product Instance Id (e.g. Serial Number).
2. Data Formats Supported: Version of UMP Format*, MIDI Protocols, and whether Jitter Reduction Timestamps can be used.
3. Device Topology: including which Groups are currently valid for transmitting and receiving messages and which Groups are available for MIDI-CI transactions.

These properties can be used for Devices to auto-configure through bidirectional transactions, thereby enabling the best connectivity between the Devices. These properties can also provide useful information to users for manual configuration.

** Note: The version of UMP Format can be used to determine which UMP messages may be used and the format of those messages, including the messages for these Device discovery transactions.*

These mechanisms use Groupless messages defined in Section ***7.1***.

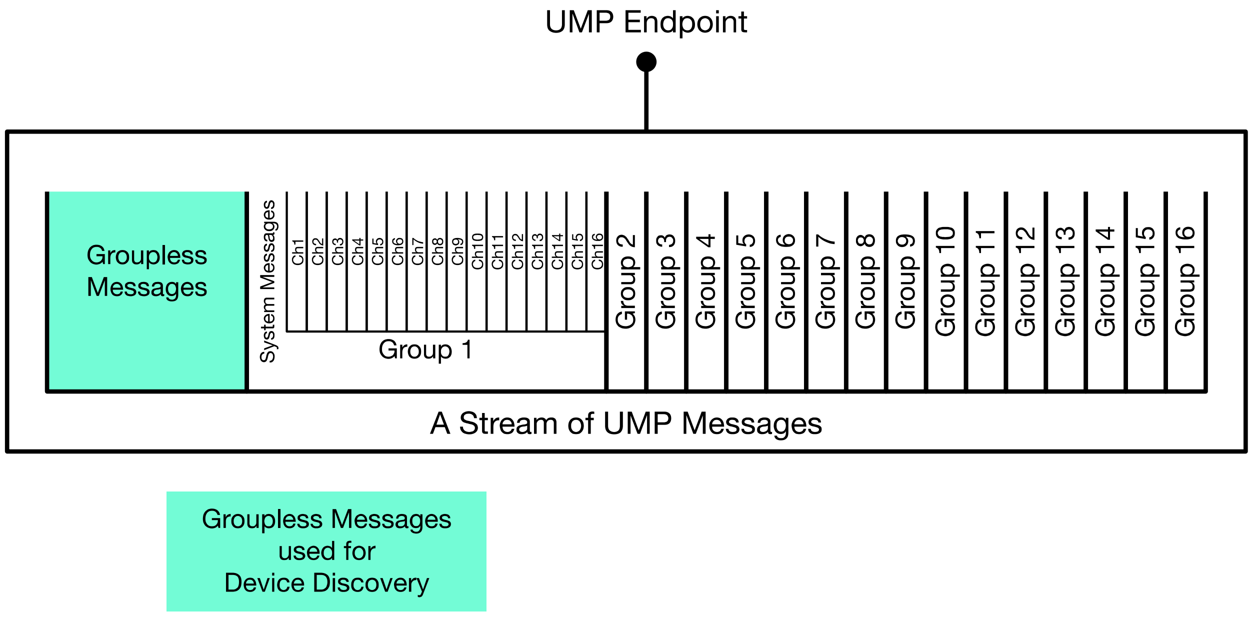


Figure 6 Device Discovery Messages on a UMP Endpoint

6 Function Blocks

6.1 Overview

A Device may describe the functions it has available on a UMP Endpoint by declaring a Function Block to represent each function.

A Device uses Function Block related messages to report topology information including the Group address(es) in use and directionality of the function. It also reports metadata including the name of the function, MIDI-CI support, hints for user interfaces which expose senders and receivers for user connection choices, and more.

- Example 1: A piano might operate on a single MIDI Channel of a single Group. The piano has a single Function Block that sends notes and sustain pedal from its keyboard and receives the same messages into its tone generator.
- Example 2: A hypothetical workstation might have three separate functions: one for the keyboard, one for the tone generator, and another for a DAW transport control. The keyboard uses one Group as it only sends out on one MIDI Channel, the tone generator uses two Groups as it provides up to 32 channels of multitimbrality, and the DAW transport control uses one Group. *The keyboard, tone generator and DAW transport control sequencers are each represented by a Function Block (3 Function Blocks in total).*

Function Blocks provide metadata to help a connected Device understand the functional components inside a remote UMP Endpoint.

- For example, *when a DAW user selects an output destination for a MIDI track, the destination UMP Endpoint's Function Blocks can present which Group addresses have active functionality with a descriptive name for display to the user.*

Function Blocks help user configuration or auto-configuration for a Sender and Receiver to be on matching Groups (and Channels) necessary for communication. Therefore, Sender and Receiver may change the Group addresses used by their Function Blocks to be on matching Group addresses.

- *For example, Device A declares it has a Tone Generator on Groups 3-4. Device B may move its Keyboard function from Group 1 to Group 3 so that the two Devices send and receive MIDI Messages on the same Group. See Declaring a Change in a Function Block in Section **6.1.2**.*

See Sections ***7.1.1, 7.1.2, 7.1.7,*** and ***7.1.8*** for how Function Blocks are declared in UMP. If a UMP Endpoint does not declare any Function Block topology, then the user has less information on how to configure Devices to be on matching Group (and Channel) addresses for communication.

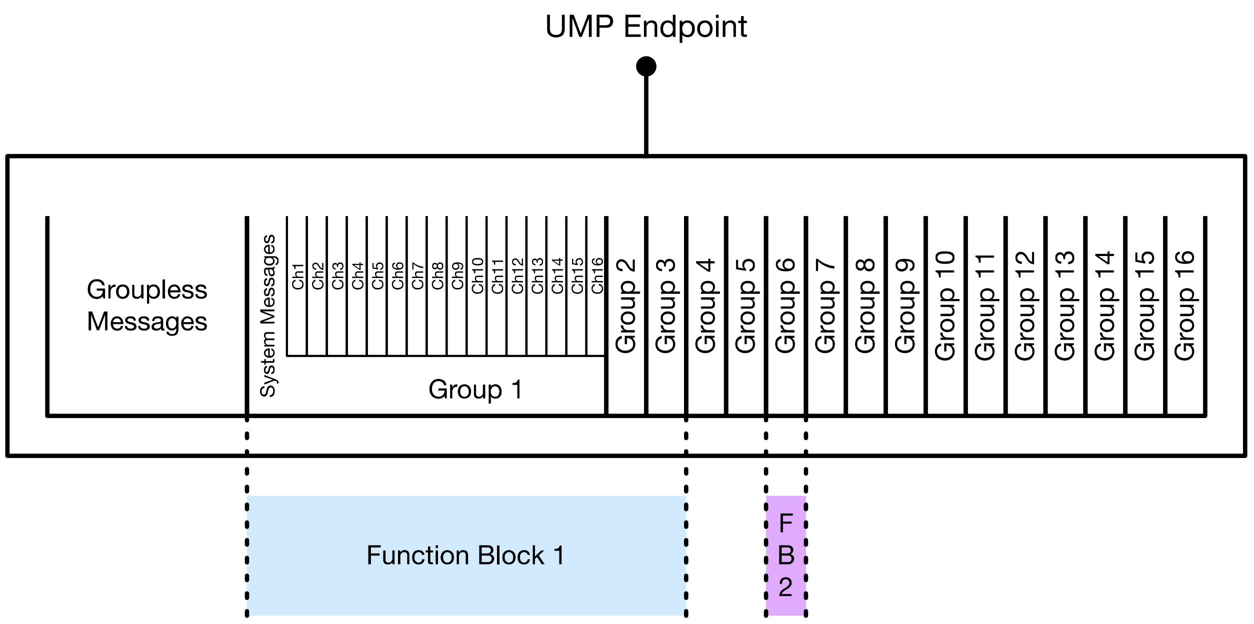


Figure 7 Example of Group and Channel Addresses and Function Block Topology

Note: A Device might expose multiple functions by way of multiple UMP Endpoints instead of or in addition to declaring multiple Function Blocks on a single UMP Endpoint.

Note: The USB Class Specification for MIDI Devices, Version 2.0, [USBIF02] defines Group Terminal Blocks which is a similar concept to Function Blocks. There is a potential overlap of features or conflict of declared topology. See Appendix I: Using USB MIDI Group Terminal Blocks and Function Blocks.

6.1.1 Function Blocks Features

Function Blocks have the following features:

- Function Blocks may start on any Group and span 1 to 16 Groups. Group numbers within a Function Block are monotonically increasing.
- A UMP Endpoint may declare up to 32 Function Blocks.
- Function Blocks may be bi-directional, input only, or output only.
- Function Blocks may change their starting Group and number of Groups spanned.
- There may be device designs or use cases where Function Blocks need to overlap. Therefore, any number of Function Blocks may exist on any Group. Each UMP Endpoint can choose to allow its Function Blocks to overlap or not.
- A Function Block has specific rules for use of MIDI-CI. See ***[MA02]*** MIDI Capability Inquiry (MIDI-CI) specification (version 1.2 or higher) for details.

Optionally, a UMP Endpoint can declare its Function Blocks to be static. In this case, a UMP Endpoint will not change any of the properties of its Function Blocks after initial discovery, hence it guarantees to not change the Group addresses of Function Blocks nor Function Block names.

6.1.2 Declaring a Change in a Function Block

Upon changing its Function Block structure, a Device sends a Function Block Info Notification message (See Section ***7.1.8***) to inform other Devices that there has been a change to the Device's Function Block structure.

6.2 MIDI 1.0 Byte Stream Ports

A UMP Endpoint may include connections to internal MIDI 1.0 functions or to external Devices by MIDI 1.0 Byte Stream Ports (such a 5 Pin DIN) proxied on the UMP Stream. These MIDI 1.0 connections may be represented by Function Blocks.

If a UMP Endpoint declares MIDI 2.0 Protocol but a Function Block represents a MIDI 1.0 connection, then MIDI 1.0 Protocol may optionally be used for messages to/from that Function Block.

If a UMP Endpoint is using the MIDI 2.0 Protocol, then the Device should have a translator (See Appendix D: Translation: MIDI 1.0 and MIDI 2.0 Messages) for backward compatibility for Function Blocks that represent MIDI 1.0 Byte Stream Ports.

6.2.1 MIDI 1.0 Function Block Design Options

A MIDI 2.0 environment benefits greatly from bidirectional connections between Devices, but MIDI 1.0 Input and Output Ports are not always paired for use in a bidirectional manner.

There are different options for implementation of MIDI 1.0 connections within a UMP environment, depending on the intended design and topology features of the Device. MIDI-CI is more likely to operate successfully using a bidirectional MIDI 1.0 In/Out Pair, as shown in Option 1.

6.2.1.1 Option 1: Function Block for Known MIDI 1.0 In/Out Pair

When the design includes a MIDI In and MIDI Out which are intended to function as a bidirectional pair, the combination of the MIDI In and MIDI Out should be represented in a single Function Block spanning a single Group. This is often the best choice for internal MIDI 1.0 functions which use the MIDI 1.0 Byte Stream Data Format and may also apply to external MIDI I/O Ports in connections intended for bidirectional connections.

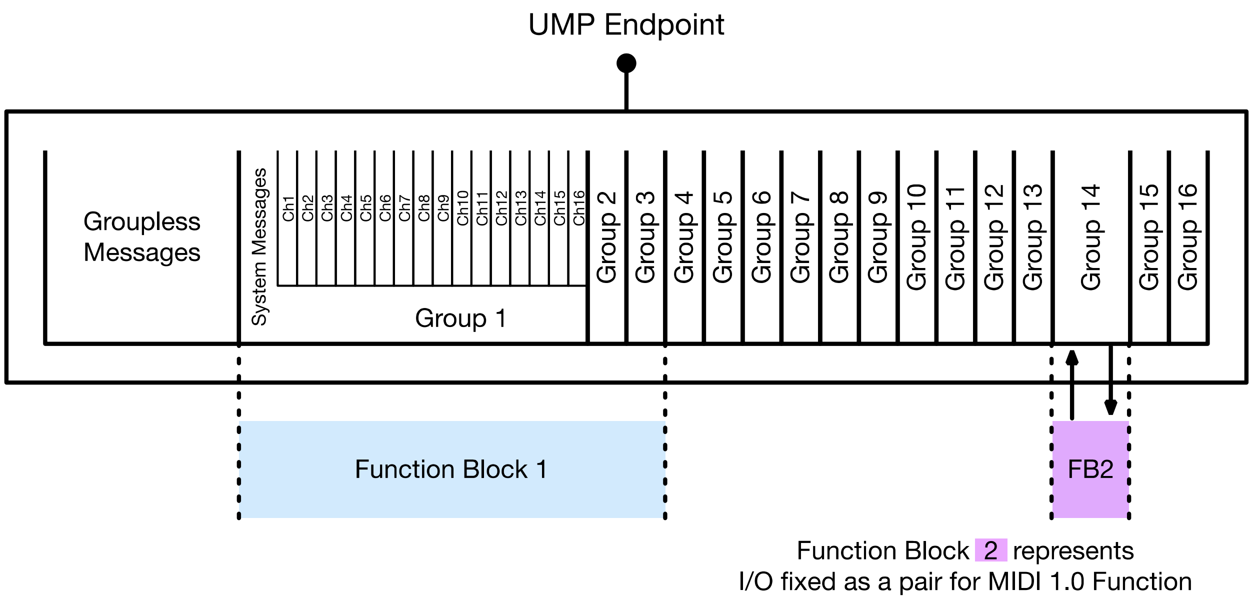


Figure 8: Example Function Block for MIDI 1.0 In/Out Known Pair

6.2.1.2 Option 2: Function Block for Individual MIDI 1.0 Ports

When the design includes MIDI In Ports and MIDI Out Ports which are unlikely to be used as bidirectional pairs, each Port should be represented by a single Function Block spanning a single Group.

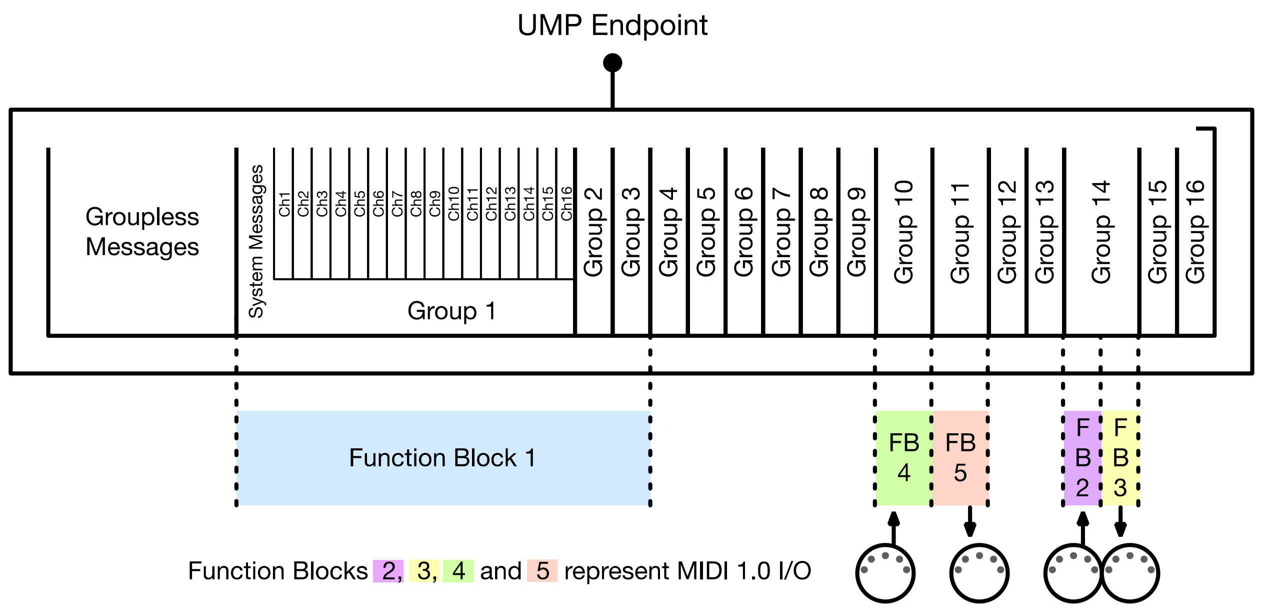


Figure 9 Example Function Blocks for Unpaired MIDI Connections

The Function Block for each Port spans a single Group. Multiple Function Block may be assigned to the same Group if the Device design or user application requires it.

6.2.2 Overlapping Function Blocks with MIDI 1.0 Ports

Section ***6.1.1*** defines that multiple Function Blocks may be assigned to the same Group. The example design in ***Figure 9*** includes two Function Blocks overlapping on a single Group (Group 14).

Function Blocks which represent MIDI 1.0 functions or MIDI 1.0 Ports may overlap on the same Group as other Function Blocks which are not MIDI 1.0. However, if such an overlap is allowed by the Device design, it is up to the Device to handle the required translation.

7 MIDI Messages in UMP Format

This Section defines or reserves all possible MIDI Message formats in the UMP Format:

- Section 7.1 UMP Stream Messages
- Section 7.2 Utility Messages
- Section 7.3 MIDI 1.0 Channel Voice Messages
- Section 7.4 MIDI 2.0 Channel Voice Messages
- Section 7.5 Flex Data Messages
- Section 7.6 System Common and System Real Time Messages
- Section 7.7 System Exclusive (7-Bit) Messages
- Section 7.8 System Exclusive 8 (8-Bit) Messages
- Section 7.9 Mixed Data Set Message

See also:

- Appendix F: All Defined UMP Formats
- Appendix G: All Defined Messages

7.1 UMP Stream Messages

UMP Stream Messages are addressed to the UMP Endpoint, without a Group or Channel assignment. All UMP Stream Messages are 128-bit messages containing the following fields:

- 4 bits Message Type with value 0xF
- 2 bits Format
- 10 bits Status

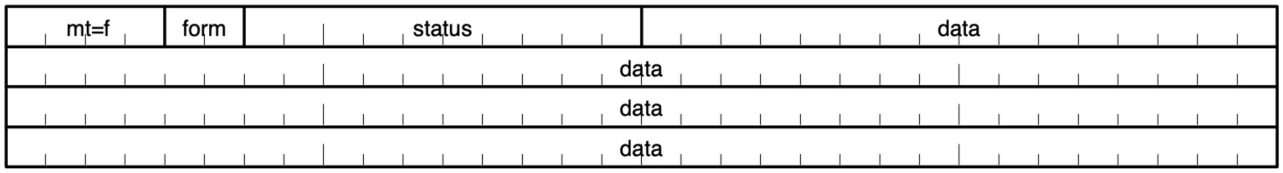


Figure 10 UMP Stream Message General Format

Format

0x0 - Complete message in one UMP

0x1 - Start of a message which spans two or more UMPs

0x2 - Continuing a message which spans three or more UMP. There might be multiple Continue UMPs in a single message

0x3 - End of message which spans two or more UMPs

In this version of the UMP Format and MIDI 2.0 Protocol, UMP Stream Messages allow discovery of UMP Endpoint information and Function Block topology and are used by a Device to send Function Block change notifications. These UMP Stream Messages are used in bidirectional transactions between UMP Endpoints.

7.1.1 Endpoint Discovery Message

A Device may request basic information about the UMP Endpoint connected via the UMP stream.

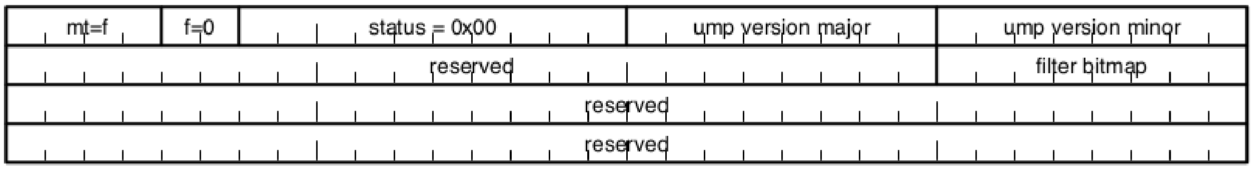


Figure 11 Endpoint Discovery Message Format

Filter

The Filter bitmap is used by the requesting Device to specify the information to be retrieved.

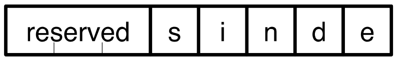


Figure 12 Endpoint Discovery Message Filter Bitmap Field

- e = Requesting an Endpoint Info Notification
- d = Requesting a Device Identity Notification
- n = Requesting an Endpoint Name Notification
- i = Requesting a Product Instance Id Notification
- s = Requesting a Stream Configuration Notification

Each bit set will result in an individual reply.

UMP Version – major, minor version

This indicates what revision of this specification of this Device.

Table 5 UMP Major and Minor Versions

Specification Version	Major Version	Minor Version
1.0	N/A	N/A
1.1	1	1
Future Revisions	TBD	TBD

When receiving an Endpoint Discovery message, the Device shall examine the received UMP Major and Minor Versions to determine response behavior.

- If the received message is of the same version as the Device, then no special considerations are necessary.
- If the received version is higher than the Device's supported version, the Device shall only process the fields defined for its supported version and shall ignore any appended fields and any reserved values and bits.
- If the received version is lower than the Device's supported version, the Device shall only process the fields, values, and bits defined in the received version of UMP.
- This version 1.1 of the M2-104-UM UMP and MIDI 2.0 Protocol specification does not define version number semantics that would indicate breaking changes.

7.1.2 Endpoint Info Notification Message

A UMP Endpoint shall send an Endpoint Info Notification after receiving and in reply to an Endpoint Discovery message with the 'e' bit set in the Filter Bitmap field. A UMP Endpoint should send an Endpoint Info Notification when any property in this message has changed. This provides Information to the receiving UMP connection to understand the other end of the connection.

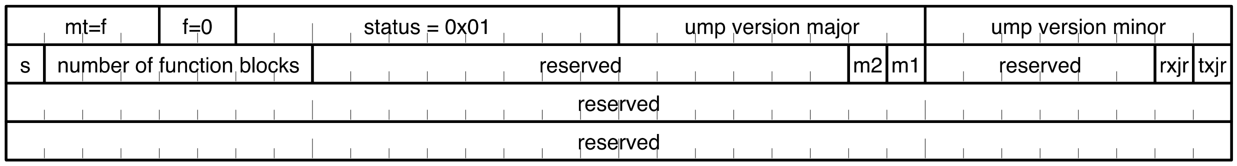


Figure 13 Endpoint Info Notification Message Format

S – Static Function Blocks

If this bit is set high, the UMP Endpoint shall not change any of the properties of its Function Blocks after initial discovery.

Number of Function Blocks (0-32)

Up to 32 Function Blocks can be declared on a UMP Stream. This indicates the existence of Function Blocks that can be discovered using the Discover Function Blocks message.

- 0x00 = No Function Blocks on this UMP Endpoint
- 0x01-0x20 = Number (NFB) of Function Blocks (NFB=1 to 32). Each Function Block is assigned a number, from 0 through NFB-1 as its identifier for Get Function Blocks mechanisms (See Sections *7.1.7 to 7.1.9*).
- 0x21-0x7F = Reserved

The number of Function Blocks on a UMP Endpoint shall not change. A UMP Endpoint may change the number of active Function Blocks by using the mechanisms described in Section *7.1.8*.

UMP Version – major, minor version

This indicates the revision of this specification used by this Device. For this revision of the specification, set UMP Version Major to 0x01 and UMP Minor to 0x01.

M2 – MIDI 2.0 Protocol Capability

UMP Endpoint Supports receiving and sending MIDI 2.0 Protocol (See Section *3.3*)

M1 – MIDI 1.0 Protocol Capability

UMP Endpoint Supports receiving and sending MIDI 1.0 Protocol (See Section *3.2*)

RXJR – Receive JR Timestamp Capability

UMP Endpoint supports receiving JR Timestamps (See Section *4*)

TXJR – Transmit JR Timestamp Capability

UMP Endpoint supports sending JR Timestamps (See Section *4*)

7.1.3 Device Identity Notification Message

A UMP Endpoint may send a Device Identity Notification at any time, or when it receives an Endpoint Discovery message where the Filter field bitmap 'd' bit set.

mt=f	f=0	status = 0x02		reserved			
reserved		r	dev manuf sysex id byte 1	r	dev manuf sysex id byte 2	r	dev manuf sysex id byte 3
r	device family (lsb)	r	device family (msb)	r	dev family model # (lsb)	r	dev family model # (msb)
r	sw revision level byte 1	r	sw revision level byte 2	r	sw revision level byte 3	r	sw revision level byte 4

Figure 14 Device Identity Notification Message Format

Four fields for Device Identification

The four fields described below identify the Device using the same data as defined by the "Device Inquiry" Universal System Exclusive message (See MIDI 1.0 Detailed Specification ***[MA01]***). The data is formatted as follows:

3 bytes Device Manufacturer

This is the System Exclusive ID of the Device manufacturer. For System Exclusive ID values that are only 1 byte in length, the System Exclusive ID value is in the first byte and the remaining 2 bytes are filled with zeroes: ID 00 00

2 bytes Device Family

This identifies the related group of models to which the Device belongs.

2 bytes Device Family Model Number

This identifies a specific model from the Device Manufacturer.

4 bytes Software Revision Level

This is the version number of a Device model number. This is typically the version of software or firmware but may also be the version of hardware.

7.1.4 Endpoint Name Notification

A UMP Endpoint may send an Endpoint Name Notification message at any time, or when it receives an Endpoint Discovery message with the Filter field bitmap 'n' bit set.

mt=f	form	status = 0x03	name byte 1	name byte 2
name byte 3		name byte 4	name byte 5	name byte 6
name byte 7		name byte 8	name byte 9	name byte 10
name byte 11		name byte 12	name byte 13	name byte 14

Repeat Status = 0x03 As Desired to Declare Full Name

Figure 15 Endpoint Name Notification Message Format

UMP Endpoint Name

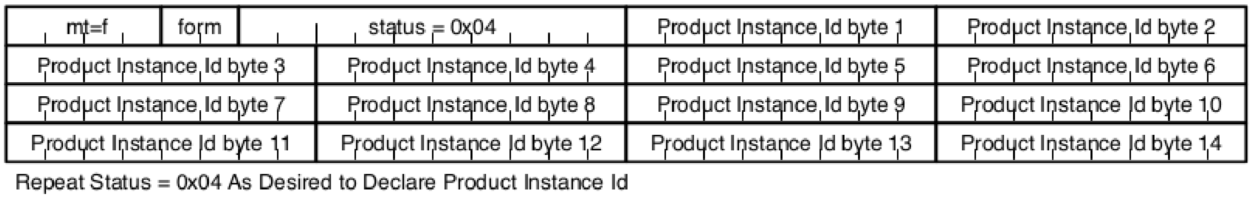
UMP Endpoint name is encoded in UTF-8. The UMP Endpoint name may be contained in a sequence of UMPs. The name shall not be any longer than 98 bytes in size, either in a Complete message (if less than 15 bytes) (Form = 0x0) or comprised of one Start UMP (Form = 0x1), up to five optional Continue UMPs (Form = 0x2) and an End UMP (Form = 0x3).

If the name ends in the middle of a UMP, then the remaining data bytes shall be set to 0x00. This indicates the end of the Name value. This shall only occur in an End UMP (Form = 0x3) or Complete UMP (Form = 0x0).

7.1.5 Product Instance Id Notification Message

Devices should declare a Product Instance Id. Product Instance Id should be, where possible, the same as the Serial Number of the Device and should be a unique number per Manufacturer/Family/Model.

A UMP Endpoint may send out its Product Instance Id at any time, or when it receives an Endpoint Discovery message where the Filter field bitmap 'i' bit set.



mt=f	form	status = 0x04	Product Instance Id byte 1	Product Instance Id byte 2
Product Instance Id byte 3		Product Instance Id byte 4	Product Instance Id byte 5	Product Instance Id byte 6
Product Instance Id byte 7		Product Instance Id byte 8	Product Instance Id byte 9	Product Instance Id byte 10
Product Instance Id byte 11		Product Instance Id byte 12	Product Instance Id byte 13	Product Instance Id byte 14

Repeat Status = 0x04 As Desired to Declare Product Instance Id

Figure 16 Product Instance Id Notification Message Format

Product Instance Id

Product Instance Id shall be ASCII Text in the ordinal range 32-126. The Product Instance Id may be contained in a set of UMPs. The Product Instance Id shall not be any longer than 42 bytes in size, either in a Complete message (if less than 15 bytes) (Form = 0x0) or comprised of a Start message (Form = 0x1), a single Continue message (Form = 0x2) if needed, and an End message (Form = 0x3).

If the Product Instance Id ends in the middle of a MIDI message, then the remaining data bytes shall be set to 0x00. This indicates the end of the Product Instance Id value. This shall only occur in an End message (Form = 0x3) or Complete message (Form = 0x0).

The Product Instance Id can be used to:

- Re-identify a Device after power cycling
- Distinguish multiple Devices of the same model
- Unify multiple UMP Endpoints under one Device

7.1.6 Selecting a MIDI Protocol and Jitter Reduction Timestamps for a UMP Stream

Stream Configuration Request (see Section *7.1.6.2*) is the MIDI standard method for selecting MIDI Protocols and JR Timestamps. Endpoint Discovery message (see Section *7.1.1*) is used to discover the Protocols and JR Timestamps availability. Some devices, interfaces, APIs, or transports might have additional means for discovering or selecting Protocols and JR Timestamps to fit the needs of a particular MIDI system.

These mechanisms to select a Protocol and JR Timestamps replace the Protocol Negotiation mechanisms which were included in initial versions of MIDI-CI, but which have been deprecated from MIDI-CI.

7.1.6.1 Steps to Select Protocol and Jitter Reduction Timestamps

The following describes the process for two Devices to select a Protocol and JR Timestamps. The rules for the messages used are defined in Section 7.

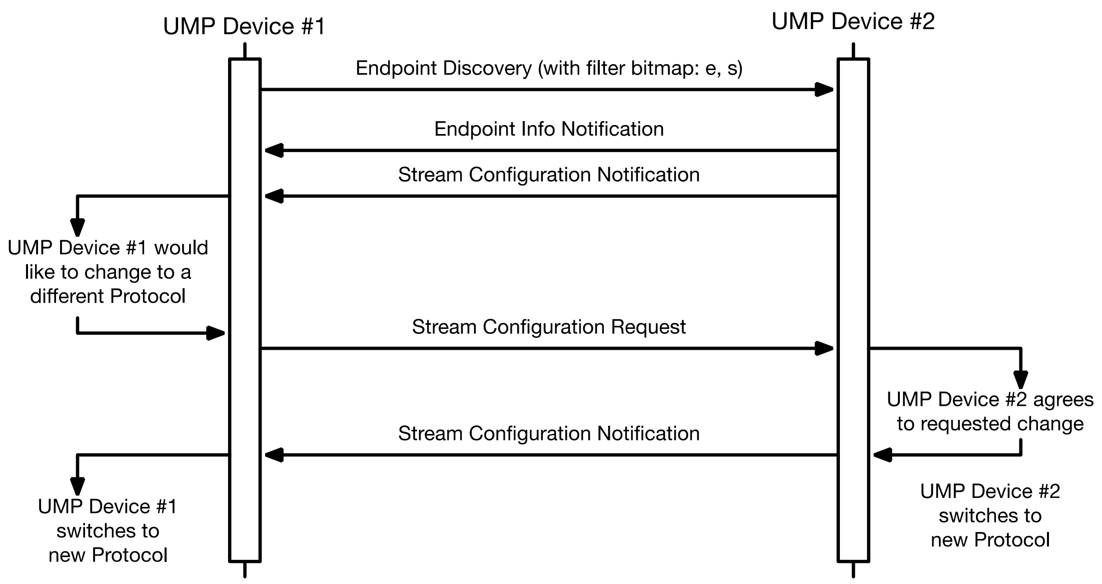

UMP Device #1
UMP Device #2
Endpoint Discovery (with filter bitmap: e, s)
Endpoint Info Notification
Stream Configuration Notification
UMP Device #1 would like to change to a different Protocol
Stream Configuration Request
UMP Device #2 agrees to requested change
Stream Configuration Notification
UMP Device #1 switches to new Protocol
UMP Device #2 switches to new Protocol

Figure 17 Steps to Select Protocol and Jitter Reduction Timestamps

Step 1 Discover Available Features

A Sender sends an Endpoint Discovery message (see Section *7.1.1*) with the Filter field set to retrieve the UMP Endpoint Info and Stream Configuration from a Receiver. The Receiver replies 1) with an Endpoint Info Notification message (see Section *7.1.2*) to declare the features supported for selection, including the list of supported Protocols and whether JR Timestamps are available, and 2) with a Stream Configuration Notification message (see Section *7.1.6.3*) to declare the current Protocol and whether the UMP Endpoint is currently configured to send and/or receive JR Timestamps.

Step 2 Request a Change in Protocol and Jitter Reduction Timestamps

If the Protocol and JR Timestamps information provided in the Stream Configuration Notification message do not match the desired configuration, then the Sender requests a change by sending Stream Configuration Request message (see Section *7.1.6.2*) with the desired Protocol and JR Timestamps configuration.

The Sender does not change its Protocol and JR Timestamps configuration until it has received a Stream Configuration Notification message in reply (See Figure *17*).

Step 3 Receive Notification of Change in Protocol and Jitter Reduction

If a Stream Configuration Request message was sent the Receiver replies with a Stream Configuration Notification message to confirm the requested configuration. If the Receiver is unable to change some or all parts of the configuration, the Receiver replies with its current configuration.

Once the Sender has received the Stream Configuration Notification message it is able to start sending messages using the Protocol and JR Timestamps configuration as declared in the Stream Configuration Notification reply.

7.1.6.2 Stream Configuration Request

Device (A) can request that a UMP Endpoint of a connected Device (B) use a chosen Protocol and JR Timestamps for all messages sent and received.

Device (A) should not use the requested Protocol on its UMP Endpoint until a Stream Configuration Notification message (See Section *7.1.6.3*) has been received as a reply from the connected Device (B).

This mechanism selects a single Protocol, either MIDI 1.0 Protocol or MIDI 2.0 Protocol. All data should be exchanged in the selected Protocol except in the following case: If MIDI 2.0 Protocol is selected but a Function

Block represents (declares) a MIDI 1.0 connection, then MIDI 1.0 Protocol may optionally be used for messages to/from that Function Block

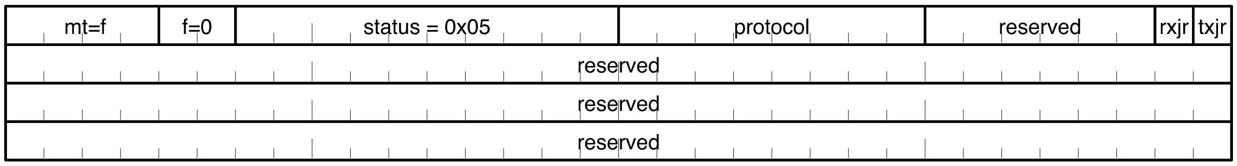


Figure 18 Stream Configuration Request Message Format

Protocol

The Protocol value in this field indicates that the UMP Endpoint sending this message will send its messages using the declared Protocol and expects messages sent to it will also use the same.

0x01 – MIDI 1.0 Protocol

0x02 – MIDI 2.0 Protocol

Note: All other values are reserved.

RXJR – Receive JR Timestamp

If the JR Timestamp Receive value is set in this field, then the UMP Endpoint receiving this message (Device (B)) can expect incoming messages to be prefixed with JR Timestamps.

If Device (A) requests that Device (B) receives JR Timestamps, then Device (A) shall not send JR Timestamps until after a Stream Configuration Notification message (See Section *7.1.6.3*) has been received as a reply from the connected Device (B).

TXJR – Transmit JR Timestamp

If the JR Timestamps Transmit value is set in this field, then the UMP Endpoint receiving this message (Device (B)) shall prefix all messages with JR Timestamps.

If Device (A) requests that Device (B) send JR Timestamps, then Device (A) should not expect to receive JR Timestamps until after a Stream Configuration Notification message (See Section *7.1.6.3*) has been received as a reply from the connected Device (B).

7.1.6.3 Stream Configuration Notification Message

This message declares the current Protocol and JR Timestamps of messages the Device will send and receive.

A UMP Endpoint shall send a Stream Configuration Notification:

- As a reply to an Endpoint Discovery message with the 's' bit set in the Filter Bitmap field.
- As a reply to a Stream Configuration Request message.
- As a notification when any property in this message has changed.

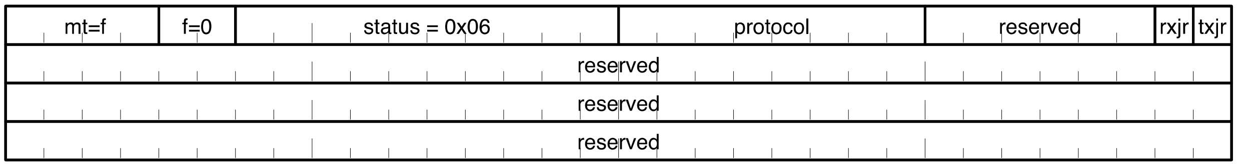


Figure 19 Stream Configuration Notification Message Format

Protocol

The Protocol value in this field indicates that the UMP Endpoint sending this message will send its messages using the declared Protocol and expects messages sent to it will also use the same.

0x01 – MIDI 1.0 Protocol

0x02 – MIDI 2.0 Protocol

Note: All other values are reserved.

RXJR – Receive JR Timestamp

If the JR Timestamp Receive value is set in this field, then the UMP Endpoint sending this message is expecting messages sent to it will include JR Timestamps.

TXJR – Transmit JR Timestamp

If the JR Timestamps Transmit value is set in this field, then the UMP Endpoint sending this message will send all messages with JR Timestamps.

7.1.7 Function Block Discovery Message

This message requests details about the Function Block configuration available on the receiving UMP Endpoint.

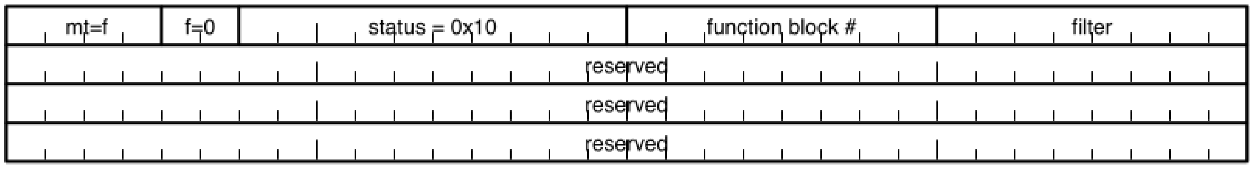


Figure 20 Function Block Discovery Message Format

Function Block

This is the number of the Function Block that the Sender is requesting information about. Endpoint Info Notification message declares the Number of Function Blocks (NFB) (See Section *7.1.2*). Each Function Block is assigned a number, from 0 through NFB-1 as its identifier. Requesting individual Function Blocks should use a value in the range of 0 (0x00) to 31 (0x1F).

Use 0xFF to request information about all Function Blocks.

Example: If Number of Function Blocks = 3 (in the previous Endpoint Info Notification message), then the Function Block Discovery message can be sent with 0x00, 0x01, or 0x02 in the Function Block Number field). Or set the field to 0xFF to retrieve all Function Blocks.

Filter

The Filter bitmap is used by the requesting Device to retrieve Function Block information.

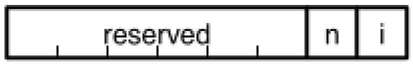


Figure 21 Function Block Discovery Filter Bitmap Field Format

- i = Requesting a Function Block Info Notification
- n = Requesting a Function Block Name Notification

Each bit set will result in an individual reply.

7.1.8 Function Block Info Notification

If a UMP Endpoint has one or more Function Blocks and receives a Function Block Discovery message with the 'i' bit set in the Filter Bitmap field, then the UMP Endpoint shall reply by sending a Function Block Info Notification message. If the Function Block Discovery message has the Function Block Number field set to 0xFF, then the reply shall be one Function Block Info Notification message per Function Block.

A UMP Endpoint which has not declared that its Function Blocks are static should send a Function Block Info Notification when any property in this message has changed. This provides information to the receiving UMP connection to understand the other UMP Endpoint's Function Block configuration.

The requested Function Blocks are declared using a series of MT=0xF UMP with a status=0x11.

mt=f	f=0	status = 0x11	a	function block #	reserv	ui hint	m 1.0	dir
first group		number of groups spanned		midi-ci message version/form	max num of sysex8 streams			
reserved								
reserved								

Figure 22 Function Block Info Notification Message Format

Function Block Active (a)

This declares if this Function Block is currently active. If the bit is set low this indicates that this Function Block is inactive and can be ignored at this time. If the bit is set high this indicates that this Function Block is active and should be processed. This field allows Function Blocks to change their active state without affecting the Function Block number of other Function Blocks.

If the Endpoint Info Notification message has declared that its Function Blocks are static, then the UMP Endpoint shall not have any inactive Function Blocks and all Function Blocks shall have this bit set to high.

Function Block Number

This declares the number (NFB) of the Function Block this reply is describing. The value is in the range of 0 (0x00) to 31 (0x1F).

Direction

This declares the Function Block’s connectivity within the topology of the UMP Endpoint.

0b00 – Reserved

0b01 – Input, Function Block receives MIDI Messages only

0b10 – Output, Function Block transmits MIDI Messages only

0b11 – Bidirectional Connections. Every Input Group member has a matching Output Group.

MIDI 1.0

If the Function Block represents a MIDI 1.0 Port, then declare it here. If this is a MIDI 1.0 Port, then Number of Groups Spanned shall be 1.

0x00 – Not MIDI 1.0

0x01 – Yes - Don't restrict Bandwidth

0x02 – Yes - Restrict Bandwidth to 31.25Kbps

0x03 – Reserved

User Interface Hint

This declares the primary role of the device as a Sender, a Receiver, or both. This is intended to be a hint for applications such as DAWs to present a user with reasonable choices for sources and destinations of MIDI messages.

0b00 – unknown or undeclared

0b01 – Function Block is primarily a Receiver or destination for MIDI messages

0b10 – Function Block is primarily a Sender or source of MIDI messages

0b11 – Function Block is both a Sender and Receiver of MIDI messages

This field does not declare actual connection topology, which is declared by the Direction field. For example, a simple controller keyboard Function Block might be represented in the User Interface Hint field as primarily a Sender while the Direction field declares that the Function Block has bidirectional connection topology because it can respond to a MIDI-CI Discovery message.

Care should be taken not to make nonsensical settings that conflict with the Direction field. For example, a Function Block which declares that Direction is "Output, transmits MIDI Messages only" shall not set the User Interface Hint field to declare that "Function Block is primarily a Receiver".

First Group

Each Function Block declares a set of one or more Groups as members of the Function Block. This value is the number of the Group, 0x0 to 0xF, with the lowest number which is a member of this Function Block.

Number of Groups Spanned

The count of Groups for members of the Function Block. The value is in the range of 1 (0x01) to 16 (0x10).

MIDI-CI Message Version/Format

The version and message format of the MIDI-CI supported. 0x00 = none or unknown. Devices which implement this version 1.1 of the Universal MIDI Packet (UMP) Format shall use version = 0x01 or higher. See MIDI Capability Inquiry (MIDI-CI) specification (version 1.1 or higher) ***[MA02]*** for the value of this field.

Max Number of SysEx 8 Streams

The number of simultaneous SysEx8 Stream IDs supported for this UMP Stream

- 0 = Receiver does not support any System Exclusive 8 messages.
- 1 = Receiver does not support multiple, simultaneous System Exclusive 8 messages.
- 2 – 255 = The number of simultaneous System Exclusive 8 Stream IDs supported

7.1.9 Function Block Name Notification

A UMP Endpoint may send out a Function Block Name Notification when it receives a Function Block Discovery message where the Filter field bitmap 'n' bit set. If the Function Block Discovery message has the Function Block Number field set to 0xFF, then the reply shall be one Function Block Name Notification message per Function Block.

A UMP Endpoint may also send out a Function Block Name Notification at any time if the Endpoint Info Notification message has not declared that its Function Blocks are static. This provides information to the receiving UMP connection to understand the other UMP Endpoint's Function Block name.

The requested Function Blocks are declared using a series of MT=0xF UMP with a status=0x12.

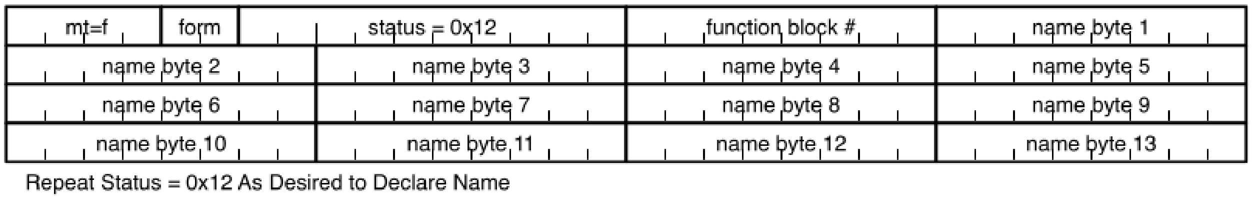


Figure 23 Function Block Name Notification Format

Function Block Name

The Function Block Name is encoded in UTF-8. The name may be contained in a set of UMPs. The name shall not be any longer than 91 bytes in size, either in a Complete message (if less than 14 bytes) (Form = 0x0) or comprised of a Start (Format = 0x1), up to five optional Continue messages (Format = 0x2) and an End message (Format = 0x3).

If the name ends in the middle of a UMP, then the remaining data bytes shall be set to 0x00. This indicates the end of the Name value. This shall only occur in an End UMP (Format = 0x3) or a Complete UMP (Form = 0x0).

Note: When deciding on a name, consideration should be given to many Devices which have a display with a limited number of characters and/or limited character set support.

7.1.10 Start of Clip Message

The Start of Clip message is used in a MIDI Clip File (See ***[MA09]***) as the first event in the Clip Sequence Data. In a MIDI Clip File, the Start of Clip message shall have a preceding Delta Clockstamp. If the Clip Sequence Data is used for musical content, then the timing of the Start of Clip message should be the start of the first bar of music.

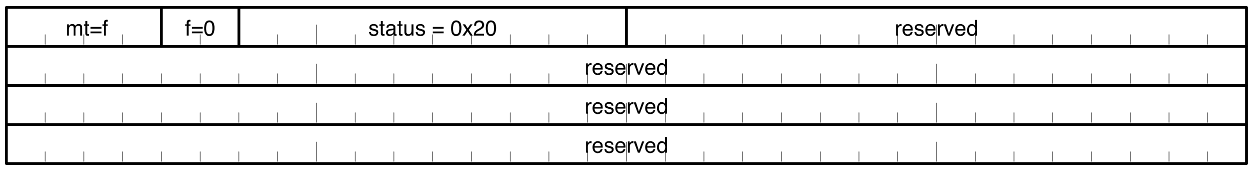


Figure 24 Start of Clip Message Format

7.1.11 End of Clip Message

The last event in the Clip Sequence Data of a MIDI Clip File (See ***[MA09]***) shall be an End of Clip message with a preceding Delta Clockstamp.

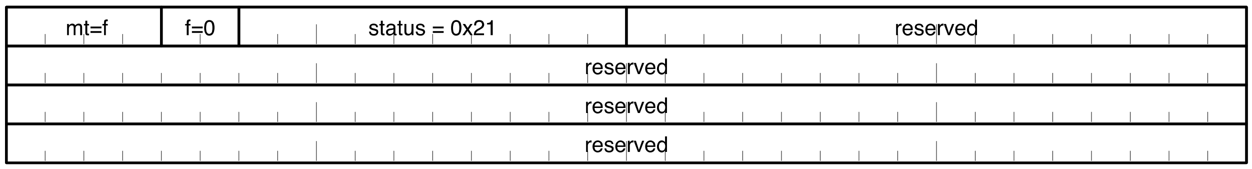


Figure 25 End of Clip Message Format

7.2 Utility Messages

The UMP Format provides a set of Utility Messages. Utility Messages include but are not limited to NOOP and timestamps, and might in the future include UMP transport-related functions.

mt=0	reserved	status	data or reserved

Figure 26 Utility Message General Format

7.2.1 NOOP

A NOOP (no operation) message is provided in the Utility Messages Message Type, using opcode zero.

mt=0	reserved	0	0	0	0	0x00000

Figure 27 NOOP Message Format

7.2.2 Jitter Reduction Timestamps

The UMP Format provides a method of managing jitter for a UMP Stream. All messages from a Sender can be transmitted with a JR Timestamp prepended. Time is based on the Sender's notion of time, declared in JR Clock messages.

The Stream Configuration Request message is used to enable or disable JR Timestamps. See Section *7.1.6.2*.

7.2.2.1 JR Clock Message

The JR Clock message defines the current time of the Sender. The Sender shall send the JR Clock message as close as possible to the time stated in the Time field. The Sender sends independent JR Clock messages, not related to any other message. JR Clock time is monotonically increasing except when it wraps around.

The Sender shall send a JR Clock message at least once every 250 milliseconds. The JR Clock messages will be received with the same jitter as other messages, so the Receiver uses JR Clock messages to discover the jitter characteristics of the connection. The Receiver may use smoothing or averaging of time from each JR Clock message compared to reception time of the JR Clock message UMP to determine a steady JR Clock to render against. Then, the Receiver can also determine a suitable delay, based on the discovered jitter, that shall then be applied to effectively render messages with increased timing accuracy.

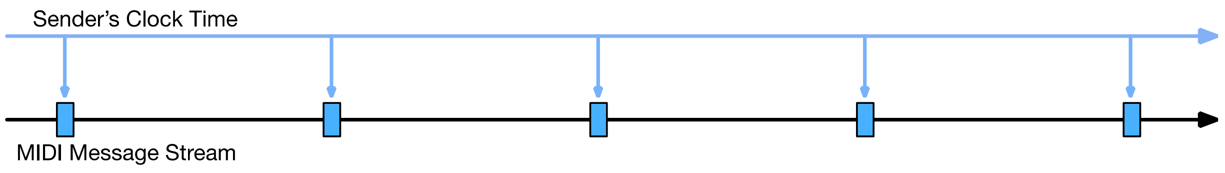


Figure 28 Sequence of JR Clock Messages

A Sender may send additional JR Clock messages with a shorter period to help the Receiver analyze the jitter and calculate the current time. Because the Sender is not mandated to send messages at an exact period (only "at least once every 250 ms" is required), the Receiver should not draw any conclusions from the interval between JR Clock messages.

There is no requirement that Senders or Receivers support the full resolution (of 1/31250 ticks per second accuracy).

mt=0	reserved	0	0	0	1	reserved	sender clock time

Figure 29 JR Clock Message Format

Sender Clock Time

A 16-bit time value in clock ticks of 1/31250 of one second (32 µsec, clock frequency of 1 MHz / 32).

The time value is expected to wrap around every 2.09712 seconds.

To avoid ambiguity of the 2.09712 seconds wrap, and to provide sufficient JR Clock messages for the Receiver, the Sender shall send a JR Clock message at least once every 250 milliseconds.

7.2.2.2 JR Timestamp Message

The JR Timestamp message defines the time of the following message(s). It is a complete message. It is not a part of another message. The timing of every non-JR Timestamp message is set by the most recent preceding JR Timestamp.

A JR Timestamp shall be sent before every non-JR Timestamp message, except in the case of simultaneous messages. If two or more messages are intended to be rendered simultaneously then they can be preceded by a single JR Timestamp. "Simultaneous" in this case is defined as being within the JR Timestamp tick (1/31250 seconds). If a message does not have its own, immediately preceding JR Timestamp, the last received JR Timestamp applies to the message.

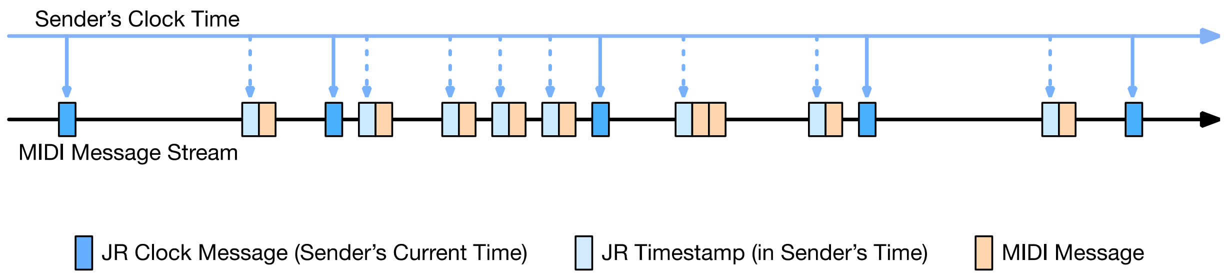

Sender's Clock Time
MIDI Message Stream
JR Clock Message (Sender's Current Time)
JR Timestamp (in Sender's Time)
MIDI Message

Figure 30 Example Sequence of MIDI Messages with JR Timestamps Prepended

JR Timestamps are specified in the Sender's time domain as communicated via JR Clock Messages. For real-time scheduling, the Receiver should convert the time for each message from the Sender's time domain to the Receiver's time domain. The Receiver shall render events at the time referenced against the time of the JR Clock Mechanism described above.

Sender: JR Timestamped messages shall be sent in the order in which they are intended to be rendered.

Receiver: JR Timestamped messages shall be rendered in the order in which they are received.

There is no requirement that Senders or Receivers support the full resolution (of 1/31250 ticks per second accuracy).

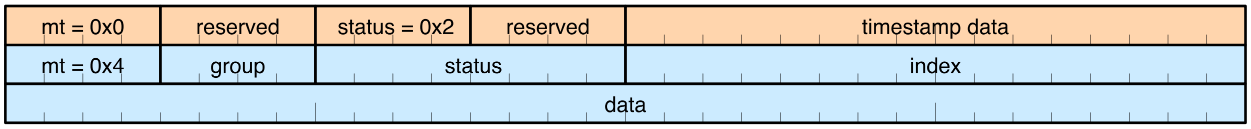
mt=0	reserved	0	0	1	0	reserved	sender clock timestamp

Figure 31 JR Timestamp Message Format

Sender Clock Timestamp

A 16-bit time value in clock ticks of 1/31250 of one second (32 µsec, clock frequency of 1 MHz / 32).

Example 1: JR Timestamped MIDI 2.0 Channel Voice Message (uses 2 Universal MIDI Packets)



Example 2: JR Timestamped System Message (uses 2 Universal MIDI Packets)

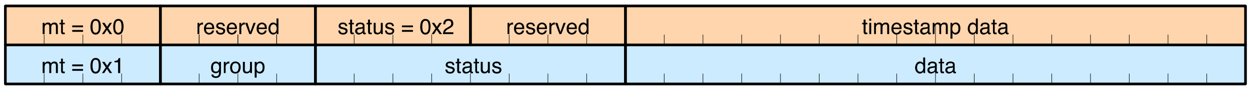


Figure 32 Examples of MIDI Messages with JR Timestamps

7.2.2.3 JR Timestamps and JR Clock Recommended Practice

When a Sender first starts sending JR Clock messages, it could send many of them for a few seconds to help the Receiver measure the jitter on the system.

Receiver Handling of Error Cases

- If a Receiver has not yet received any JR Clock messages but receives other messages, whether with JR Timestamps or not, the Receiver shall render those messages as soon as possible.
- If a Receiver that does not support JR Timestamps receives a JR Timestamp message, it should render the message as soon as possible and send a Stream Configuration Request message to switch the Sender to a protocol without JR Timestamps. See Section *7.1.6.2*.

7.2.3 Delta Clockstamp

A Delta Clockstamp mechanism is used in a Standard MIDI File to declare precise timing of events in a sequence. A Delta Clockstamp Ticks Per Quarter Note sets the timing resolution and accuracy. Then Delta Clockstamp messages declare the number of ticks since the last event. The timing of every non-Clockstamp message is set by the most recent preceding Delta Clockstamp.

7.2.3.1 Delta Clockstamp Ticks Per Quarter Note (DCTPQ)

The Delta Clockstamp Ticks Per Quarter Note message declares the unit of measure used by Delta Clockstamp messages in a MIDI Clip File. If this message is used outside of a MIDI Clip File (is sent on a UMP transport), most receivers will ignore this message.

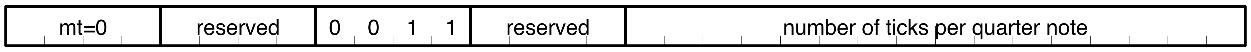


Figure 33 Delta Clockstamp Ticks Per Quarter Note Message Format

The Number of Ticks Per Quarter Note field may have a value of 1 – 65,535 (0 = Reserved).

The accuracy of timing when performing a MIDI Clip File is determined by the combination of the DCTPQ and the current tempo, as shown in the following table of examples.

Table 6 DCTPQ Timing Accuracy in Milliseconds

Beats Per Minute	**Number of Ticks Per Quarter Note (MIDI Clock = 24 TPQ)**					
	1	**24**	**96**	**480**	**960**	**65,535**
20	3000.000	125.000	31.250	6.250	3.125	0.0458
60	1000.000	41.667	10.417	2.083	1.042	0.0153
90	666.667	27.778	6.944	1.389	0.694	0.0102
120	500.000	20.833	5.208	1.042	0.521	0.0076
180	333.333	13.889	3.472	0.694	0.347	0.0051

Note: As a comparison, JR Timestamps used on a UMP transport provide timing accuracy of 0.032 milliseconds.

Note: Some applications, such as industrial control applications, may wish to use a wall clock time base. Such applications might set Tempo to 60 beats per minute. Then Number of Ticks Per Quarter Note is equal to the number of ticks per second and this message allows selection of accuracy required for the target application as low as 15.3μs.

7.2.3.2 Delta Clockstamp (DC): Ticks Since Last Event

The Delta Clockstamp message declares the time of all following messages which occur before the next Delta Clockstamp message. If this message is used outside of a MIDI Clip File (is sent on a UMP transport), most receivers will ignore this message.

The timing of every message (other than Delta Clockstamps) in a MIDI Clip File is set by the most recent preceding Delta Clockstamp. Simultaneous events may share a single Delta Clockstamp. The order of simultaneous events can be critical. Therefore, events shall always be stored and transmitted in presentation order.

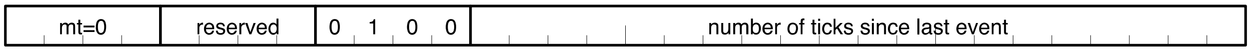


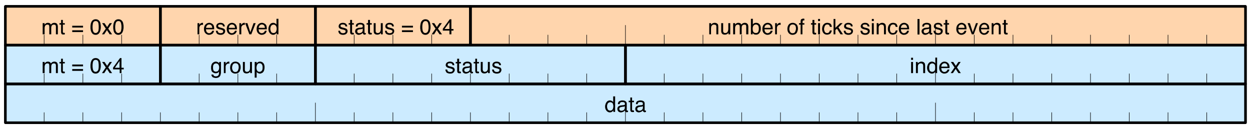
Figure 34 Delta Clockstamp Message Format

If no MIDI message has occurred during the previous 1,048,575 ticks, then the application which creates the MIDI Clip File shall insert a Delta Clockstamp followed by a NOOP message to restart the delta time count. Then the next Delta Clockstamp in the file declares the ticks since the previous NOOP message.

Table 7 Maximum Times of Selected Ticks Per Quarter Note Values at Selected Tempos

		Number of Ticks Per Quarter Note (MIDI Clock = 24 TPQ)					
		1	**24**	**96**	**480**	**960**	**65,535**
	Max Number of Quarter Notes	1048575	43691	10923	2185	1092	16
Beats Per Minute	**Seconds Per Beat**	**Maximum Time Addressable at BPM Tempo**					
20	**3s 0ms**	873h 49m	36h 25m	9h 6m	1h 49m 14s	54m 37s	48s 1ms
60	**1s 0ms**	291h 16m	12h 8m	3h 2m	0h 36m 25s	18m 12s	16s 0ms
90	**0s 667ms**	194h 17m	8h 6m	2h 1m	0h 24m 17s	12m 9s	10s 672ms
120	**0s 500ms**	145h 38m	6h 4m	1h 31m	0h 18m 12s	9m 6s	8s 0ms
180	**0s 333ms**	97h 0m	4h 2m	1h 1m	0h 12m 7s	6m 4s	5s 328ms

Example 1: Delta Clockstamped MIDI 2.0 Channel Voice Message (uses 2 Universal MIDI Packets)



Example 2: Delta Clockstamped System Message (uses 2 Universal MIDI Packets)

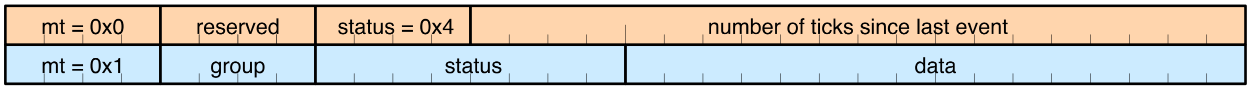


Figure 35 Examples of MIDI Messages with Delta Clockstamp

7.3 MIDI 1.0 Channel Voice Messages

In UMP, the MIDI 1.0 Channel Voice Messages are all 32-bit messages containing the following data:

- 4 bits Message Type with value 0x2
- 4 bits Group
- 24 bits of MIDI 1.0 Channel Voice Message data:
  - 8 bits Status that includes a 4-bit opcode and a 4-bit Channel number
  - 16 bits index, data, and/or reserved space

Per ***Figure 36***, for 3-byte MIDI 1.0 Channel Voice Messages, all three bytes are copied into bytes 2 through 4 of the UMP. This applies to the Note Off, Note On, Poly Pressure, Control Change, and Pitch Bend messages.

Per ***Figure 37*** for 2-byte MIDI 1.0 Channel Voice Messages, the two bytes are copied into bytes 2 and 3 of the UMP, and byte 4 is filled with 0 bits. This applies to the Program Change and Channel Pressure messages.

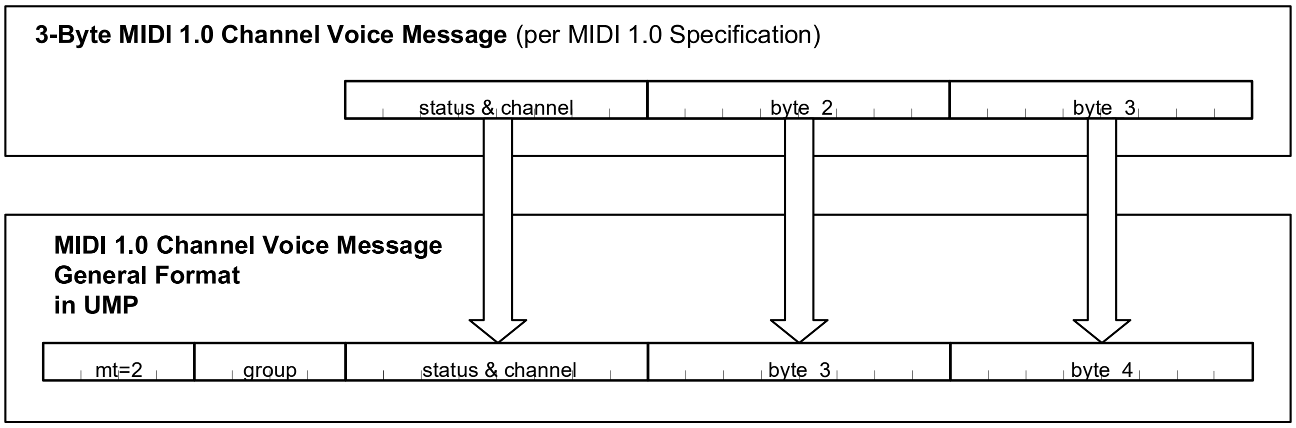

3-Byte MIDI 1.0 Channel Voice Message (per MIDI 1.0 Specification)
status & channel
byte 2
byte 3
MIDI 1.0 Channel Voice Message General Format in UMP
mt=2
group
status & channel
byte 3
byte 4

Figure 36 MIDI 1.0 3-Byte Channel Voice Message General Format

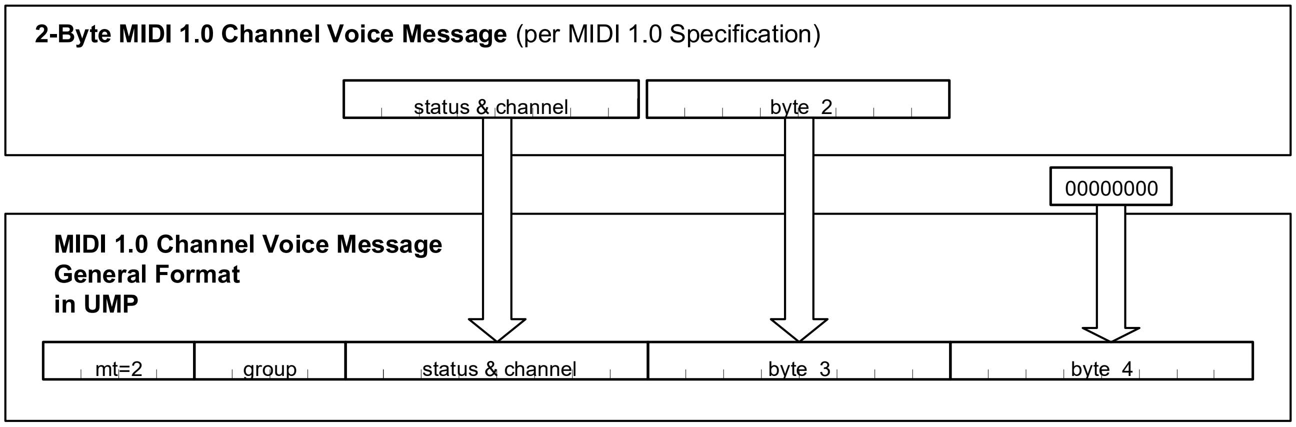

2-Byte MIDI 1.0 Channel Voice Message (per MIDI 1.0 Specification)
status & channel
byte 2
00000000
MIDI 1.0 Channel Voice Message General Format in UMP
mt=2
group
status & channel
byte 3
byte 4

Figure 37 MIDI 1.0 2-Byte Channel Voice Message General Format

7.3.1 MIDI 1.0 Note Off Message

For fundamental functions of Note Off see the MIDI 1.0 Specification ***[MA01]***.

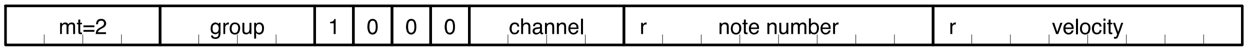


Figure 38 MIDI 1.0 Note Off Message

7.3.2 MIDI 1.0 Note On Message

For fundamental functions of Note On see the MIDI 1.0 Specification ***[MA01]***.

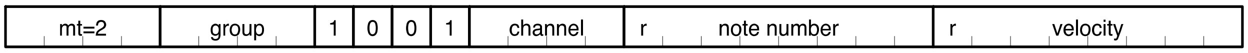


Figure 39 MIDI 1.0 Note On Message

7.3.3 MIDI 1.0 Poly Pressure Message

For fundamental functions of Poly Pressure (Polyphonic Aftertouch) see the MIDI 1.0 Specification ***[MA01]***.

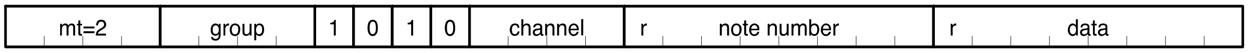


Figure 40 MIDI 1.0 Poly Pressure Message

7.3.4 MIDI 1.0 Control Change Message

For fundamental functions of Control Change see the MIDI 1.0 Specification ***[MA01]***.

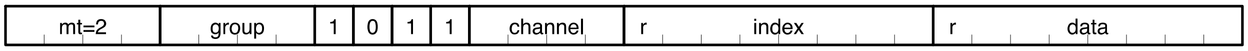


Figure 41 MIDI 1.0 Control Change Message

7.3.5 MIDI 1.0 Program Change Message

For fundamental functions of Program Change see the MIDI 1.0 Specification ***[MA01]***.

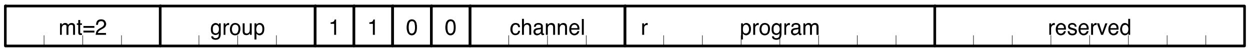


Figure 42 MIDI 1.0 Program Change Message

7.3.6 MIDI 1.0 Channel Pressure Message

For fundamental functions of Channel Pressure (Channel Aftertouch) see the MIDI 1.0 Specification ***[MA01]***.

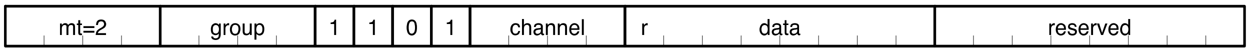


Figure 43 MIDI 1.0 Channel Pressure Message

7.3.7 MIDI 1.0 Pitch Bend Message

For fundamental functions of Pitch Bend see the MIDI 1.0 Specification ***[MA01]***.

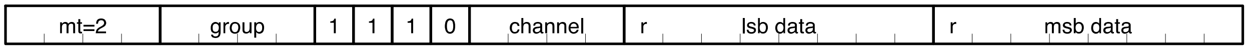


Figure 44 MIDI 1.0 Pitch Bend Message

7.4 MIDI 2.0 Channel Voice Messages

All MIDI 2.0 Channel Voice Messages are 64-bit messages containing the following fields:

- 4 bits Message Type with value 0x4
- 4 bits Group
- 8 bits Status that includes a 4-bit opcode and a 4-bit Channel number
- 16 bits Index
- 32 bits Data containing parameter/property value(s)

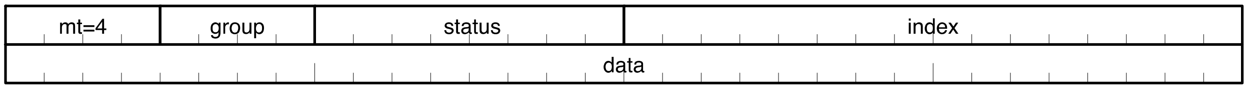


Figure 45 MIDI 2.0 Channel Voice Message General Format

Devices that use any of these MIDI 2.0 Channel Voice Messages from Message Type 0x4 in a Group shall not use any of the MIDI 1.0 Channel Voice Messages from Message Type 0x2 within that same Group.

7.4.1 MIDI 2.0 Note Off Message

For fundamental functions of Note Off see the MIDI 1.0 Specification ***[MA01]***.

The MIDI 2.0 Protocol expands the Note Off message with higher resolution Velocity, and with added Attribute Type and Attribute Data fields.

For more information, see:

Section *7.4.2* MIDI 2.0 Note On Message

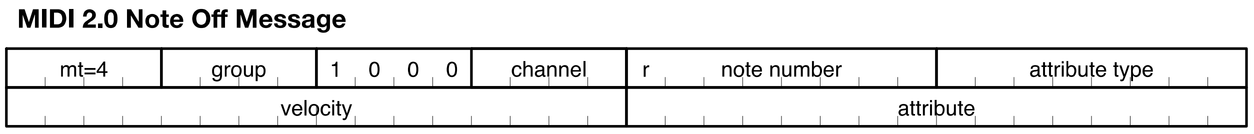


Figure 46 MIDI 2.0 Note Off Message

Attribute (Attribute Type & Attribute Data)

The Attribute Data contains properties for the Note Off. The type of data contained in the Attribute Data field is declared by the Attribute Type field. For more information, see:

Section *7.4.14* MIDI 2.0 Note On/Off: Attribute Type & Attribute Data

Section *7.4.15* MIDI 2.0 Notes and Pitch

7.4.2 MIDI 2.0 Note On Message

For fundamental functions of Note On see the MIDI 1.0 Specification ***[MA01]***.

The MIDI 2.0 Protocol expands the Note On message with higher resolution Velocity, and with added Attribute Type and Attribute Data fields.

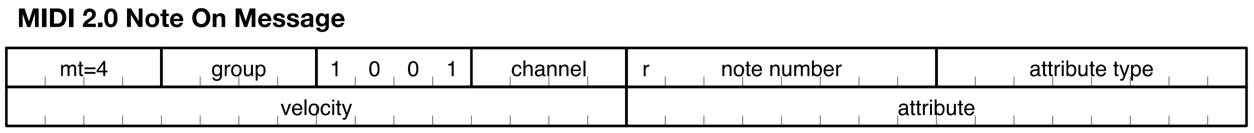


Figure 47 MIDI 2.0 Note On Message

Velocity

The allowable Velocity range for a MIDI 2.0 Note On message is 0x0000-0xFFFF. Unlike the MIDI 1.0 Note On message, a velocity value of zero does not function as a Note Off. When translating a MIDI 2.0 Note On message to the MIDI 1.0 Protocol, if the translated MIDI 1.0 value of the Velocity is zero, then the Translator shall replace the zero with a value of 1.

Attribute (attribute type & attribute data)

The Attribute Data contains properties for the Note On. The type of data contained in the Attribute Data field is declared by the Attribute Type field. For more information, see:

Section *7.4.14* MIDI 2.0 Note On/Off: Attribute Type & Attribute Data

Section *7.4.15* MIDI 2.0 Notes and Pitch

7.4.3 MIDI 2.0 Poly Pressure Message

For fundamental functions of Poly Pressure (Polyphonic Aftertouch) see the MIDI 1.0 Specification ***[MA01]***.

The MIDI 2.0 Protocol expands the resolution of the Poly Pressure message from 7 bits to 32 bits.

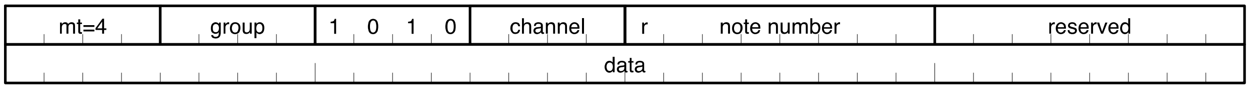


Figure 48 MIDI 2.0 Poly Pressure Message

7.4.4 MIDI 2.0 Registered Per-Note Controller and Assignable Per-Note Controller Messages

The MIDI 2.0 Protocol introduces these new messages with 256 Registered Per-Note Controllers and 256 Assignable Per-Note Controllers:

- **Registered Per-Note Controllers** have specific functions defined by MMA/AMEI specifications. Currently defined Registered Per-Note Controllers are listed in ***Appendix A***:MIDI 2.0 Registered Per-Note Controllers.

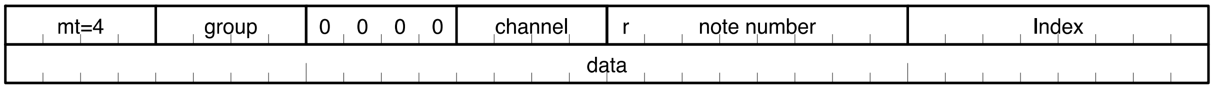


Figure 49 MIDI 2.0 Registered Per-Note Controller Message

Note: Registered Per-Note Controller numbers that have no definition are Reserved and shall not be used until they are defined by MMA/AMEI.

- **Assignable Per-Note Controllers** have no pre-defined function and are available for any device-specific or application-specific function.

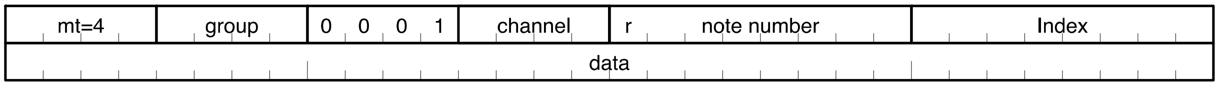


Figure 50 MIDI 2.0 Assignable Per-Note Controller Message

7.4.5 MIDI 2.0 Per-Note Management Message

The MIDI 2.0 Protocol introduces a Per-Note Management message to enable independent control from Per-Note Controllers to multiple Notes on the same Note Number.

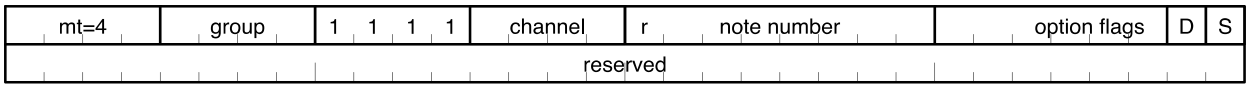


Figure 51 MIDI 2.0 Per-Note Management Message

Option Flags

When bits are set high, specific functions of the Per-Note Management message are active:

D: Detach Per-Note Controllers from previously received Note(s)

S: Reset (Set) Per-Note Controllers to default values

When a device receives a Per-Note Management message with D = 1 (Detach), all currently playing notes and previous notes on the referenced Note Number shall no longer respond to any Per-Note Controllers. Currently playing notes shall maintain the current values for all Per-Note Controllers until the end of the note life cycle.

When a device receives a Per-Note Management message with S = 1, all Per-Note Controllers on the referenced Note Number should be reset to their default values.

When a device receives a Per-Note Management message with D = 1 and S = 1, then the device should first process the Detach function, and then perform the Reset function. As a result, currently playing notes on the referenced Note Number maintain the current values for all Per-Note Controllers until the end of the note life cycle. The default value and any further changes to Per-Note Controllers shall apply to future notes only.

A Per-Note Management Message with D=0 and S=0 has no defined function.

Note: The above defined responses to Per-Note Management messages apply by default to all Per Note Controllers. Future AMEI/MMA specifications might define other responses for specific Per Note Controllers. For example, a Profile might define different responses for particular Per-Note Controllers used for specific applications.

See ***Appendix C: Using MIDI 2.0 Per-Note Messages*** for implementation guidelines.

7.4.6 MIDI 2.0 Control Change Message

For fundamental functions of Control Change see the MIDI 1.0 Specification ***[MA01]***.

The MIDI 2.0 Protocol expands the resolution of the Control Change message from 7 bits to 32 bits.

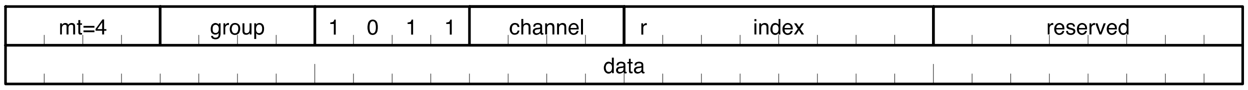


Figure 52 MIDI 2.0 Control Change Message

Note: The MIDI 1.0 Specification defines Control Change indexes 98, 99, 100, and 101 (0x62, 0x63, 0x64, and 0x65) to be used as compound sequences for Non-Registered Parameter Number and Registered Parameter Number control messages. These set destinations for Control Change index 6/38 (0x06/0x26), Data Entry. The MIDI 2.0 Protocol replaces those compound sequences with unified messages, see Section 7.4.7 MIDI 2.0 Registered Controller (RPN) and Assignable Controller (NRPN) Messages.

Note: The MIDI 1.0 Specification defines Control Change indexes 0 and 32 (0x00 and 0x20) to be used as Bank Select associated with following Program Change messages. The MIDI 2.0 Protocol replaces those compound sequences with unified messages, see Section 7.4.9 MIDI 2.0 Program Change Message.

Implementation Recommendations

Devices sending the MIDI 2.0 Protocol should not transmit Control Change messages with indexes of 6, 38, 98, 99, 100, or 101. Instead, they should transmit the new Assignable Controller messages and Registered Controller messages (see Section *7.4.7*). These new messages are more friendly to send, to receive, and to edit in a sequencer.

- Devices sending the MIDI 2.0 Protocol should not transmit Control Change messages with indexes of 0 and 32. Instead they should transmit the new MIDI 2.0 Program Change message (see Section *7.4.9*).
- Devices receiving the MIDI 2.0 Protocol should ignore Control Change messages with indexes of 0, 6, 32, 38, 98, 99, 100, and 101.
- In MIDI 2.0 Protocol, Control Change 88 shall not be used for High Resolution Velocity. The Note On 16 bit Velocity value has a higher range than the MIDI 1.0 High Resolution Velocity controller and Note On combined.

7.4.6.1 Special Control Change Formats and Values

There are several Control Change Messages that do not describe a range and instead declare a value.

CC 84 Portamento

When a Note-On is received after a Portamento Control message, the voice's pitch will glide from the pitch of a note specified in the Portamento Control message to the new Note-On's pitch. For fundamental functions of Portamento Control Change see the MIDI 1.0 Specification ***[MA01]***.

Portamento Control communicates which note number the subsequent note is gliding from. This note number can be found in the most significant 7 bits of the 32bit Data field. The least significant 25 bits are undefined and shall be ignored.

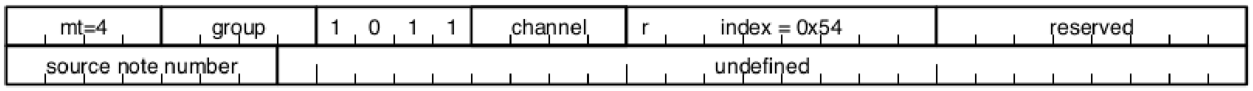


Figure 53 MIDI 2.0 Portamento Control Change Message

CC 126 – Omni-Off/Mono Message

When Mono mode is selected, a single voice is assigned per MIDI Channel. The Omni-Off/Mono Message specifies the number of channels in which Monophonic Voice messages are to be sent. For fundamental functions of Omni-Off/Mono Control Change message see the MIDI 1.0 Specification ***[MA01]***.

The number of Channels to be used is declared in the most significant 7 bits of the 32bit Data field. The least significant 25 bits are undefined and shall be ignored.

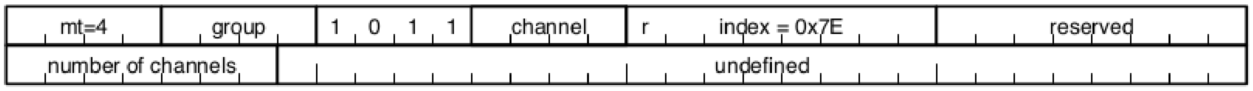


Figure 54 MIDI 2.0 Omni-Off/Mono Control Change Message

7.4.7 MIDI 2.0 Registered Controller (RPN) and Assignable Controller (NRPN) Messages

The MIDI 2.0 protocol introduces 16,384 Registered Controllers and 16,384 Assignable Controllers.

- Registered Controllers have specific functions defined by MMA/AMEI specifications. Registered Controllers map and translate directly to MIDI 1.0 Registered Parameter Numbers (RPN, see ***D.2.3***) and use the same definitions as MMA/AMEI approved RPN messages. Registered Controllers are organized in 128 Banks (corresponds to RPN MSB), with 128 controllers per Bank (corresponds to RPN LSB).

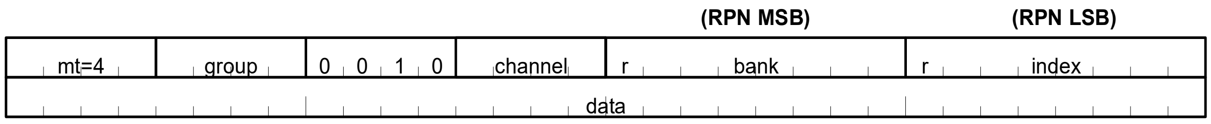


Figure 55 MIDI 2.0 Registered Controller Message

- Assignable Controllers have no specific function and are available for any device or application-specific function. Assignable Controllers map and translate directly to MIDI 1.0 Non-Registered Parameter Numbers (NRPN). Assignable Controllers are also organized in 128 Banks (corresponds to NRPN MSB), with 128 controllers per Bank (corresponds to NRPN LSB).

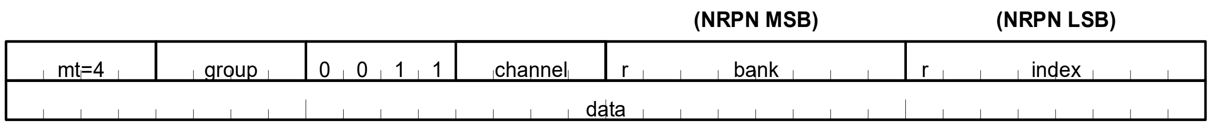


Figure 56 MIDI 2.0 Assignable Controller Message

In the MIDI 1.0 Protocol, creating and editing RPNs and NRPNs requires the use of compound (multiple) MIDI messages, which can be confusing for both developers and users. In the MIDI 2.0 Protocol, Registered Controllers and Assignable Controllers replace those compound messages with a single, unified message, making them much easier to use.

7.4.7.1 Registered Controller Formats and Values

There are several RPN's that do not describe a range and instead declare one or more values.

RPN 0x0000 – Pitch Bend Range

Pitch Bend Range uses the RPN value to set the pitch sensitivity in HCUs and cents. For fundamental functions Pitch Bend Range message see the MIDI 1.0 Specification *[MA01]*.

The HCUs are declared in the most significant 7 bits of the 32bit Data Field and cents in the next 7 bits. The least significant 18 bits are undefined and shall be ignored.

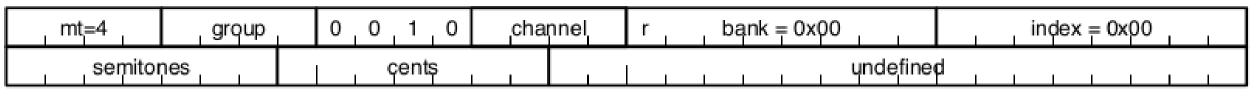


Figure 57 MIDI 2.0 Pitch Bend Sensitivity Message

RPN 0x0002 – Coarse Tuning

Coarse Tuning uses the RPN value to set the tuning. For fundamental functions Coarse Tuning message see the MIDI 1.0 Specification *[MA01]*.

The Coarse Tuning is declared in the most significant 7 bits of the 32bit Data Field. The least significant 25 bits are undefined and shall be ignored.

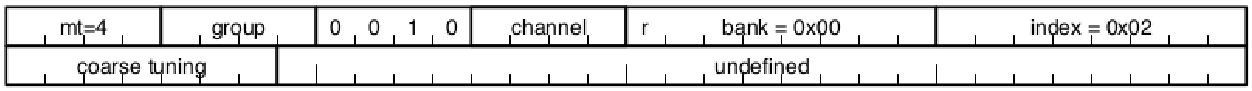


Figure 58 MIDI 2.0 Coarse Tuning Message

RPN 0x0003 – Tuning Program Change

Tuning Program Change uses the RPN value to select a Tuning Program. For fundamental functions Tuning Program Change message see the MIDI 1.0 Specification *[MA01]*.

The Tuning Program number is declared in the most significant 7 bits of the 32bit Data Field. The least significant 25 bits are undefined and shall be ignored.

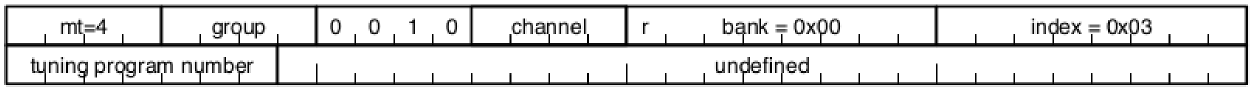


Figure 59 MIDI 2.0 Tuning Program Change Message

RPN 0x0004 – Tuning Bank Select

Tuning Bank Select uses the RPN value to select a Tuning Bank. For fundamental functions Tuning Bank Select message see the MIDI 1.0 Specification *[MA01]*.

The Tuning Bank number is declared in the most significant 7 bits of the 32bit Data Field. The least significant 25 bits are undefined and shall be ignored.

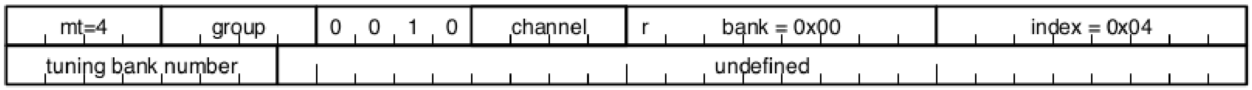


Figure 60 MIDI 2.0 Tuning Bank Change Message

RPN 0x0006 – MPE MCM

MPE MCM uses the RPN Value to declare the number of Channels used for a Lower or Upper Zone. For fundamental functions of MPE MCM message see the MPE Specification *[MA07]*.

The number of Channels to be used is declared in the most significant 7 bits of the 32bit Data field. The least significant 25 bits are undefined and shall be ignored.

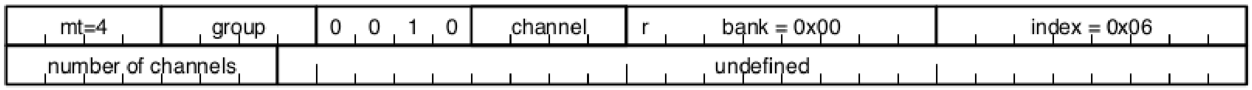


Figure 61 MIDI 2.0 MPE MCM Message

7.4.8 MIDI 2.0 Relative Registered Controller (RPN) and Assignable Controller (NRPN) Messages

Registered Controller Messages and Assignable Controller Messages (defined above in Section *7.4.7*) directly set the values of the destination properties. With the MIDI 2.0 Protocol's Relative Registered Controller and Relative Assignable Controller Messages, it is now also possible to make relative increases or decreases to the current values of those same properties.

These new messages act upon the same address space as the MIDI 2.0 Protocol's Registered Controllers and MIDI 2.0 Assignable Controllers, and use the same controller Banks. However, these Relative controllers cannot be translated to the MIDI 1.0 Protocol.

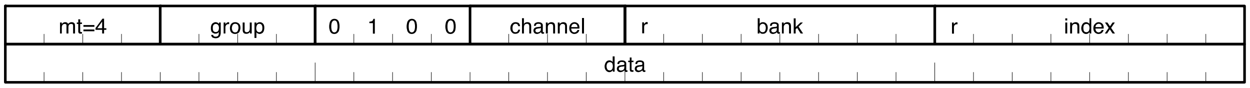


Figure 62 MIDI 2.0 Relative Registered Controller Message

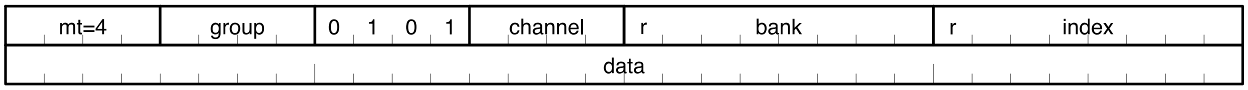


Figure 63 MIDI 2.0 Relative Assignable Controller Message

Data

The data field in the MIDI 2.0 Relative Registered Controller and Relative Assignable Controller messages contains a Two's Complement value, to provide negative and positive relative control of the destination value.

7.4.9 MIDI 2.0 Program Change Message

For fundamental functions of Program Change and Bank Select see the MIDI 1.0 Specification ***[MA01]***.

In the MIDI 2.0 Protocol, this message combines the MIDI 1.0 Protocol's separate Program Change and Bank Select messages into a single, unified message; by contrast, the MIDI 1.0 Protocol mechanism for selecting Banks and Programs requires sending three MIDI separate 1.0 Messages. The MIDI 1.0 Protocol's existing 16,384 Banks, each with 128 Programs, are preserved and translate directly to the MIDI 2.0 Protocol.

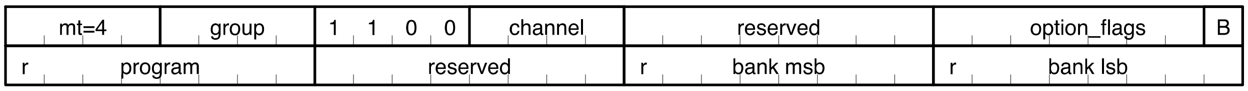


Figure 64 MIDI 2.0 Program Change Message

The MIDI 2.0 Program Change message always selects a Program. The Bank Select operation is optional, controlled by the Bank Valid bit (B):

- If the Sender sets the Bank Valid bit to 0, then the Receiver performs only the Program Change, without selecting a new Bank (i.e., the Receiver keeps its currently selected Bank). In this case, the Sender shall also fill the Bank MSB and Bank LSB fields with zeroes.
- If the Sender sets the Bank Valid bit to 1, then the Receiver performs first the Bank Select operation and then the Program Change operation.
- Other option flags not defined in this specification are Reserved and shall be set to zero.

7.4.10 MIDI 2.0 Channel Pressure Message

For fundamental functions of Channel Pressure (Channel Aftertouch) see the MIDI 1.0 Specification ***[MA01]***.

The MIDI 2.0 Protocol expands the resolution of the Channel Pressure message from 7 bits to 32 bits.

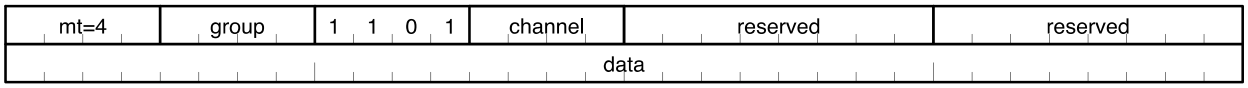


Figure 65 MIDI 2.0 Channel Pressure Message

7.4.11 MIDI 2.0 Pitch Bend Message

For fundamental functions of Pitch Bend see the MIDI 1.0 Specification ***[MA01]***.

The MIDI 2.0 Protocol expands the resolution of the Pitch Bend message from 14 bits to 32 bits. The data field is an unsigned bipolar value, centered at 0x80000000.

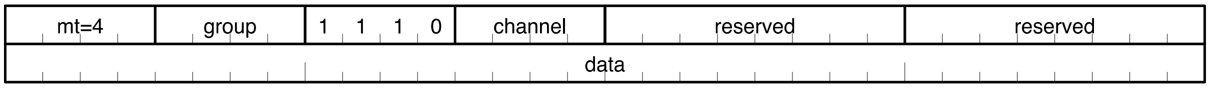


Figure 66 MIDI 2.0 Pitch Bend Message

7.4.12 MIDI 2.0 Per-Note Pitch Bend Message

The MIDI 2.0 Per-Note Pitch Bend message acts like Pitch Bend in every way, except that it applies to individual Note Numbers. The data field is an unsigned bipolar value, centered at 0x80000000.

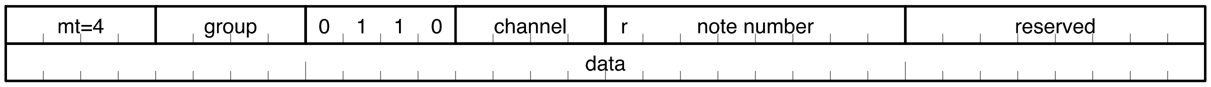


Figure 67 MIDI 2.0 Per-Note Pitch Bend Message

7.4.13 Registered Controller (RPN) for Sensitivity of Per-Note Pitch Bend

This Registered Controller defines how to communicate the intended amount of pitch change which will be controllable by subsequent Per-Note Pitch Bend messages. The selected amount of controllable pitch bend range is common to all Note Numbers on the selected MIDI Channel.

This Registered Controller translates to an equivalent Registered Parameter Number (RPN) in the MIDI 1.0 Protocol. However, this RPN has no function within the MIDI 1.0 Protocol.

7.4.13.1 Registered Controller Bank 0, Index 7 (RPN #00/07)

The Registered Controller for Sensitivity of Per-Note Pitch Bend #00/07 sets the controllable pitch range (up and down) from the current sounding pitch of a Note when using MIDI 2.0 Per-Note Pitch Bend messages. All Notes which follow this message should respond to Per-Note Pitch Bend messages within the pitch range set by the sensitivity value.

The selected sensitivity value is common to all Note Numbers on the selected MIDI Channel.

Note: "Per-Note" in the name of this message relates specifically to the "Per-Note Pitch Bend", not to the sensitivity which is set via this message. The selected sensitivity is shared by all Note Numbers.

MIDI 2.0 Registered Controller for Sensitivity of Per-Note Pitch Bend

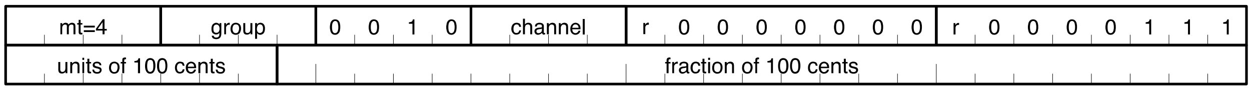


Figure 68 RC for Sensitivity of Per-Note Pitch Bend Message

MIDI 2.0 Per-Note Pitch Bend value is an unsigned bipolar value, centered at 0x80000000. The value of the Registered Controller for Sensitivity of Per-Note Pitch Bend sets the range of pitch bend down (Per-Note Pitch Bend value from 0x80000000 to 0x00000000) and an equal range of pitch bend up (Per-Note Pitch Bend value from 0x80000000 to 0xFFFFFFFF).

The sensitivity value of the Registered Controller for Sensitivity of Per-Note Pitch Bend is expressed as a 7.25 fixed-point unsigned value that specifies an interval in units of 100 Cents. The integer part is the number of 100-Cent units. The fractional part is a fraction of one 100 Cents.

Note: The data format of this value is consistent with the value of the Registered Per-Note Controller Message with Controller #3 (Pitch 7.25). This is different from the data format of RPN #00.00 (Channel Pitch Bend Sensitivity), which has an MSB with 7 bits HCU and LSB with a value in Cents between 0 and 99.

7.4.13.2 Supported Resolution

The Registered Controller for Sensitivity of Per-Note Pitch Bend provides 32 bits of resolution, more than needed or supportable by many Devices.

A Receiver that recognizes Registered Controller for Sensitivity of Per-Note Pitch Bend shall recognize the 7 bits of integer precision, subject to the supported range (see Section *7.4.13.3*).

A Receiver that recognizes Registered Controller for Sensitivity of Per-Note Pitch Bend may interpret and respond to any number of bits of fractional resolution that the Receiver can support.

> For example: A receiver might recognize the 7 bits of integer precision and only 8 bits of fractional precision, resulting in a net precision of 100/256 Cents.
>
> *Note: Historically, many implementations of RPN #00.00 Pitch Bend Sensitivity ignored the Cents field. For this reason, typical MIDI 1.0 transmitters configured Pitch Bend sensitivity in 100-Cent units and scaled Pitch Bend values as needed. This results in improved compatibility with devices that do not implement fractional sensitivity.*

7.4.13.3 Supported Range

The Registered Controller for Sensitivity of Per-Note Pitch Bend provides up to almost +/- 12800 Cents of range, more than needed or supportable by many Devices. Devices may support a subset of the whole range of sensitivity values.

> For example: A receiver might only respond to values between 0x00000000 and 0x0C000000 (sensitivity between +/- 0 Cents and +/- 1200 Cents).

7.4.13.4 Implementing a Unique Per-Note Range Amount for Each Note Number

This section does not define any specification rules or recommendations; it is informational only.

The Registered Controller for Sensitivity of Per-Note Pitch Bend sets an equal sensitivity for all note Numbers. In some cases, it is desirable to have a unique amount of Pitch Bend for a specific Note Number or for each Note Number.

A MIDI 2.0 Sender may implement a unique amount of Pitch Bend for each Note:

1. The Sender should set the Registered Controller for Sensitivity of Per-Note Pitch Bend to the sensitivity required by the Note Number with the widest Pitch Bend range requirement.
2. For Note Numbers that require a smaller Pitch Bend amount, the Sender should send Per-Note Pitch Bend messages with only a subset of the values from the whole range of available values. The 32 bit value of Per-Note Pitch Bend has sufficient resolution to provide smooth changes of pitch, even while using only a subset range of the total available values.

7.4.14 MIDI 2.0 Note On/Off: Attribute Type & Attribute Data

Attribute Type and Attribute Data fields enable a MIDI 2.0 Protocol Note On or Note Off message to address more properties than a MIDI 1.0 Protocol Note On or Note Off message. Those properties might be defined as articulation information, pitch information, or any other performance data such as strike position on a drum or cymbal.

The currently defined Attribute Types are:

Table 8 Defined Attribute Types for MIDI 2.0 Note On & Note Off

Attribute Type	Definition	Notes
0x00	No Attribute Data	Sender shall set Attribute Value to 0x0000 Receiver shall ignore Attribute Value
0x01	Manufacturer Specific	Interpretation of Attribute Data is determined by manufacturer
0x02	Profile Specific	Interpretation of Attribute Data is determined by MIDI-CI Profile in use
0x03	Pitch 7.9	See Section ***7.4.15.3***
0x04 – 0xFF	Reserved	Do not use

Attribute Type 0x00: No Attribute Data

In a Note On/Off message with no attribute data, the Attribute Type shall be set to 0x00 and the Attribute Data shall be set to 0x0000.

Attribute Type 0x01: Manufacturer Specific Data (and Unknown Data Type)

If a Sender transmits Attribute data that does not conform to any defined Attribute Types, then it should set the Attribute Type to 0x01. If a Sender transmits Attribute data but the type of data is unknown, then it should set the Attribute Type to 0x01.

Attribute Type 0x02: Profile Specific Data

When Attribute Type = 0x02, the meaning of the Attribute Data field is defined by the currently active MIDI-CI Profile. This definition of the Attribute Data is only known by devices which understand the Profile. Devices that do not understand the currently active Profile should ignore the Attribute Data when the Attribute Type is set to 0x02. Attribute Type = 0x02 is intended to declare the values of Profile properties that are unique to one Profile and are unlikely to be used by or interoperate with any other Profile.

Note: Attribute Types That should Not Use Attribute Type = 0x02

Attribute Type = 0x02 should not be used for properties which might be shared by multiple Profiles.

For example, a hypothetical Guitar Profile might define a distinct Attribute Type as "Position", with the Attribute Data value declaring the picking position from the bridge to the nut. A hypothetical Violin Profile might both use the same "Position" Attribute Type for bowing position. A hypothetical Drum Profile might use the same for strike position on a drum or cymbal from center of a to outer edge.

MA/AMEI should define dedicated Attribute Type values (not 0x02) with specific purpose for properties such as "Position" which are shared by or are likely to be shared by several Profiles.

Attribute Type = 0x02 should not be used for properties in Profiles which are widely used.

For example, a hypothetical Piano Profile might define that the Attribute Data field contains a 2nd velocity property for hammer velocity. Even if no other Profiles need a 2nd velocity property, Piano is a very common instrument and warrants the assignment of a unique dedicated Attribute Type. MA/AMEI should define a dedicated Attribute Type value (not 0x02) for Attribute Data in a Profile that is expected to be widely supported by many Devices.

Attribute Type 0x03: Pitch 7.9

When using this Attribute Type, the Note Number should be treated as a Note Index only; it does not imply any scale or pitch. Pitch is a Q7.9 fixed-point unsigned integer that specifies a pitch in HCUs. See Section ***7.4.15.3*** for implementation details, including interaction with other messages that influence or determine pitch.

7.4.15 MIDI 2.0 Notes and Pitch

The MIDI 2.0 Protocol preserves all the tuning definitions of the MIDI 1.0 Protocol, including Note Number, MIDI Tuning Standard, Master Tuning RPN 01 and RPN 02, and Pitch Bend. In addition, the MIDI 2.0 Protocol adds new mechanisms for Per-Note Tuning and Pitch control.

Pitch of a Note is determined by any combination of the following message components, some of which override (take priority over) others:

- Messages that Set the Default Pitch as done in the MIDI 1.0 Protocol (pitch is only roughly defined):
  - Note On with Note Number
- Messages that Set Pitch (override Default) with Persistent State for Subsequent Note Ons:
  - MIDI Tuning Standard
  - Registered Per-Note Controller #3: Pitch 7.25
- Messages that Set Pitch (override Default) for One Note Only:

  - Note On With Attribute #3 Pitch 7.9
- Messages that Modify Pitch Relatively from Any Existing Pitch State:
  - Channel Tuning RPN 01 and RPN 02
  - Per-Note Pitch Bend
  - Pitch Bend

Note: There might be other messages, from among the currently reserved messages, or mechanisms defined by MMA/AMEI in the future that also determine pitch. Such messages or mechanisms might be defined in future revisions of the MIDI 2.0 Protocol, MIDI-CI Profile specifications, or Articulation Types, or other expansions of MIDI.

Note: Receivers that select samples for playing a note based on Note Number might choose to instead select samples based on the first 7 bits of the pitch data in the last valid Registered Per-Note Controller #3: Pitch 7.25 or in the Note On With Attribute #3 Pitch 7.9.

7.4.15.1 MIDI Tuning Standard

The MIDI 1.0 Protocol and the MIDI 2.0 Protocol both support the existing MIDI Tuning Standard, which is formatted as a System Exclusive message. For fundamental functions and details of MIDI Tuning Standard, see the MIDI 1.0 Specification ***[MA01]***.

7.4.15.2 MIDI 2.0 Registered Per-Note Controller #3: Pitch 7.25

Registered Per-Note Controller #3 is defined as Pitch 7.25. The message's 32-bit data field contains:

- 7 bits: Pitch in HCUs, based on default Note Number equal temperament scale
- 25 bits: Fractional Pitch above Note Number (i.e., fraction of one HCU)

Pitch is a Q7.25 fixed-point unsigned integer that specifies a pitch in HCUs. The integer part shall be interpreted as if it were the pitch implied by the MIDI 1.0 Note Number as defined by the MIDI 1.0 Specification ***[MA01]*** in a 12-tone equal tempered scale with A=440 (Note number 69 [0x45]). The fractional part is a fraction of one HCU.

A Receiver that is capable of receiving Registered Per-Note Controller #3: Pitch 7.25 is free to interpret and respond to any number of bits of tuning resolution that the Receiver can support. Support for all 25 bits of fractional pitch resolution is not mandated. However, at least 9 bits should be supported (strongly recommended).

Pitch Bend and Per-Note Pitch Bend act as offsets from the pitch set by Registered Per-Note Controller #3: Pitch 7.25.

Important: The Pitch set by this Registered Per-Note Controller #3: Pitch 7.25 overrides the pitch set by previous MIDI Tuning Standard (MTS) messages. Controllers create persistent state, so all notes that follow this message use the tuning of the Registered Per-Note Controller #3: Pitch 7.25, unless they have other tuning information in the Note On message.

Two Typical Uses of Registered Per-Note Controller #3: Pitch 7.25:

- Registered Per-Note Controller #3 (Pitch 7.25) modifies the pitch of an individual Note Number. A set of these messages for multiple Note Numbers can be used to define a complete tuning table for any and all 128 Note Numbers.
- Registered Per-Note Controller #3 (Pitch 7.25) can also be used to control pitch in real time throughout the life cycle of a note.

7.4.15.3 MIDI 2.0 Note On With Attribute #3 Pitch 7.9

Attribute Type #3 is defined as **Pitch 7.9**. The 16-bit Attribute Value field contains:

- 7 bits: Pitch in HCUs, based on default Note Number equal temperament scale
- 9 bits: Fractional pitch above Note Number (i.e., fraction of one HCU)

When using this Attribute Type, the Note Number should be treated as a note index only; it does not imply any scale or pitch. Attribute Pitch is a Q7.9 fixed-point unsigned integer that specifies a pitch in HCUs. The integer part shall be interpreted as if it were the pitch implied by the Note Number as defined by the MIDI 1.0 Specification ***[MA01]*** in a 12-tone equal tempered scale with A=440 (Note number 69 [0x45]). The fractional part is a fraction of an HCU. That has a resolution of 1/512 HCUs, which provides an accuracy of approximately 0.2 cents.

Pitch Bend and Per-Note Pitch Bend act as offsets from the Attribute #3: Pitch 7.9.

Important: The Pitch set by this Attribute Pitch #3: 7.9 overrides the pitch previously set or implied by other mechanisms such as Registered Per-Note Controller #3: Pitch 7.25 and the MIDI Tuning Standard (MTS). This override is valid only for the one Note containing the Attribute #3: Pitch 7.9; it is not valid for any subsequent Notes.

7.5 Flex Data Messages

This section defines the use of Message Type = 0xD in the Universal MIDI Packet data format. Flex Data Messages have flexible addressing and may consist of multiple UMPs.

The Flex Data Messages Type has a large index of status values for a wide range of applications. Some of the messages are intended for real time set commands, with priority similar to Channel Voice Messages. When a Flex Data message is intended for real time application, the size of the message should be kept as small as possible.

7.5.1 Flex Data Messages General Format

The Message Type 0xD format has standardized mechanisms which may be used by any message definitions with the Message Type field set to 0xD:

- A Format field is used to optionally allow a message to have a variable size data, in multiples of 128 bits.
- An Address field is used to indicate if the message is addressed to a Channel (using the Channel field) or to a Group (like System Messages).

The message format includes a large Status space for many messages to be defined in the future.

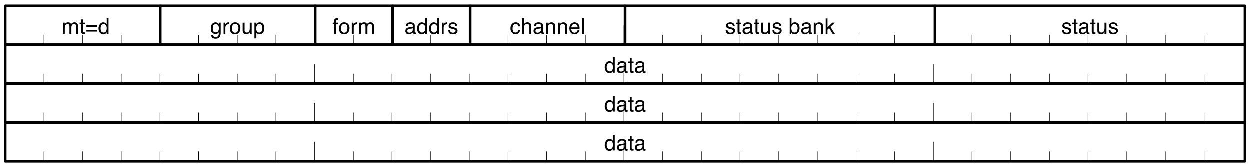


Figure 69 Flex Data Messages General Format

Format (form)

The 2-bit **Format** field determines the role of each UMP in a Flex Data Message:

Table 9 Flex Data Message Format Field Values

Format Field Value	UMP Type
0	Complete Flex Data Message in one UMP
1	Flex Data Message Start UMP
2	Flex Data Message Continue UMP There might be multiple Continue UMPs in a single message
3	Flex Data Message End UMP

A short Flex Data Message might fit into one UMP. Other Flex Data Messages span multiple UMPs.

Every Flex Data Message shall be in one of two formats:

- A Complete Flex Data Message in one UMP
  Or
- Begin with a Flex Data Message Start UMP and terminate with a Flex Data Message End UMP. Optional Flex Data Message Continue UMP may be used between Start and End UMPs to provide sufficient payload space for any data set.

A Flex Data Message shall not be larger than 32 UMPs. Therefore, the largest Flex Data message allowed would consist of a Start UMP, 30 Continue UMPs, and an End UMP.

Address (addrs)

The 2-bit Address field determines the address destination of each UMP in a Flex Data Message:

Table 10 Flex Data Message Address Field Values

Address Field Value	Message Addressing
0	Message is sent to the Channel set in the Channel Field
1	Message is sent to the Group (ignore Channel Field)
2	Reserved
3	Reserved

Channel

This 4-bit field declares a destination for the Flex Data message.

When Address field is Set to 0x0:

Send to MIDI Channel 1 (0x0) through MIDI Channel 16 (0xF).

When Address field is larger than 0x0:

MIDI Channel field is reserved, set to 0x0.

Status Bank and Status

The Status Bank field provides up to 256 message classifications. The Status field provides up to 256 message definitions within each Status Bank.

The following Status Banks are defined:

Table 11 Status Bank Classifications

Status Bank	Classification
0x00	Setup & Performance Events (does not include text events)
0x01	Metadata Text
0x02	Performance Text Events (including lyrics)
0x03-FF	Reserved

7.5.2 Limitations of Interspersing Other Messages with Flex Data Messages

Senders should obey the following data rules that govern interspersing other messages and termination of Flex Data Messages within a Group:

1. The Sender should not send any other Message or UMP between the Start and End of the Flex Data Message, except for Flex Data Message Continue UMPs for the same message, System Real Time Messages, and JR Clock Messages.
2. System Real Time Messages and JR Clock Messages may be inserted between the UMPs of a Flex Data Message, to maintain timing synchronization.

7.5.3 Set Tempo Message

This message sets musical tempo by declaring the number of 10 nanosecond units per quarter note. This may also help to increase the accuracy of timing synchronization over the accuracy provided when using MIDI Clocks alone, by allowing a calculation of forecasted time to subsequent MIDI Clock messages.

- If a Device is sending MIDI Clock messages over a MIDI Transport, Set Tempo messages shall only occur at the same timing as when a MIDI Clock occurs.
- If a Device is NOT sending MIDI Clock messages over a MIDI Transport, Set Tempo messages should only occur at the sender's notion of timing of 1/24 of a 1/4 note.
- In a Standard MIDI File, Set Tempo messages shall only occur at the timing of 1/24 of a 1/4 note.

mt=d	group	frm=0	adr=1	channel	status bank = 0x00	status - 0x00
number of 10 nanosecond units per quarter note						
reserved						
reserved						

Figure 70 Set Tempo Message Format

Format (form)

Format shall be set to 0 (Complete Flex Data Message in one UMP).

Address (addrs)

Address shall be set to 1 (Group).

Number of 10 Nanosecond units Per Quarter Note

The time per quarter note is 32 bits in units of 10 Nanoseconds. This allows a four-minute piece at 120 beats per minute to be accurate within 5 microseconds at the end of the piece. The slowest tempo supported is 1.39698 bpm.

A sequencer may determine how many bits of resolution it will support to achieve the accuracy that the sequencer deems appropriate. A sequencer is not required to support all declared Tempos on import; very slow or very fast tempos might be changed to the minimum or maximum of the sequencer at the time of importing.

7.5.4 Set Time Signature Message

This message declares and sets a Time Signature for subsequent bars.

- If a Device is sending MIDI Clock messages over a MIDI Transport, Set Time Signature messages shall only occur at the time when one bar ends (according to any previously set time signature if such existed) and the next bar begins with the new time signature.

- If a Device is NOT sending MIDI Clock messages over a MIDI Transport, Set Time Signature messages should only occur at the sender's notion of when one bar ends (according to the previously set time signature) and the next bar begins with the new time signature.
- In a Standard MIDI File, Set Time Signature messages shall only occur when one bar ends (according to any previously set time signature if such existed) and the next bar begins with the new time signature.

mt=d	group	frm=0	adr=1	channel	status bank = 0x00	status = 0x01
numerator		denominator			number of 1/32 notes	reserved
reserved						
reserved						

Figure 71 Set Time Signature Message Format

Format (form)

Format shall be set to 0 (Complete Flex Data Message in one UMP).

Address (addrs)

Address shall be set to 1 (Group).

Numerator

The Numerator field contains a value from 1 to 256, supporting up to 256 beats in a bar.

Denominator

The Denominator field contains a value in negative power of 2 (2 represents a quarter note, 3 represents an eighth note, etc.) If the value is set to zero, there is a non-standard denominator.

Number of 1/32 Notes

The Number of 1/32 Notes field expresses the number of 1/32 notes in 24 MIDI Clocks. This is copied from SMF 1, where it is stated:

> "This was added because there are already multiple programs which allow the user to specify that what MIDI thinks of as a quarter-note (24 clocks) is to be notated as, or related to in terms of, something else."

7.5.5 Set Metronome Message

This message sets metronome functions.

- This message may be used by a sequencer to configure its metronome.
- This message may be sent to configure the metronome of a receiver.

mt=d	group	frm=0	adr=1	channel	status bank = 0x00	status = 0x02
num clocks per primary click		bar accent part 1			bar accent part 2	bar accent part 3
num subdivision clicks 1		num subdivision clicks 2			reserved	
reserved						

Figure 72 Set Metronome Message Format

Format (form)

Format shall be set to 0 (Complete Flex Data Message in one UMP).

Address (addrs)

Address shall be set to 1 (Group).

Primary Clicks

Number of MIDI Clocks Per Primary Click is used to set a simple metronome, typically repeating at the value declared by a Set Tempo message.

Bar Accents

The three Bar Accent fields are used to set accents, typically sounding on the downbeat of a bar and optionally sounding for divisions of a bar.

If the bar is not divided, then the value of Bar Accent Part 1 shall be equal to the number of beats in the bar, Bar Accent Part 2 shall be set to zero, and Bar Accent Part 3 shall be set to zero.

A bar may optionally be divided by up to 3 parts. For example:

- A bar of 5/4 may be declared to have a division of 3 + 2.
- A bar of 9/4 may be declared to have a division of 3 + 3 + 3 or of 4 + 4 + 1.

The sum of Bar Accent Part 1 plus Bar Accent Part 2 plus Bar Accent Part 3 fields shall be equal to the number of beats in a bar.

Subdivision Clicks

Subdivision Click is added for clicks between the Primary Clicks. The value of the Number of Subdivision Clicks declares how many Subdivision Clicks which will sound within the period of a Primary Click.

For example, in 4/4 time, a subdivision might be used to sound 2 x 1/8 note clicks or 4 x 1/16 notes per quarter note. In 6/8 time, a subdivision might be used to sound 3 x 1/8 clicks in 6/8 time.

Set a value of 0 to not use any Subdivision Clicks.

There are two identical Number of Subdivision Clicks fields. This allows sounding of two overlapping subdivision clicks. This might be used to sound both 1/8 and 1/16 notes (probably with different sounding clicks) or for more complex combinations such as overlapping of 3 clicks and 4 clicks within the same period.

7.5.6 Example Set Metronome Messages

Set Metronome: 4/4 with Bar accent, no subdivisions

mt=d	group	frm=0	adr=1	channel	status bank = 0x00	status = 0x02
24 = quarter notes		4 = accent every 4th beat			0	0
0		0			reserved	
reserved						

Set Metronome: 6/8 with Bar accent and 1/8 triplets

mt=d	group	frm=0	adr=1	channel	status bank = 0x00	status = 0x02
36 = dotted 1/4 notes		2 = accent every 2nd beat			0	0
3 = 1/8 note triplets		0			reserved	
reserved						

Set Metronome: 5/4 (3+2 accent) and 1/8 notes

mt=d	group	frm=0	adr=1	channel	status bank = 0x00	status = 0x02
24 = quarter notes		3 = accent for 3 beats			2 = accent for 2 beats	0
2 = 1/8 notes		0			reserved	
reserved						

Set Metronome: 3/4 with Bar accent, 1/8 notes + 1/8 triplets

mt=d	group	frm=0	adr=1	channel	status bank = 0x00	status = 0x02
24 = quarter notes		3 = accent every 3rd beat			0	0
2 = 1/8 notes		3 = 1/8 note triplets			reserved	
reserved						

Figure 73 Example Set Metronome Messages

7.5.7 Set Key Signature Message

This message sets the Key Signature for up to 7 sharps or up to 7 flats. A field is provided to declare which note within the key is the tonic note.

mt=d	group	frm=0	addrs	channel	status bank= 0x00	status = 0x05
sharps/flats	tonic note	reserved				
reserved						
reserved						

Figure 74 Set Key Signature Message Format

Sharps/Flats Field

This is a 4-bit field with a two's complement signed value. Positive values declare the number of sharps in the Key Signature. Negative numbers declare the number of flats in the Key Signature. A value of -8 (binary 1000) = unknown or non-standard.

Tonic Note Field

This field uses the musical Mode to declares the Tonic note within the selected Key Signature:

0x0 = unknown or non-standard

0x1 = A

0x2 = B

0x3 = C

0x4 = D

0x5 = E

0x6 = F

0x7 = G

0x8 - 0xF = reserved

The sharps/flats field determines whether the note in value of the Tonic Note Field is the note is natural, sharp or flat. For example:

Table 12 Sharps and Flats Examples

Sharps/Flats Field	Tonic Note Field	Intended Tonic Note
One Sharp	D	D Natural
Five Sharps	D	D Sharp
Four Flats	D	D Flat

7.5.8 Set Chord Name Message

This message declares the name of a chord. The chord can have an optional, alternate bass note and an optional bass chord.

mt=d	group	frm=0	addrs	channel	status bank = 0x00		status = 0x06	
t. sharps/flats	chord tonic	chord type			alter 1 type	alter 1 degree	alter 2 type	alter 2 degree
alter 3 type	alter 3 degree	alter 4 type		alter 4 degree	reserved			
b. sharps/flats	bass note	bass chord type			alter 1 type	alter 1 degree	alter 2 type	alter 2 degree

Figure 75 Set Chord Message Format

Tonic Sharps/Flats

This is a 4-bit field with a two's complement signed value. Positive values declare the number of sharps applied to the Tonic Note. Negative numbers declare the number of flats Applied to the Tonic Note.

Table 13 Tonic Sharps and Flats Values

Two's Complement Value	Decimal Value	Applied to Tonic
0x2	2	Double Sharp
0x1	1	Sharp
0x0	0	Natural
0xF	-1	Flat

0xE	-2	Double Flat
All Other Values Reserved	All Other Values Reserved	Reserved

Chord Tonic Note

This field declares the Tonic note of the chord:

0x0 = Unknown (used for No Chord)

0x1 = A

0x2 = B

0x3 = C

0x4 = D

0x5 = E

0x6 = F

0x7 = G

0x8 - 0xF = reserved

The Tonic Sharps/Flats field determines whether the note in the Tonic Note field is natural, sharp, double sharp, flat, or double flat.

Chord Type Field

Chord Type is an enumerated field.

Table 14 Chord Type Field Values

Value	**Chord Type**
0x00	Clear Chord - No Chord
0x01	Major
0x02	Major 6th
0x03	Major 7th
0x04	Major 9th
0x05	Major 11th
0x06	Major 13th
0x07	Minor
0x08	Minor 6th
0x09	Minor 7th
0x0A	Minor 9th
0x0B	Minor 11th
0x0C	Minor 13th
0x0D	Dominant

0x0E	Dominant ninth
0x0F	Dominant 11th
0x10	Dominant 13th
0x11	Augmented
0x12	Augmented seventh
0x13	Diminished
0x14	Diminished seventh
0x15	Half diminished (Diminished triad, minor seventh)
0x16	Major-minor or Minor-major (Minor triad, major seventh)
0x17	Pedal (e.g. XF 1+8)
0x18	Power (e.g. XF 1+5)
0x19	Suspended 2nd (e.g. XF 1+2+5)
0x1A	Suspended 4th
0x1B	7 Suspended 4th
0x1C - 0xFF	Reserved

Alteration Type and Alteration Degree Fields

Alterations can be made to the declared chord. An alteration is declared by the combination of an Alteration Type and an Alteration Degree. Up to 4 alterations can be made to the main Chord Type and up to 2 alterations can be made to the Bass Chord Type.

Alteration Type:

0: No alteration

1: Add degree

2: Subtract degree

3: Raise degree, adding if needed

4: Lower degree, adding if needed

5-15: Reserved

Degree:

The number indicating the degree of the chord (1 for the root, 3 for third, etc.) which is altered.

Bass Sharps/Flats Field

This is a 4-bit field with a two's complement signed value. Positive values declare the number of sharps applied to the Bass Note. Negative numbers declare the number of flats applied to the Bass Note. A value of -8 (binary 1000) = Bass Note is the same as the Chord Tonic Note; Set the Bass Note field to 0x0.

Table 15 Bass Note Sharps and Flats Values

Two's Complement Value	Decimal Value	Applied to Bass Note
0x2	2	Double Sharp
0x1	1	Sharp
0x0	0	Natural
0xF	-1	Flat
0xE	-2	Double Flat
0x8	-8	Same Note as Chord Tonic Note
All Other Values Reserved	All Other Values Reserved	Reserved

Bass Note Field

This field declares the Bass note of the chord:

0x0 = same as the Chord Tonic Note. Set the Bass Sharps/Flats field to -8 (binary 1000).

0x1 = A

0x2 = B

0x3 = C

0x4 = D

0x5 = E

0x6 = F

0x7 = G

0x8 - 0xF = reserved

The sharps/flats field determines whether the note in the Bass Note field is natural, sharp or flat.

Bass Chord Type Field

Bass Chord Type is an enumerated field.

0x00 = Clear Bass Chord - No Bass Chord

0x01 - 0xFF = same values as Chord Type Field

7.5.8.1 Example Set Chord Name Messages

Set Chord Name: B♭Min

mt=d	group	f=0	addrs	channel	status bank = 0x00		status = 0x06	
0xF	0x2	chord type = 0x07			0x0	0x0	0x0	0x0
0x0	0x0	0x0		0x0	reserved			
0x8	0x0	bass chord type = 0x00			0x0	0x0	0x0	0x0

Set Chord Name: DMaj/F#

mt=d	group	f=0	addrs	channel	status bank = 0x00		status = 0x06	
0x0	0x4	chord type = 0x01			0x0	0x0	0x0	0x0
0x0	0x0	0x0		0x0	reserved			
0x1	0x6	bass chord type = 0x00			0x0	0x0	0x0	0x0

Set Chord Name: CMaj7 $^{\#11}$

mt=d	group	f=0	addrs	channel	status bank = 0x00		status = 0x06	
0x0	0x3	chord type = 0x03			0x3	0xB	0x0	0x0
0x0	0x0	0x0		0x0	reserved			
0x8	0x0	bass chord type = 0x00			0x0	0x0	0x0	0x0

Figure 76 Example Set Chord Name Messages

7.5.9 Text Messages Common Format

All Flex Data Messages with Status Bank field set to 0x01 or 0x02 shall contain text encoded in UTF-8 format, without a Byte Order Mark.

If the text ends in the middle of a UMP, then the remaining data bytes shall be set to 0x00 to indicate the end of the text. The value 0x00 shall only occur in an End UMP (Form = 0x3) or Complete UMP (Form = 0x0).

mt=d	group	form	addrs	channel	status bank	status
data						
data						
data						

Figure 77 Flex Data Text Messages Common Format

7.5.9.1 Messages Which use the Text Common Format

The following table lists messages which conform to the Text Messages Common Format.

Table 16 Text Messages by Status

Status Bank	Status	Message	Notes
0x01	0x00	Unknown Metadata Text Event	
0x01	0x01	Project Name	Address field should be set to 1 (Group)

0x01	0x02	Composition (Song) Name	
0x01	0x03	MIDI Clip Name	Address field should be set to 1 (Group)
0x01	0x04	Copyright Notice	
0x01	0x05	Composer Name	
0x01	0x06	Lyricist Name	
0x01	0x07	Arranger Name	
0x01	0x08	Publisher Name	
0x01	0x09	Primary Performer Name	
0x01	0x0A	Accompanying Performer Name	
0x01	0x0B	Recording/Concert Date	See Section ***7.5.9.2***
0x01	0x0C	Recording/Concert Location	
0x02	0x00	Unknown Performance Text Events	
0x02	0x01	Lyrics	
0x02	0x02	Lyrics Language	
0x02	0x03	Ruby	
0x02	0x04	Ruby Language	

Multiple Entities/Names

Some data in messages which conform to the Text Messages Common Format might represent more than a single entity. For example, there may be multiple people to be represented by Composer Name or Accompanying Performer Name messages. A device or system may choose to put each entity into a single message or put multiple entities into a single message. When multiple entries are declared in a single message, the entries should be separated by a comma. For example, a Composer Name message might include: "John Lennon, Paul McCartney".

Examples:

Following are two examples of the Copyright Notice message using this format.

©2022 AMEI

mt=d	group	0	addrs	channel	status bank = 0x01	status = 0x04
0xC2		0xA9			2	0
2		2			space	A
M		E			I	0x00

Copyright 2022 MIDI Association Publishing

mt=d	group	1	addrs	channel	status bank = 0x01	status = 0x04
C		o			p	y
r		i			g	h
t		space			2	0

mt=d	group	2	addrs	channel	status bank = 0x01	status = 0x04
2		2			space	M
I		D			I	space
A		s			s	o

mt=d	group	2	addrs	channel	status bank = 0x01	status = 0x04
c		i			a	t
i		o			n	space
P		u			b	l

mt=d	group	3	addrs	channel	status bank = 0x01	status = 0x04
i		s			h	i
n		g			0x00	0x00
0x00		0x00			0x00	0x00

Figure 78 Example Flex Data Text Messages

7.5.9.2 Recording/Concert Date

The Recording/Concert Date message shall conform to the Text Messages Common Format. The data of the Recording/Concert Date message shall also conform to ISO 8601 for the Date with an optional time.

7.5.10 Lyric Data Message

The Lyric Data message contains Lyrics as Unicode UTF-8 text.

Each message shall contain an individual syllable of a word (or a word if the word has only one syllable). The syllable shall be placed or sent at the time at which the sung note begins. A syllable shall be contained in either:

- a single Complete UMP (Format = 0x0)

  or
- a multi-packet message terminated with an End UMP (Format = 0x3).

To best honor the design characteristics of Flex Data Messages and the rules in Section 7.5.2, all parts of a Lyric Data Message should be sent together. When placed in a MIDI Clip File, all parts of the Lyric Data Message shall be placed at the same time, sharing a single Delta Clock Stamp.

If the syllable ends in the middle of a UMP (always a Complete UMP or End UMP) but the word has not ended, then the remaining data bytes shall be set to 0x00 to indicate the end of the syllable.

A Space character (0x20) at the end of the text shall indicate the end of a word. If a Lyric Data message does NOT end with a space it is then known that the next lyric event following is a continuation of the same word.

A Carriage Return character (0x0D) at the end of the text shall indicate the end of a line.

A Line Feed character (0x0A) which follows after a Carriage Return character shall indicate the end of a paragraph.

After the end of a syllable, word, line, and/or paragraph, if the data ends in the middle of a UMP (always a Complete or End), then the remaining data bytes shall be set to 0x00 to indicate the end of the text.

The value 0x00 shall not occur in a Start UMP (Format = 0x1). The value 0x00 in a Continue UMP (Format = 0x1) indicates a melisma (See Section *7.5.10.1*).

Hyphens shall only be used for hyphenated words like “sixty-four”. Otherwise, the absence of a space / CR / LF at the end of the text indicates that more text follows within the word, so applications that want to insert hyphens can do so then.

mt=d | group	form | addrs | channel	status bank = 0x02	status = 0x01
data / reserved	data / reserved	data / reserved	data / reserved
data / reserved	data / reserved	data / reserved	data / reserved
data / reserved	data / reserved	data / reserved	data / reserved

Figure 79 Lyric Data Message Format

7.5.10.1 Melisma Event

A Lyric Data Complete UMP (Format = 0x0) which is filled with the value 0x00 does not end the current word and does not specify a new syllable; it therefore specifies a melisma. This indicates that the syllable in the previous Lyric Data message should continue to be sung.

If the melisma is the end of a word, then another Lyric Data message with a Space character (0x20) shall follow immediately after the melisma. Carriage Return (0x0D) and Line Feed (0x0A) characters may be included in this word end Lyric Data event if needed.

7.5.11 Lyric Language Message

The Lyric Language message has a data field which is a BCP 47 language identifier.

Note that BCP 47 uses the ASCII subset of UTF-8.

mt=d | group	form | addrs | channel	status bank = 0x02	status = 0x02
data / reserved	data / reserved	data / reserved	data / reserved
data / reserved	data / reserved	data / reserved	data / reserved
data / reserved	data / reserved	data / reserved	data / reserved

Figure 80 Lyric Language Message Format

All Lyric messages that follow a Lyric Language message at the same address are to be interpreted as using the declared language, until the next Lyric Language message is received at this address. If languages change during a sequence, send a Lyric Language message preceding the language switch.

> *For example: If Channel 1 always has English lyrics and Channel 2 always has Japanese lyrics, only one Lyric Language message needs to be sent per channel, before any Lyric Data messages are sent on those Channels.*

7.5.12 Ruby Data Message

The Ruby Data message contains Ruby lyrics as Unicode UTF-8 text. The positioning of Ruby Data for display is to be determined by the specific application which is using the data.

Each message shall contain an individual syllable of a word (or a word if the word has only one syllable). The syllable shall be placed or sent at the time at which the sung note begins. A syllable shall be contained in either:

- a single Complete UMP (Form = 0x0)

  or
- a multi-packet message terminated with an End UMP (Form = 0x3).

To best honor the design characteristics of Flex Data Messages and the rules in Section 7.5.2, all parts of a Ruby Data Message should be sent together. When placed in a MIDI Clip File, all parts of the Ruby Data Message shall be placed at the same time, sharing a single Delta Clock Stamp.

If the syllable ends in the middle of a UMP (always a Complete UMP or End UMP) but the word has not ended, then the remaining data bytes shall be set to 0x00 to indicate the end of the syllable.

A Space character (0x20) at the end of the text shall indicate the end of a word.

A Carriage Return character (0x0D) at the end of the text shall indicate the end of a line.

A Line Feed character (0x0A) following a Carriage Return character shall indicate the end of a paragraph.

After the end of a syllable, word, line, and/or paragraph, if the data ends in the middle of a UMP (always a Complete or End), then the remaining data bytes shall be set to 0x00 to indicate the end of the text.

The value 0x00 shall not occur in a Start (Form = 0x1) UMP. The value 0x00 in a Continue (Form = 0x1) UMP indicates a melisma (See Section ***7.5.10.1***).

Hyphens shall only be used for hyphenated words like “sixty-four”. Otherwise, the absence of a space / CR / LF at the end of the text indicates that more text follows within the word, so applications that want to insert hyphens can do so then.

mt=d	group	form	addrs	channel	status bank = 0x02	status = 0x03
data / reserved		data / reserved			data / reserved	data / reserved
data / reserved		data / reserved			data / reserved	data / reserved
data / reserved		data / reserved			data / reserved	data / reserved

Figure 81 Ruby Data Message Format

7.5.12.1 Melisma Event

A Ruby Data Complete UMP (Format = 0x0) which is filled with the value 0x00 does not end the current word and does not specify a new syllable; it therefore specifies a melisma. This indicates that the syllable in the previous Ruby Data message should continue to be sung.

If the melisma is the end of a word, then another Ruby Data message with a Space character (0x20) shall follow immediately after the melisma. Carriage Return (0x0D) and Line Feed (0x0A) characters may be included in this word end Ruby Data event if needed.

7.5.13 Ruby Language Message

The Ruby Language message has a data field which is a BCP 47 language identifier,

mt=d	group	form	addrs	channel	status bank = 0x02	status = 0x04
data / reserved		data / reserved			data / reserved	data / reserved
data / reserved		data / reserved			data / reserved	data / reserved
data / reserved		data / reserved			data / reserved	data / reserved

Figure 82 Ruby Language Message Format

All Ruby Data messages that follow a Ruby Language message at the same address are to be interpreted as using the declared language, until the next Ruby Language message is received at this address. If languages during a sequence, send a Ruby Language message preceding the language switch.

7.6 System Common and System Real Time Messages

System Common and System Real Time messages contain the same data as the message definitions in the MIDI 1.0 Specification ***[MA01]***.

System Messages in the MIDI 1.0 Protocol are 1, 2, or 3 bytes long. The same messages in the UMP Format are formatted to fit in a single 32-bit UMP.

Messages shorter than 3 bytes in the MIDI 1.0 Protocol have unused bytes in the UMP. These unused bytes are Reserved, shall be set to zero, and shall not be used because they might be defined by MMA/AMEI in future revisions of the UMP or MIDI protocols.

System Exclusive Messages are a unique type of System Message, and are specified in Section *7.7*. Status values 0xF0 and 0xF7, which in Non-UMP MIDI 1.0 Systems are used with System Exclusive messages, are not used for UMP System Exclusive; instead, they are reserved.

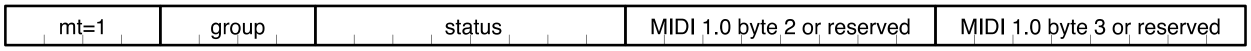


Figure 83 System Message General Format

Table 6 indicates which System Common and System Real Time Messages use this UMP Format.

Table 17 Messages that use System Message General Format

Message	Status	MIDI 1.0 Byte 2 and 3 or Reserved	
Reserved	0xF0	Reserved	Reserved
MIDI Time Code	0xF1	0nnndddd	Reserved
Song Position Pointer	0xF2	0lllllll*	0mmmmmmm*
Song Select	0xF3	0sssssss	Reserved
Reserved	0xF4	Reserved	Reserved
Reserved	0xF5	Reserved	Reserved
Tune Request	0xF6	Reserved	Reserved
Reserved	0xF7	Reserved	Reserved
Timing Clock	0xF8	Reserved	Reserved
Reserved	0xF9	Reserved	Reserved
Start	0xFA	Reserved	Reserved
Continue	0xFB	Reserved	Reserved
Stop	0xFC	Reserved	Reserved
Reserved	0xFD	Reserved	Reserved
Active Sensing	0xFE	Reserved	Reserved
Reset	0xFF	Reserved	Reserved

** Note: Song Position Pointer data is presented with LSB before MSB, as in the MIDI 1.0 Protocol.*

7.6.1 Consideration of Timing Clock on UMP Endpoints

System Common and System Real Time may be sent to any Group. A Device selects which Timing Clock(s) it is using from Timing Clocks received on different Groups (or its internal clock(s)). Many Devices only sync to a single Timing Clock, even if the Device has more than one function.

If a Receiver is synced to a single Timing Clock on one Group, it is not necessary to send Timing Clock on other Groups to that Receiver. This decision might be made by the Sender or might be a user selection.

If a Device includes a MIDI 1.0 output represented as a Group in the UMP Endpoint for the Device, the Device must select which Timing Clocks are sent on that output. This may come from Timing Clock messages received on the Group, from another Group, or the Device's own internal Timing Clock.

7.7 System Exclusive (7-Bit) Messages

UMP System Exclusive messages carry the same data payload as MIDI 1.0 Protocol System Exclusive messages, and can be translated directly to and from MIDI 1.0 Protocol System Exclusive Messages.

The MIDI 1.0 Protocol bracketing method with 0xF0 Start and 0xF7 End Status bytes is not used in the UMP Format. Instead, the SysEx payload is carried in one or more 64-bit UMPs, discarding the 0xF0 and 0xF7 bytes. The standard ID Number (Manufacturer ID, Special ID 0x7D, or Universal System Exclusive ID), Device ID, and Sub-ID#1 & Sub-ID#2 (if applicable) are included in the initial data bytes, just as they are in MIDI 1.0 Protocol message equivalents.

System Exclusive Messages use Message Type 0x3.

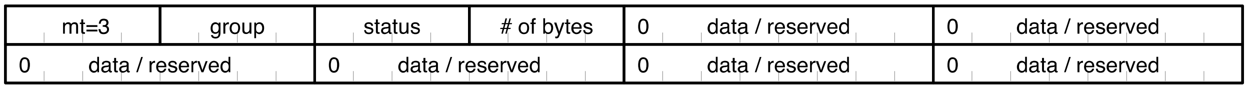


Figure 84 System Exclusive (7-Bit) Message Format

status

The 4-bit Status field determines the role of each UMP in a System Exclusive message:

Table 18 Status Field Values for System Exclusive (7-Bit) Messages

Status Field Value	UMP Type
0x0	Complete System Exclusive Message in one UMP
0x1	System Exclusive Start UMP
0x2	System Exclusive Continue UMP There might be multiple Continue UMPs in a single message.
0x3	System Exclusive End UMP

A short System Exclusive message might fit into one UMP. Other System Exclusive messages might span multiple UMPs.

Every System Exclusive Message shall be in one of two formats:

1. A Complete System Exclusive Message in one UMP

   Or

2. Begin with a System Exclusive Start UMP and terminate with a System Exclusive End UMP. Optional System Exclusive Continue UMPs may be used between the Start and End UMPs to provide sufficient payload space for any data set.

# of bytes

This declares the number of valid data bytes in each UMP, starting with the byte after the # of bytes field through to the end of the 64-bit UMP (i.e., 0 to 6 bytes).

Any unused bytes in the UMP are reserved, and shall be set to zero.

Note: Each System Exclusive UMP may contain fewer than 6 bytes of data. A Start or Continue with fewer than 6 bytes does not signify a message end.

7.7.1 Limitations of Interspersing Other Messages with System Exclusive UMPs

A significant feature of UMP System Exclusive Messages is direct compatibility with MIDI 1.0 Protocol System Exclusive Messages in all MIDI protocols and all MIDI systems.

To preserve robust connection to all MIDI devices and systems, Senders shall obey the following data rules of the MIDI 1.0 Protocol that govern interspersing other messages and termination of System Exclusive within a Group:

- The Sender shall not send any other Message or UMP on the same Group between the Start and End of the System Exclusive Message, except for System Exclusive Continue UMPs, and System Real Time Messages.
- System Real Time Messages on the same Group may be inserted between the UMPs of a System Exclusive message, in order to maintain timing synchronization.
- If any Message or UMP on the same Group, other than a System Exclusive Continue UMP or a System Real Time Message, is sent after a System Exclusive Start UMP and before the associated System Exclusive End UMP, then that UMP shall terminate the System Exclusive Message.

Messages which are Groupless (MT = 0x0 and 0xF) and those which are sent to a different Group may be interspersed with System Exclusive Messages.

7.8 System Exclusive 8 (8-Bit) Messages

System Exclusive 8 messages have many similarities to the MIDI 1.0 Protocol's original System Exclusive messages, but with the added advantage of allowing all 8 bits of each data byte to be used. By contrast, MIDI 1.0 Protocol System Exclusive requires a 0 in the high bit of every data byte, leaving only 7 bits to carry actual data. A System Exclusive 8 Message is carried in one or more 128-bit UMPs with Message Type 0x5.

Note: System Exclusive 8 Messages cannot be translated to Non-UMP MIDI 1.0 Systems. Many MIDI applications will continue to use traditional System Exclusive (7-bit) Messages (Section 7.7) for compatibility across a wide range of MIDI devices. System Exclusive 8 is suitable for applications that only apply to devices that use the UMP Format.

The initial data bytes found in MIDI 1.0 Protocol System Exclusive messages are included in the bytes directly following the Stream ID in System Exclusive 8. These bytes are Manufacturer ID (including Special ID 0x7D, or Universal System Exclusive IDs), Device ID, and Sub-ID#1 & Sub-ID#2 (if applicable).

Manufacturer ID numbers, which are 7-bit and 21-bit values in the MIDI 1.0 Protocol, are encoded in a 16-bit identifier (MfrID, see Section *7.10*) for System Exclusive 8 messages.

mt=5	group	status	# of bytes	stream id	data / reserved
data / reserved		data / reserved		data / reserved	data / reserved
data / reserved		data / reserved		data / reserved	data / reserved
data / reserved		data / reserved		data / reserved	data / reserved

Figure 85 System Exclusive 8 (8-Bit) Message Format

status

The 4-bit Status field determines the role of each UMP in a System Exclusive 8 message:

Table 19 Status Field Values for System Exclusive 8 (8-Bit) Messages

Status Field Value	UMP Type
0x0	Complete System Exclusive 8 Message in one UMP
0x1	System Exclusive 8 Start UMP
0x2	System Exclusive 8 Continue UMP There might be multiple Continue UMPs in a single message.
0x3	System Exclusive 8 End UMP

A short System Exclusive 8 message might fit into one UMP. Other System Exclusive 8 messages span multiple UMPs.

Every System Exclusive 8 Message shall be in one of two formats:

1. A Complete System Exclusive 8 Message in one UMP

Or

2. Begin with a System Exclusive 8 Start UMP, and terminate with a System Exclusive 8 End UMP. Optional System Exclusive 8 Continue UMP may be used between Start and End UMPs to provide sufficient payload space for any data set.

# of bytes

This 4-bit field declares the number of valid data bytes in each UMP, starting from and including the Stream ID through to the end of the 128-bit UMP (i.e., 1 to 14 bytes). Stream ID is mandatory (1 byte), so a value of 0x0 is not valid in the # of bytes field.

The special value 0xF is used in an End UMP to abort a System Exclusive 8 message (see Section *7.8.1*).

Unused bytes in the UMP are reserved and shall be set to zero.

Note: Each System Exclusive 8 UMP may contain fewer than 14 bytes of data. A Start or Continue with fewer than 14 bytes does not signify a message end.

Stream id

Interleaving of multiple simultaneous System Exclusive 8 messages is enabled by use of an 8-bit Stream ID field.

- A device which supports only one stream shall use 0 as the Stream ID.
- If a Sender wants to use more than one simultaneous stream, a Function Block Info Notification message from the Receiver declares how many simultaneous Stream IDs are supported (N). If either the Sender or the Receiver does not support Function Block Discovery message to discover the Receiver's support for more than one simultaneous Stream, then the Sender shall not send more than one simultaneous stream.
- For devices that support multiple streams, only Stream IDs from 0 to (N-1) shall be used.
- Stream IDs allow for simple mergers to be created. Streams from multiple sources can be merged, with the Merger device reassigning Stream IDs as necessary. Before a merger sends simultaneous System Exclusive 8 messages merged from various sources, it shall first perform a Function Block Discovery message to determine how many simultaneous Stream IDs are supported by the Receiver (N).

7.8.1 Unexpected End of Data

If the Sender runs out of payload data before sending a System Exclusive 8 End UMP, then the Sender shall send a System Exclusive 8 End UMP with all data bytes set to zero, and the **# of bytes** field set to either of the two following values:

- **0x1** if the Sender knows that the previous data in the SysEx8 message is valid. The value of 0x1 indicates that the first byte, the Stream Id, is valid and that no further data is included.
- **0xF** if the previous data is an incomplete message, or if the resulting quality of previous data is unknown.

*Note: Since System Exclusive 8 Messages cannot be translated to Non-UMP MIDI 1.0 Systems, there are no prohibitions against interspersing other message UMPs, as there are with the 7-bit System Exclusive Messages (see Section **7.7.1**).*

7.9 Mixed Data Set Message

Mixed Data Set messages can carry any data payload, without the 7-bit restriction of the MIDI 1.0 Protocol. This mechanism is targeted primarily for use with large data sets, including non-MIDI data.

*Note: Small data sets should continue to use System Exclusive (7-Bit) Messages (Section 5.4) for compatibility across a wide range of MIDI devices, or use System Exclusive 8 (8-Bit) Messages (**Section 7.8**) for applications that only apply to devices that use the UMP Format.*

Note: Mixed Data Set Messages cannot be translated to non-UMP MIDI 1.0 Systems. As a result, Mixed Data Set Messages are only suitable for applications that use the UMP Format.

The Mixed Data Set can carry non-MIDI data payloads such as XML or device firmware updates. The format of the data payload itself is not defined by this document, only the header and payload UMP Formats are defined.

Mixed Data Set messages can carry industry-standardized payloads using Universal System Exclusive IDs defined by MMA/AMEI in the header. Devices can use Mixed Data Set messages to carry any proprietary data using the device manufacturer's own Manufacturer ID.

Data is sent in 128-bit UMPs. Multiple 128-bit UMPs make up one Mixed Data Set Chunk. Each Mixed Data Set Chunk has one Mixed Data Set Header UMP, followed by multiple Mixed Data Set Payload UMPs. Multiple Mixed Data Set Chunks make up the total Mixed Data Set.

Mixed Data Set Messages use Message Type 0x5.

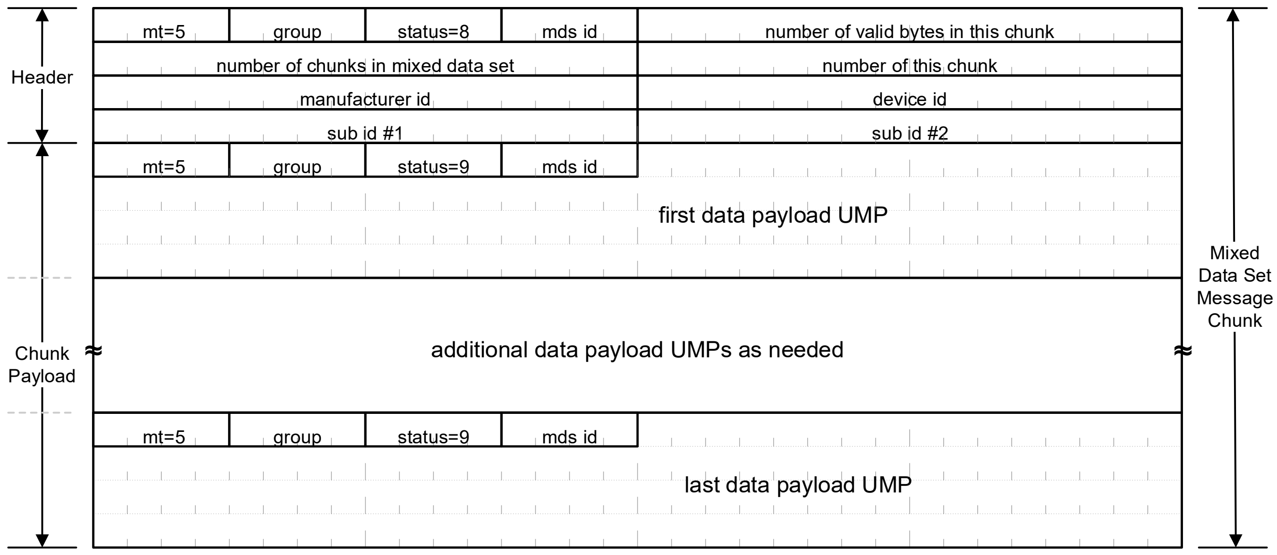


Figure 86 Mixed Data Set Chunk Format

Note: The total Mixed Data Set Message may require multiple Chunks.

Status

0x8: Mixed Data Set Header

0x9: Mixed Data Set Payload

MDS ID:

Each Mixed Data Set Message is assigned an MDS ID, included in every Chunk to clearly tie multiple parts together. This also differentiates between up to 16 simultaneous Mixed Data Set messages within one Group.

Number of Valid Bytes in this Message Chunk

This field contains the size of this Mixed Data Set Message Chunk in bytes including the header. The number of Message Payload UMPs in this Chunk is calculated as required to deliver the full Number of Valid Bytes in This Message Chunk field.

If **Number of Valid Bytes in This Message Chunk** is not an integer multiple of 16, then the Sender shall use pad bytes at the end of the last data payload to fill out the UMP. The pad bytes are set to zero and are reserved.

Number of Chunks in Mixed Data Set

This declares the number of Chunks expected in the data set. The Sender shall set this value to zero if the number of chunks is unknown (e.g. for streaming data). However, when the number of chunks is unknown, the final Chunk shall declare a new value for **Number of Chunks in Mixed Data Set** which matches the Chunk count value declared in the **Number of This Chunk** field.

Number of This Chunk

The Sender shall assign each Chunk of the message an incrementing Chunk count number, starting from 1.

The end of the messages is reached when (**Number of this Chunk** = **Number of Chunks in Mixed Data Set**).

See ***Section 7.9.1*** for exception cases for the ending of a Mixed Data Set.

Manufacturer ID

This field contains Manufacturer ID. The ID is encoded in a 16-bit ID (MfrID) per Section ***7.10***.

Device ID

If the **Manufacturer ID** field contains a Universal System Exclusive ID, then this **Device ID** field is intended to indicate which device in the system is supposed to respond.

The **Device ID** 0xFFFF, sometimes referred to as the ‘all call’ Device ID, is equivalent to the 0x7F value in the MIDI 1.0 Protocol and is used to indicate that all devices should respond. For more details, see Device ID in the MIDI 1.0 Specification ***[MA01]***.

If the **Manufacturer ID** is manufacturer specific, then the manufacturer may define the use of this field.

Sub ID #1

If the **Manufacturer ID** field contains a Universal System Exclusive ID, then other MMA/AMEI specifications related to that Universal System Exclusive ID define the **Sub ID #1** field. For more details, see Sub ID #1 in the MIDI 1.0 Specification ***[MA01]***.

If the Manufacturer ID is manufacturer specific, then the manufacturer may define the use of this field.

Sub ID #2

If the **Manufacturer ID** field contains a Universal System Exclusive ID, other MMA/AMEI specifications related to that Universal System Exclusive ID define the **Sub ID #2** field. For more details, see Sub ID #2 in the MIDI 1.0 Specification ***[MA01]***.

If the **Manufacturer ID** is manufacturer specific, then the manufacturer may define the use of this field.

7.9.1 End of Mixed Data Set

Under normal circumstances the Mixed Data Set ends and the current MDS ID is closed when (**Number of this Chunk** = **Number of Chunks in Mixed Data Set**).

If Sender runs out of data or is otherwise unable to complete a data set before reaching the expected end of the Mixed Data Set, then the Sender shall terminate the data set and close the MDS ID in either of the following two ways:

- If the Sender knows that the data in this Mixed Data Set Message Chunk is valid, then this final Chunk shall declare a new value for the **Number of Mixed Data Set** Message Chunks in Mixed Data Set which matches the Number of this Chunk.
- If the Sender does NOT know that the data already sent in this Mixed Data Set Message is valid, then for this final Chunk it shall set the **Number of this Chunk** field to Zero.

If the Sender runs out of payload data before sending a final Mixed Data Set Message Chunk as above, then the Sender should send one more Mixed Data Set Message Chunk with **Number of Bytes in This Message Chunk** set to 16 (header bytes only) and set the **Number of Chunks in Mixed Data Set** and **Number of this Chunk** fields as defined above.

Note: Mixed Data Set Messages cannot be translated to Non-UMP MIDI 1.0 Systems. Therefore, there are no prohibitions against interspersing other message UMPs, as there are with the 7-bit System Exclusive Messages described in Section 7.7.1.

7.10 16-Bit Manufacturer IDs

The Manufacturer ID used in System Exclusive 8 and Mixed Data Set messages encodes the 7-bit and 21-bit Manufacturer IDs and Universal System Exclusive IDs from the MIDI 1.0 Protocol into a 16-bit ID (MfrID). MMA/AMEI might define other messages in the future which also use this format.

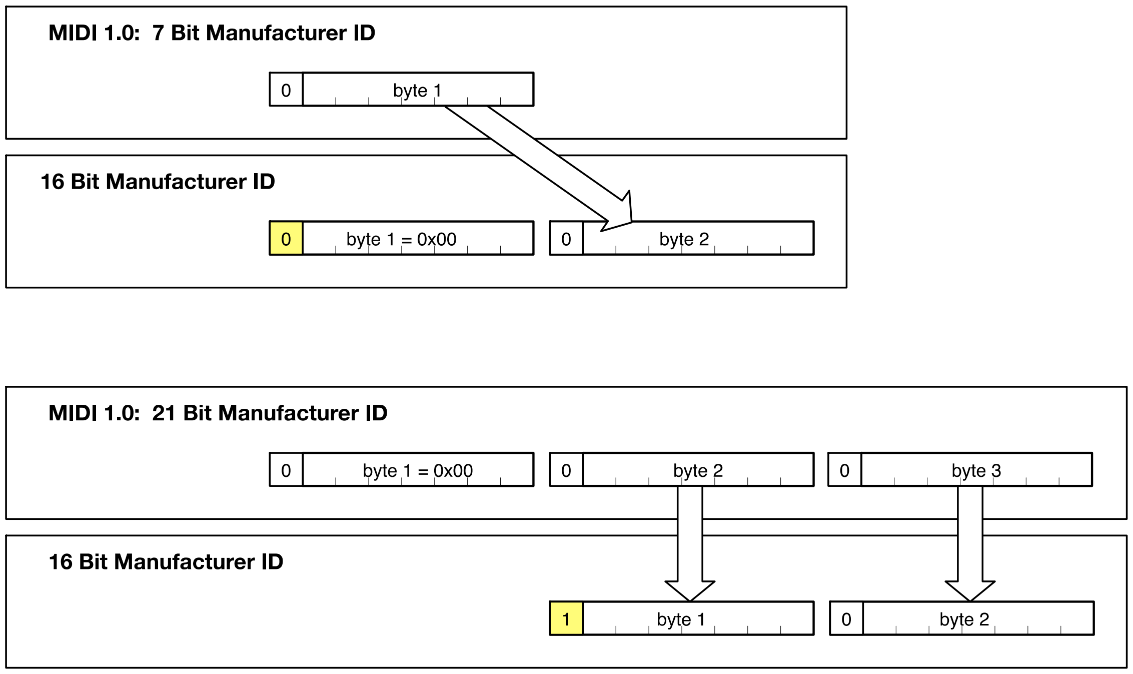


Figure 87 Manufacturer ID Translations

7-Bit (1-byte) Manufacturer IDs

All MIDI 1.0 style 7-bit Manufacturer IDs are expanded to 16 bits, with the highest byte set to 0x00 followed by the lowest byte set to same value as in the MIDI 1.0 format.

21-Bit (3-byte) Manufacturer IDs

All MIDI 1.0 style 21-bit Manufacturer IDs have their highest byte set to 0x00. This first byte 0x00 is replaced by the most significant bit set high in the lowest byte of the new format. The 7-bit values from byte 2 and byte 3 of the 21-bit Manufacturer ID are copied into the highest and lowest byte of the new format, respectively.

Special IDs

Special ID values are encoded into the 16-bit format following the format as shown above for all other 7-bit Manufacturer IDs:

Table 20 16-Bit Values for 7-Bit Special IDs

Special ID	7-Bit Value	16-Bit Value
Non-Commercial / Research No Public Release	0x7D	0x007D
Universal System Exclusive Non-Real Time	0x7E	0x007E
Universal System Exclusive Real Time	0x7F	0x007F
Reserved	0x00	0x0000

Example Conversion Code

Convert MIDI 1.0 Protocol 3-byte Sys Ex ID (MFID_1, MFID_2, MFID_3) to MIDI 2.0 Protocol 16-bit format (MfrID)

```
if (MFID_1 == 0x00)
  // 3-Byte format: use Bytes 2 & 3, and set high bit
  MfrID = 0x8000 | (MFID_2 << 8) | MFID_3;
else
  // 1-Byte format: use Byte 1 only
  MfrID = MFID_1;
```

Convert MIDI 2.0 Protocol 16-bit MfrID to three MIDI 1.0 Protocol Sys Ex ID bytes (MFID_1, MFID_2, MFID_3)

```
if ((MfrID & 0x8000) == 0) {
  // 1-Byte format
  MFID_1 = (MfrID & 0x007F);
  MFID_2 = 0;
  MFID_3 = 0;
} else {
  // 3-Byte format
  MFID_1 = 0;
  MFID_2 = ((MfrID & 0x7F00) >> 8;
  MFID_3 = (MfrID & 0x007F);
}
```

Table 21 MIDI 2.0 MfrID Conversions of Example Existing Manufacturer IDs

Manufacturer	MIDI 1.0 1- or 3-Byte ID			mfid_32	MIDI 2.0 16-bit MfrID		
	MFID_1	MFID_2	MFID_3		MfrID	MfrID_hi	MfrID_lo
Moog	0x04	–	–	0x00040000	0x0004	0x00	0x04
Midi 9	0x09	–	–	0x00090000	0x0009	0x00	0x09
Yamaha	0x43	–	–	0x00430000	0x0043	0x00	0x43
Mark of the Unicorn	0x00	0x00	0x3b	0x0000003b	0x803b	0x80	0x3b
imitone	0x00	0x02	0x13	0x00000213	0x8213	0x80	0x3b
Sensel Inc	0x00	0x02	0x1d	0x0000021d	0x821d	0x82	0x1d
Samick	0x00	0x20	0x25	0x00002025	0xa025	0xa0	0x25
Native Instruments	0x00	0x21	0x09	0x00002109	0xa109	0xa1	0x09
Bome Software	0x00	0x21	0x32	0x00002132	0xa132	0xa1	0x32

Appendix A: MIDI 2.0 Registered Per-Note Controllers

The following table lists the MIDI 2.0 Registered Per-Note Controller numbers whose application/function has been defined.

Table 22 MIDI 2.0 Registered Per-Note Controllers

RPNC Number	Registered Per-Note Controller Name	Default	Reference
0	Reserved	–	–
1	Modulation	–	–
2	Breath	–	–
3	Pitch 7.25	–	***Section 7.4.15.2***
4–6	Reserved	–	–
7	Volume	–	–
8	Balance	–	–
9	Reserved	–	–
10	Pan	–	–
11	Expression	–	–
12–69	Reserved	–	–
70	Sound Controller 1	Sound Variation	–
71	Sound Controller 2	Timbre/Harmonic Intensity	–
72	Sound Controller 3	Release Time	–
73	Sound Controller 4	Attack Time	–
74	Sound Controller 5	Brightness	–
75	Sound Controller 6	Decay Time	MMA RP-021 ***[MA04]***
76	Sound Controller 7	Vibrato Rate	
77	Sound Controller 8	Vibrato Depth	
78	Sound Controller 9	Vibrato Delay	
79	Sound Controller 10	Undefined	
80–90	Reserved	–	–
91	Effects 1 Depth	Reverb Send Level	MMA RP-023 ***[MA05]***
92	Effects 2 Depth (formerly Tremolo Depth)	–	–
93	Effects 3 Depth	Chorus Send Level	MMA RP-023 ***[MA05]***

94	Effects 4 Depth *(formerly Celeste [Detune] Depth)*	–	–
95	Effects 5 Depth *(formerly Phaser Depth)*	–	–
96 and above	Reserved	–	–

Appendix B: Special Control Change Messages

B.1 Channel Mode Messages: Applicable Channels

MIDI has eight Channel Mode Messages. These are special purpose Control Change messages.

- CC#120 All Sound Off
- CC#121 Reset All Controllers
- CC#122 Local Control
- CC#123 All Notes Off
- CC#124 Omni Off
- CC#125 Omni On
- CC#126 Mono On (Poly Off)
- CC#127 Poly On (Omni Off)

The UMP Format preserves the fundamental definition of these messages, with added clarifications for implementation as follows below.

The MIDI 1.0 Specification ***[MA01]*** states: "These messages are recognized only when sent on the Basic Channel to which a Receiver is assigned, regardless of the current mode."

In UMP implementations, Channel Mode messages are defined the same as in the MIDI 1.0 Specification ***[MA01]*** within a single Group. Functionality of Mode Messages received in one Group does not apply to Channels in any other Group in the device.

B.2 Reset All Controllers

The MIDI 2.0 Protocol has newly defined controller types. The function of the Reset All Controllers message remains as defined by the MIDI 1.0 Specification ***[MA01]***.

The following new Per-Note Controllers are NOT reset by the Reset All Controllers message:

- MIDI 2.0 Registered Per-Note Controllers
- MIDI 2.0 Assignable Per-Note Controllers
- MIDI 2.0 Per-Note Pitch Bend

Appendix C: Using MIDI 2.0 Per-Note Messages

The Per-Note Messages of the MIDI 2.0 Protocol (Poly Aftertouch, MIDI 2.0 Per-Note Registered Controllers, MIDI 2.0 Per-Note Assignable Controllers, and MIDI 2.0 Per-Note Pitch Bend) bring expanded expression beyond the MIDI 1.0 Protocol. But the assumed statefulness of MIDI controllers, now at the Per-Note level, brings some new challenges. Per-Note Controllers are shared between all notes that share the same Note Number.

This appendix examines in depth three implementation options for Per-Note Controllers:

- Shared Per-Note Controllers: Useful for some traditional MIDI instruments, used in a manner similar to Poly Pressure in the MIDI 1.0 Protocol.
- With Per-Note Management Message: Enables increased Per-Note expression capability.
- Fully Independent Control with Note Number Rotation mechanism, Per-Note Pitch mechanisms, and Per-Note Management message: Useful for multitouch devices that allow multiple simultaneous notes on the same pitch.

C.1 Shared Per-Note Controllers

For the simplest implementation of Per-Note Controllers, notes of the same Note Number share Per-Note Controllers. ***Figure 88*** shows a typical example where the trailing envelope of Note A shares the Per-note Controllers that are also controlling Note B.

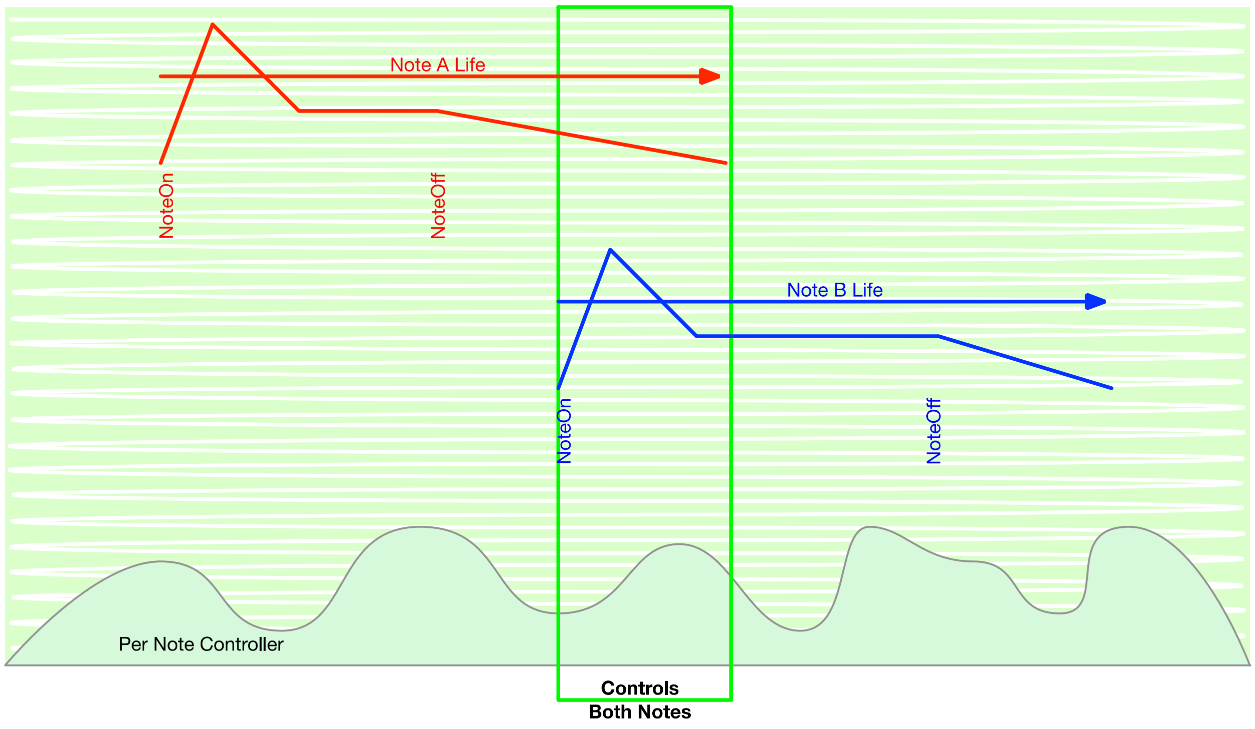


Figure 88 Two Notes of Same Note Number Share Per-Note Controllers

Per-Note Controller sharing is not problematic on some devices with traditional musical performance interfaces. This implementation has always been true for the MIDI 1.0 Protocol with Polyphonic Pressure. With Polyphonic Pressure on a synthesizer keyboard, it is assumed that when you stop playing a note, Pressure value has returned to a value of zero.

However, this can be a limitation for some instruments which allow multitouch and separate expression on more than one simultaneously sounding note on the same Note Number. Sequencing/editing in software might also suffer from problems when notes overlap.

C.2 Using a MIDI 2.0 Per-Note Management Message Before Note On to Reallocate Per-Note Expression

To enable separate control of notes on the same Note Number, the Sender inserts a Per-Note Management message with Detach bit set before any new Note On message (see ***Figure 89***). The Receiver uses the Per-Note Management message to detach Per-Note Controllers from any current sounding Notes of the target Note Number and reset the assignment to the next following Note of the same Note Number.

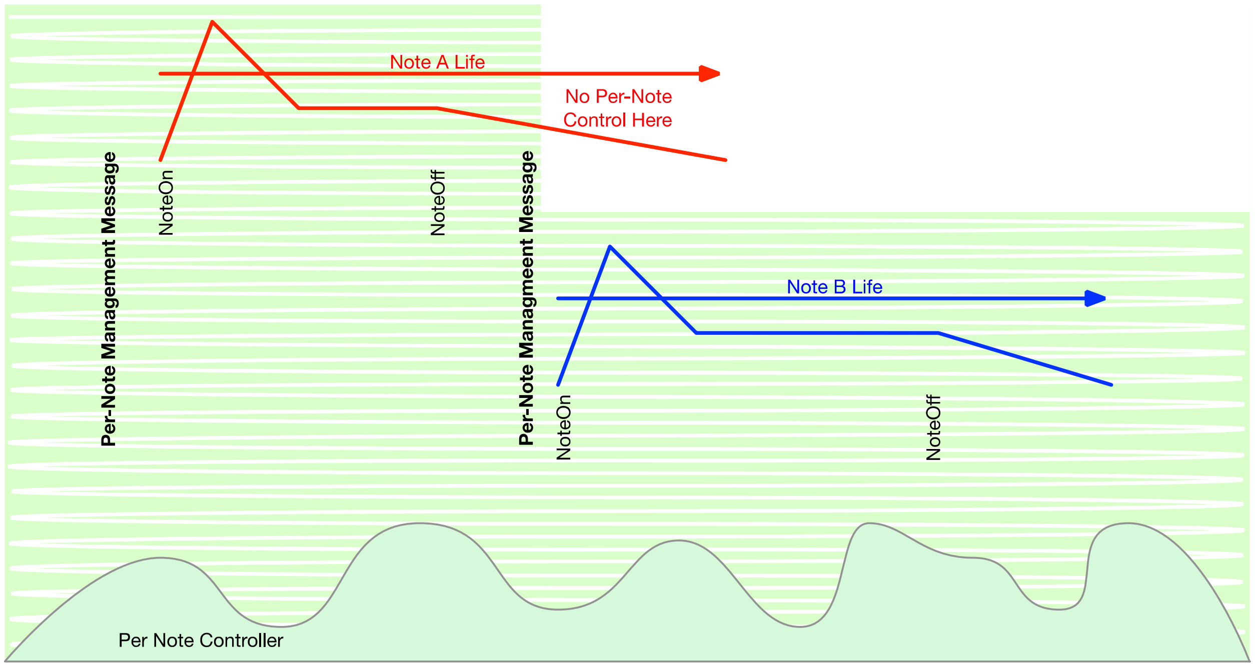


Figure 89 Only the Note After the Per-Note Management Message has Per-Note Control

Following the Per-Note Management message, Per-Note controllers are used to set up the upcoming note or to control it while it is sounding. Note A is no longer controlled by Per-Note Controllers.

Note A might continue to sound while keeping the last known state of controllers that occurred before the Per-Note Management message.

Note B might optionally reset Per-Note Controller **Values** upon receiving the Per-Note Management message. In this case, if no other Per-Note Controllers are sent between the Per-Note Management and the next Note One, the new Note B uses its default values of all Per-Note Controllers.

mt=4	group	1	1	1	1	channel	r	note number	option flags	D	S
reserved											

Figure 90 D and S Fields in MIDI 2.0 Per-Note Management Message

D: Detach Per-Note Controllers from previous sounding Note(s)

S: Reset (Set) Per-Note Controllers to default values

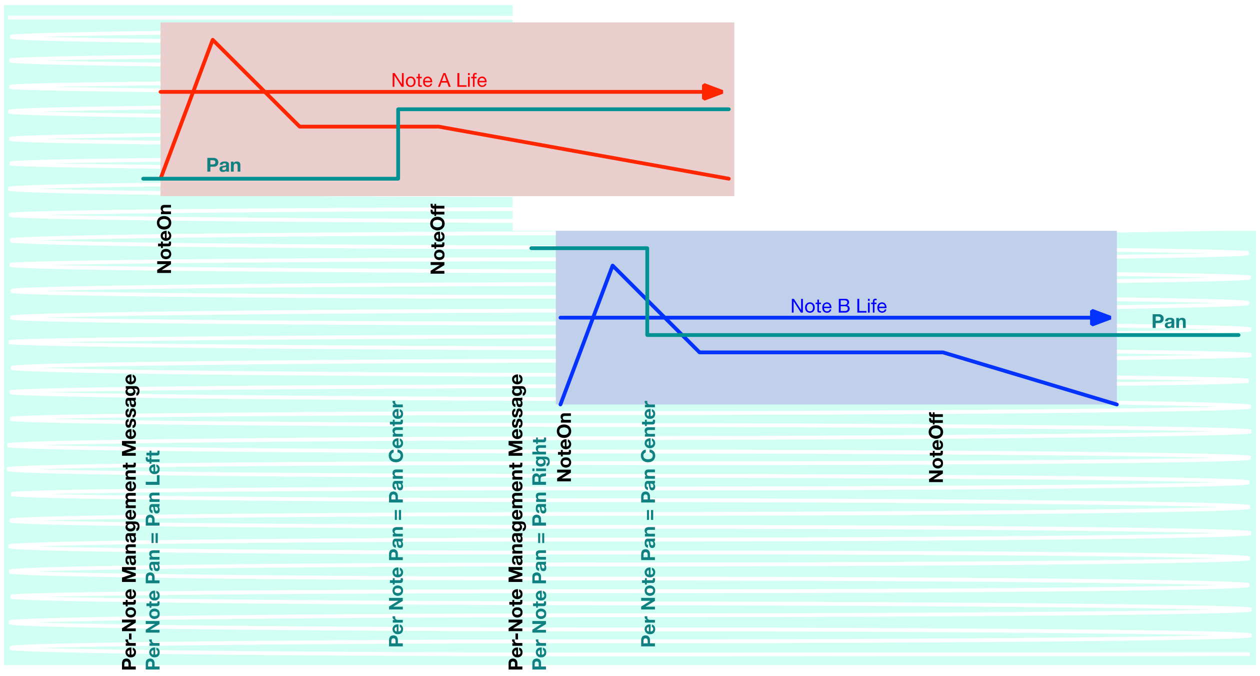

Note A Life
Pan
NoteOn
NoteOff
Note B Life
Pan
NoteOn
NoteOff
Per-Note Management Message
Per Note Pan = Pan Left
Per Note Pan = Pan Center
Per-Note Management Message
Per Note Pan = Pan Right
Per Note Pan = Pan Center

Figure 91 Per-Note Management Example with Per-Note Pan

```
Per-Note Management @Note Number 60
Per-Note Controller @Note Number 60, Pan Left
Note On #60
Per-Note Controller @Note Number 60, Pan Center
Note Off #60

Per-Note Management @Note Number 60
Per-Note Controller @Note Number 60, Pan Right
Note On #60
Per-Note Controller @Note Number 60, Pan Center
Note Off #60
```

C.3 Using Note Number Rotation, Per-Note Pitch, and Per-Note Management Message for Independent Per-Note Expression

A MIDI 2.0 Protocol Sender can have fully independent control over individual Notes, even applied to simultaneous Notes on the same pitch. MIDI Polyphonic Expression (MPE) on the MIDI 1.0 Protocol uses a Channel Rotation mechanism for this kind of flexible expressive control with up to 16 notes of polyphony. In the MIDI 2.0 Protocol, a Note Number Rotation mechanism can replace the Channel Rotation mechanism for some applications. This improves on MPE by utilizing only a single MIDI Channel while providing polyphony of up to 128 notes.

Using the MIDI 2.0 Protocol, the Sender plays Notes with added Pitch data. The added Pitch data overrides any notion of pitch that might be implied by the Note Number field in the Note On, Note Off, and Per-Note Controllers. Note Number loses any implication of pitch and only functions as a Note Index.

The Pitch data for each note can come from two different sources:

- Registered Per-Note Controller #3: Pitch 7.25 (PNCC#3)
- Note On With Attribute #3 Pitch 7.9 (AttrPitch7.9)

In either case, a HCU field in the message sets a pitch as a Note Number of the same value might otherwise imply.

The Sender assigns a Note Number to each note it sends in a rotating fashion. It might try to use the same value for Note Number as in the Pitch data whenever feasible to serve translation to the MIDI 1.0 Protocol. Or it might rotate through all 128 Note Number on a Least Recently Used basis to more-robustly avoid Per-Note controller overlap. Or it might use any other scheme it sees fit to assign Note Numbers.

Note Numbers are reused for notes of various pitch. In order guarantee that a new note does not adopt any state from controllers previously addressed to that Note Number, the Sender sends Per-Note Management message before sending every Note On message.

Receiver Implementation

Receivers do not necessarily need to know that a rotation scheme is used. They shall respond to the two standard methods of Pitch control listed above. Many Receivers already do this, to support alternate scales or flexible microtuning. Receivers shall also implement the Per-Note Management message.

Note: Receivers that select samples for playing a note based on Note Number might choose to instead select samples based on the first 7 bits of the pitch data in the last valid Registered Per-Note Controller #3: Pitch 7.25 or in the Note On With Attribute #3 Pitch 7.9.

Sender Implementation

Senders have two choices of source for Pitch Data for each Note, described below as Method 1 and Method 2. The choice between the two methods will largely be determined by the Sender's user performance/controller interface.

Method 1: Sender Using Registered Per-Note Controller #3: Pitch 7.25 (PNCC#3)

Some Sender devices' performance interfaces are designed to provide continuous control over pitch for every note for the whole life cycle of the note. Such controllers should use the Registered Per-Note Controller #3: Pitch 7.25 (PNCC#3) to achieve that continuous control.

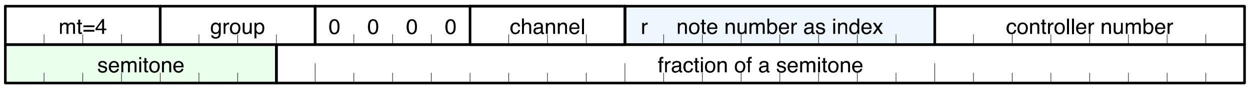


Figure 92 MIDI 2.0 Registered Per-Note Controller Message with Controller #3 (Pitch 7.25)

Such devices can then use this pitch controller with Note Rotation and Per-Note Management messages to achieve independent expressive control over each note. The message sequence for two successive notes that both play a Middle C might look like this:

```
Per-Note Management @Note Number 00
PNCC#3 @Note Number 00 Set Pitch 60.0
Note On #00 (Pitch sounds as 60.0)
Several other Per-Note Controllers @Note Number 00
Note Off #00

Per-Note Management @Note Number 01
PNCC#3 @Note Number 01 Set Pitch 60.0
Note On #01 (Pitch sounds as 60.0)
Several other Per-Note Controllers @Note Number 01
Note Off #01
```

Because the two notes of the same pitch use different Note Numbers, they can even overlap in time. Multiple notes can sound simultaneously on the pitch of Middle C, each with its own dedicated set of Per-Note Controllers.

Method 2: Sender Using Note On With Attribute #3 Pitch 7.9 (AttrPitch7.9)

Some Sender devices' performance interfaces are designed to provide continuous control over various parameters, but pitch is generally constant for the whole life cycle of the note. Such controllers can use the Registered Per-Note Controller #3: Pitch 7.25 (PNCC#3) as described above. Or such devices can use Note On messages with AttrPitch7.9 with Note Rotation to achieve independent expressive control over each note. This alternate mechanism is only suited to applications that do not need to use the Note On Attribute field for any other purpose.

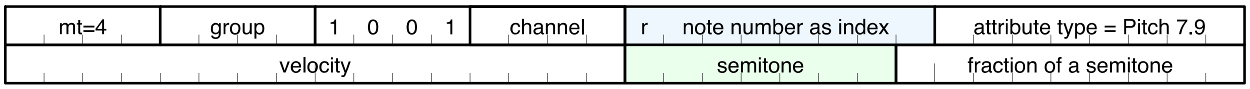


Figure 93 MIDI 2.0 Note On Message with Attribute #3 (Pitch 7.9)

The message sequence of two successive notes that play a Middle C might look like this:

```
Per-Note Management @Note Number 00
Note On #00 with AttrPitch7.9 = 60.0
Several Per-Note Controllers @Note Number 00
Note Off #00

Per-Note Management @Note Number 01
Note On #01 with AttrPitch7.9 = 60.0
Several Per-Note Controllers @Note Number 01
Note Off #01
```

Because the two notes use different Note Numbers, they can even overlap in time. Multiple notes can sound simultaneously on the pitch of Middle C, each with its own dedicated set of Per-Note Controllers.

Appendix D: Translation: MIDI 1.0 and MIDI 2.0 Messages

This section explains how MIDI 1.0 Protocol messages are translated to MIDI 2.0 Protocol messages and vice versa, including translation between data fields of different sizes. Proper translation is crucial for preserving intended functionality across a MIDI 1.0 Protocol / MIDI 2.0 Protocol boundary.

There is one strict set of translation rules, the Default Translation Mode, which is compliant with the MIDI 2.0 Specifications. To be compliant, a device must be able to operate in the Default Translation Mode where it shall follow every rule in ***Appendix D.1*** through ***Appendix D.3*** of this specification.

Devices may optionally make Alternate Translation Modes (i.e., using different translation rules) available as detailed in ***Appendix D.4***.

D.1 Data Value Translations

In the MIDI 1.0 Protocol, data values are represented by 7-bit or 14-bit numbers. In the MIDI 2.0 Protocol, data values are represented by 16-bit or 32-bit numbers. This section explains how to convert between these different resolutions when translating MIDI 1.0 Protocol messages to MIDI 2.0 Protocol messages and vice versa.

Also see the MIDI 2.0 Bit Scaling and Resolution specification ***[MA08]***. If there is a discrepancy between this specification and ***[MA08]***, then ***[MA08]*** should be considered authoritative.

D.1.1 Overview

Default translation of data values shall always scale the value to the full range. For example, this ensures that continuous controllers always go from minimum to maximum. Discrete enumerations are usually encoded by dividing the range into sections, where each section represents one enumeration value. This encoding also survives data scaling (as long as the number of sections does not exceed the data range).

When translating MIDI Protocol 1.0 values, translation should be lossless, in the sense that translating a MIDI 1.0 Protocol message to a MIDI 2.0 Protocol message and then back to the MIDI 1.0 Protocol should yield the same or equivalent data as the original MIDI 1.0 Protocol message. Translating MIDI 2.0 Protocol messages to the MIDI 1.0 Protocol and back to the MIDI 2.0 Protocol will usually result in quantization, due to the lower resolution of the MIDI 1.0 Protocol.

D.1.2 Core Rules

- Minimum/Lowest value in each data field is translated to Minimum/Lowest
- Maximum/Highest value in each data field is translated to Maximum/Highest

  For example, a 7-bit value of 127 is translated to a 16-bit value of 65535.

- Center Value always translates to Center Value

```
Center = TRUNC( (Highest + 1) / 2 )
```

Table 23 Center Value Examples

Value Size	Center Value	
	Hex	Binary
7 bits	0x40	8’b 01000000
14 bits	0x2000	16’b 00100000 00000000
8 bits	0x80	8’b 10000000
16 bits	0x8000	16’b 10000000 00000000
32 bits	0x80000000	32’b 10000000 00000000 00000000 00000000

- When upscaling, smoothly distribute low resolution values on the range of the high resolution.
- The translation algorithm shall yield the same output as the input data when translating:

  MIDI 1.0 Protocol → MIDI 2.0 Protocol → MIDI 1.0 Protocol

  Note: In some cases, translation in each direction might be performed by independent entities, and in such cases this result is not mandated.

D.1.3 Default Upscaling Method: Min-Center-Max

The Min-Center-Max algorithm for upscaling values to higher resolution works as follows:

- For values from minimum to the center, use simple bit shifting. This ensures smooth increments towards the center value. The center value remains the center value.
- Use an expanded bit-repeat scheme for the range from center to maximum. This causes the values to smoothly increase from center to maximum value.

The Min-Center-Max algorithm is the default method for upscaling values. See the MIDI 2.0 Bit Scaling and Resolution specification ***[MA08]*** for more information.

Code for the Min-Center-Max Upscaling Algorithm

(Optimized for readability, not efficiency.)

```
// power of 2, pow(2, exp)
uint32_t power_of_2(uint8_t exp)
{
    return 1 << exp; // implement integer power of 2 using bit shift
}

// preconditions: srcBits > 1, dstBits<=32, srcBits < dstBits
uint32_t scaleUp(uint32_t srcVal, uint8_t srcBits, uint8_t dstBits)
{
    uint8_t scaleBits = (dstBits - srcBits);    // number of bits to upscale
    uint32_t srcCenter = power_of_2(srcBits-1); // center value for srcBits, e.g.
                                                // 0x40 (64) for 7 bits
                                                // 0x2000 (8192) for 14 bits
    // simple bit shift
    uint32_t bitShiftedValue = srcVal << scaleBits;
    if (srcVal <= srcCenter ) {
        return bitShiftedValue;
    }
    // expanded bit repeat scheme
    uint8_t repeatBits = srcBits - 1; // we must repeat all but the highest bit
    uint32_t repeatMask = power_of_2(repeatBits) - 1;
    uint32_t repeatValue = srcVal & repeatMask; // repeat bit sequence
    if (scaleBits > repeatBits) { // need to repeat multiple times
        repeatValue <<= (scaleBits - repeatBits);
    } else {
        repeatValue >>= (repeatBits - scaleBits);
    }
    while (repeatValue != 0) {
        bitShiftedValue |= repeatValue; // fill lower bits with repeatValue
        repeatValue >>= repeatBits;     // move repeat bit sequence to next position
    }
    return bitShiftedValue;
}
```

First, the scaled value using bit shift is calculated by shifting left by the difference of the different bit sizes. If the original value is the center value or smaller, the bit shifted value is returned.

For values above the center, a `repeatValue` is calculated: it is the original value with the top 2 bits removed. So it has `repeatBits` significant bits. Finally, the `repeatValue` is used according to the Bit-Repeat scheme to fill the low order bits of the bit shifted value.

Code for Min-Center-Max Scaling Up from 7-Bit to 16-Bit

```
uint16_t scaleUp7to16(uint8_t value7) {
    uint16_t bitShiftedValue = (uint16_t)value7 << 9;
    if (value7 <= 64) {
        return bitShiftedValue;
    }
    // use bit repeat bits from extended value7
    Uint16_t repeatValue6 = (uint16_t)value7 & 0x3F;
    return bitShiftedValue
              | (repeatValue6 << 3)
              | (repeatValue6 >> 3);
}
```

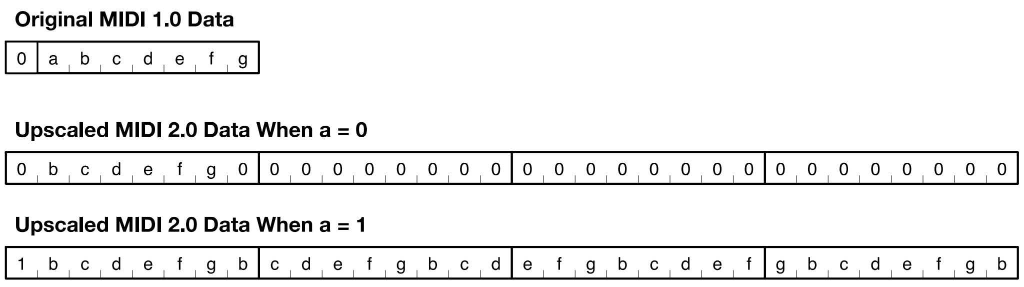

Original MIDI 1.0 Data
0 a b c d e f g
Upscaled MIDI 2.0 Data When a = 0
0 b c d e f g 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Upscaled MIDI 2.0 Data When a = 1
1 b c d e f g b c d e f g b c d e f g b c d e f g b c d e f g b

Figure 94 Value Upscaling Diagram

Numerical Examples

- 10 (0x0a) → 0x1400
- 64 (0x40) → 0x8000
- 87 (0x57) → 0xaeba
- 127 (0x7f) → 0xffff

D.1.4 Downscaling Translation Methods

For scaling a high resolution value to a value with lower resolution, simple bit shifting (i.e. cutting off the lower bits) is sufficient and accurate enough.

Code for Downscaling Algorithm

```
uint32_t scaleDown(uint32_t srcVal, uint8_t srcBits, uint8_t dstBits) {
    // simple bit shift
    uint8_t scaleBits = (srcBits - dstBits);
    return srcVal >> scaleBits;
}
```

Numerical Examples

- 0x1400 → 0x0a
- 0x8000 → 0x40

- 0xaeba → 0x57
- 0xffff → 0x7f

D.1.5 Special Considerations

Some devices assign a special meaning to Minimum and Maximum values of some properties. If a Translator is aware of a special case, then the Translator may choose to translate near-zero data values to a value of 1, and to translate near-Maximum data values to a value of (Maximum - 1).

D.2 MIDI 2.0 to MIDI 1.0 Default Translation

D.2.1 Note On/Off, Poly Pressure, Control Change

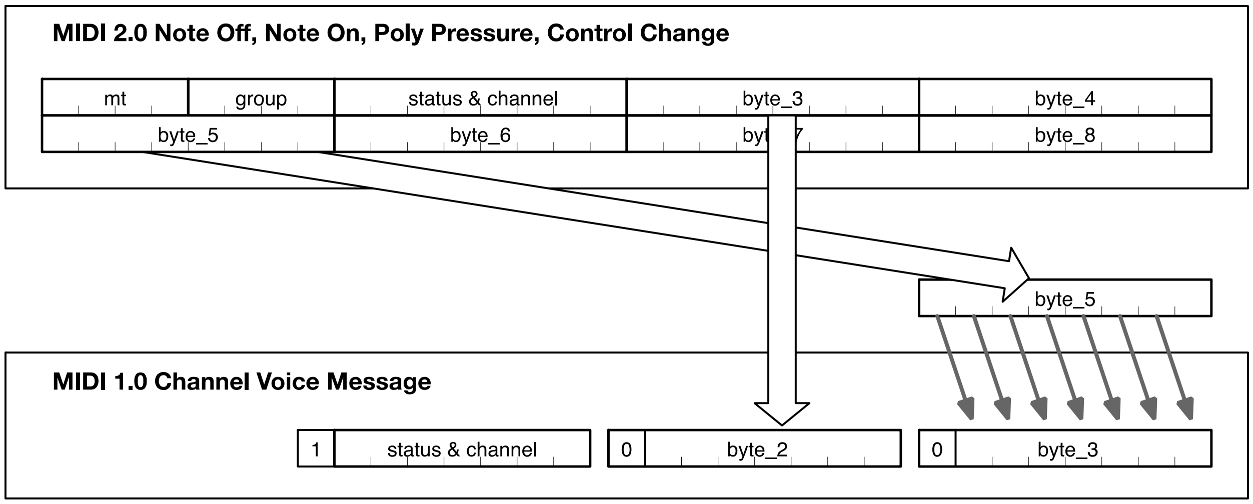


Figure 95 Translate MIDI 2.0 Note Off, Note On, Poly Pressure, and Control Change to MIDI 1.0

MIDI 2.0 Note On Velocity

The allowable Velocity range for a MIDI 2.0 Note On message is 0x0000-0xFFFF. However, depending on the chosen translation method, near-zero values can result in a MIDI 1.0 Note On with Velocity of 0x00, which has the same function as a Note Off. Therefore, if the translated MIDI 1.0 value of the Velocity is 0x00, replace the value with 0x01. If translation to MIDI 1.0 High Resolution Velocity Prefix (using Control Change #88, see MMA/AMEI CA#031 *[MA03]*) is supported, then the minimum combined value for the 14-bit velocity is 0x0080.

D.2.2 Channel Pressure

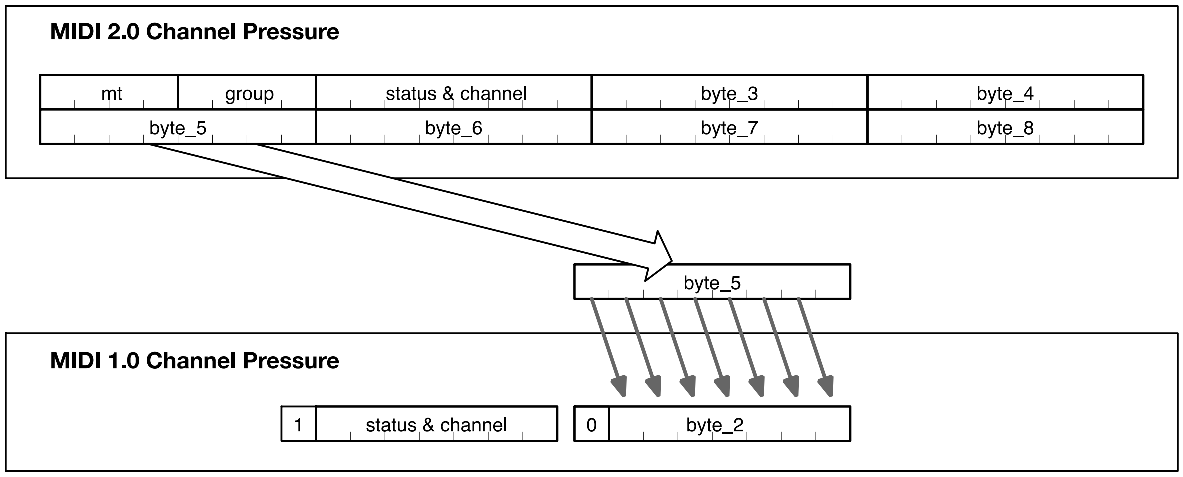


Figure 96 Translate MIDI 2.0 Channel Pressure to MIDI 1.0

D.2.3 Assignable Controllers (NRPN) and Registered Controllers (RPN)

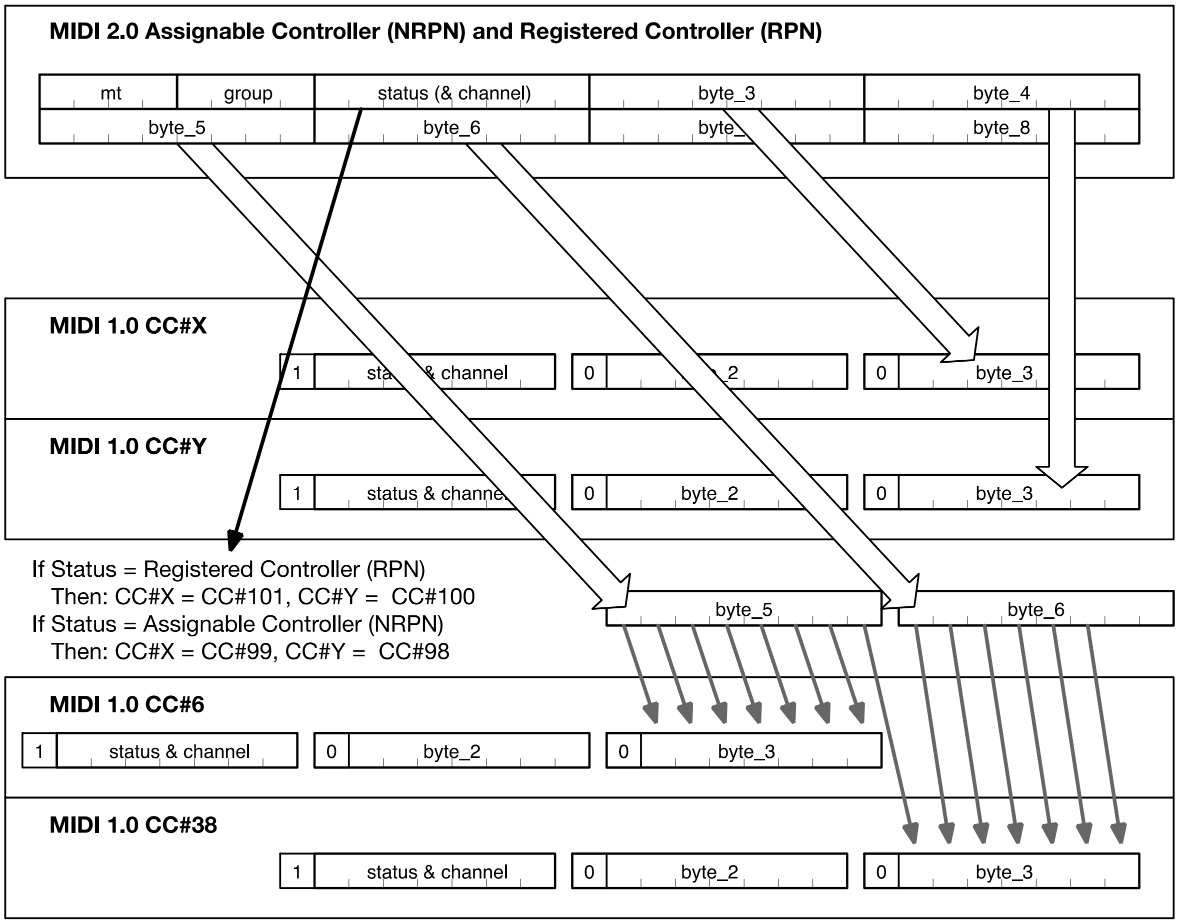


Figure 97 Translate MIDI 2.0 Assignable (NRPN) and Registered (RPN) Controller to MIDI 1.0

Assignable Controllers and Registered Controllers

Assignable Controllers and Registered Controllers are singular messages in the MIDI 2.0 Protocol. When translating to the MIDI 1.0 Protocol, each message generates a sequence of four MIDI 1.0 Protocol messages.

D.2.4 Program Change and Bank Select

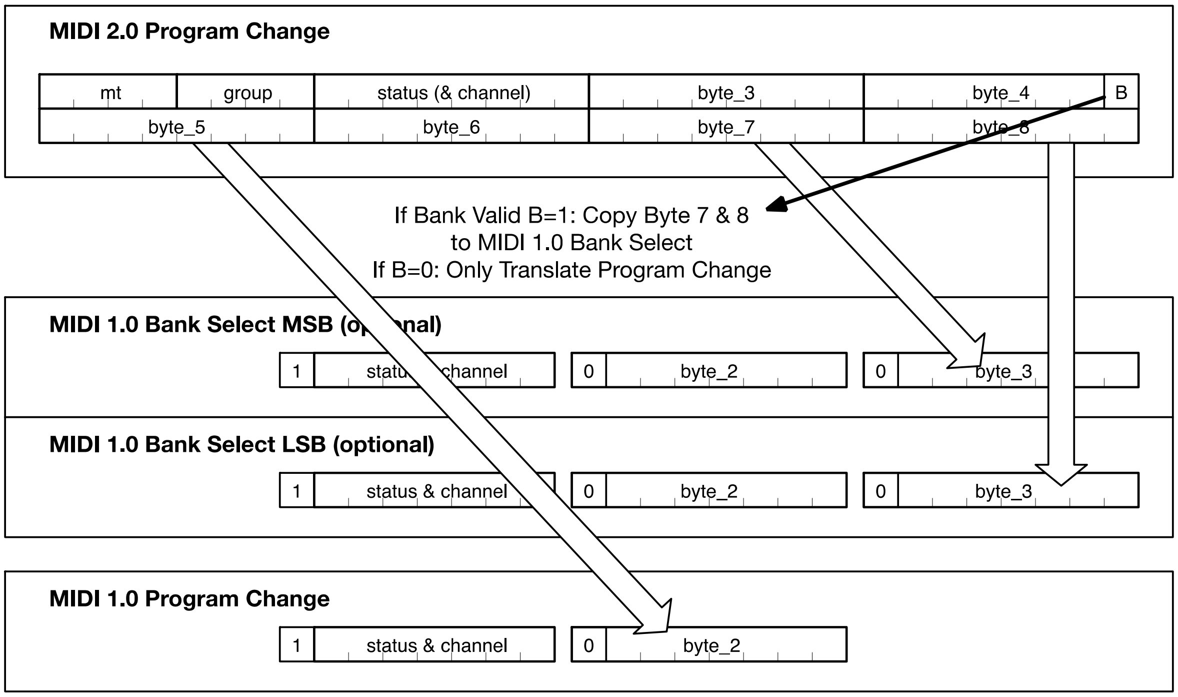


Figure 98 Translate MIDI 2.0 Program Change to MIDI 1.0

Program Change & Bank Select

Program Change and Bank Select are one message in the MIDI 2.0 Protocol. When translating to the MIDI 1.0 Protocol they generate up to three messages:

- If the value of the Bank Valid (B) bit is 0, then only translate the Program Change value to a MIDI 1.0 Protocol Program Change message.
- If the value of the Bank Valid bit is 1, then translate to three MIDI 1.0 Protocol messages in the following order:

  A. Bank Select MSB

  B. Bank Select LSB

  C. Program Change

D.2.5 Pitch Bend

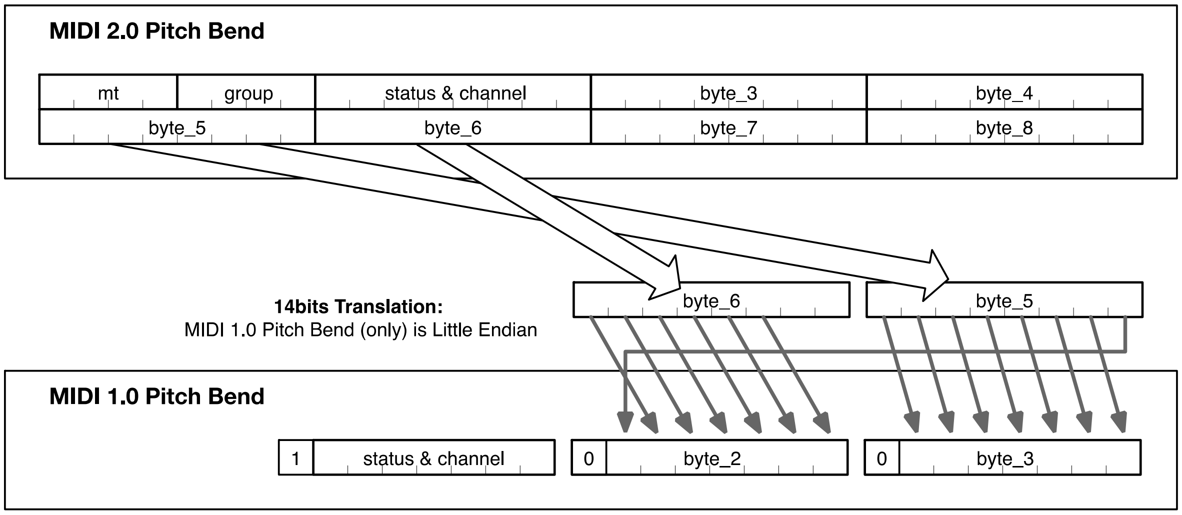


Figure 99 Translate MIDI 2.0 Pitch Bend to MIDI 1.0

Note that Pitch Bend values in the MIDI 1.0 Protocol are presented as Least Significant Byte (LSB) before Most Significant Byte (MSB) (little-endian).

D.2.6 System Messages

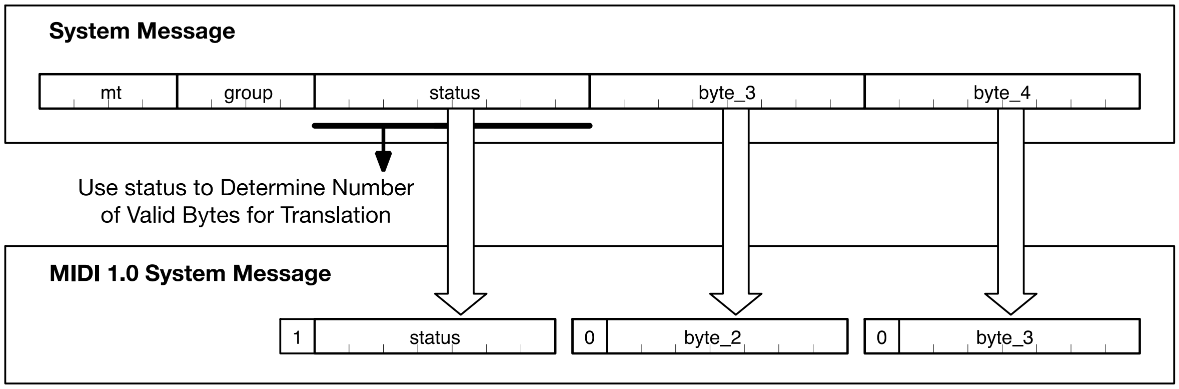


Figure 100 Translate MIDI 2.0 System Message to MIDI 1.0

D.2.7 System Exclusive

When translating System Exclusive Messages from the MIDI 2.0 Protocol to the MIDI 1.0 Protocol, all the data bytes from the whole message (often spanning multiple UMPs) are placed between a starting Status Byte of 0xF0 and an ending Status byte of 0xF7.

Example:

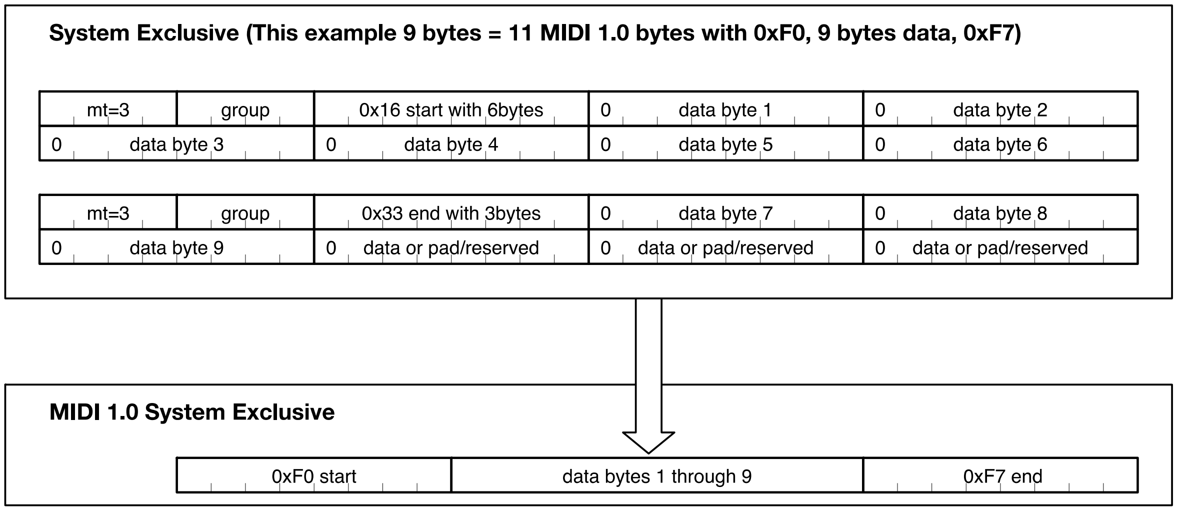


Figure 101 Translate MIDI 2.0 System Exclusive to MIDI 1.0

D.2.8 Messages That Cannot Be Translated to MIDI 1.0

The following MIDI 2.0 Protocol messages have no equivalent messages in the MIDI 1.0 Protocol:

- Relative Registered Controllers
- Relative Assignable Controllers
- Per-Note Controllers
- Per-Note Management
- Per-Note Pitch Bend

As a result, the Default Translation does not address these MIDI 2.0 Protocol Messages. However, translations for these MIDI 2.0 Protocol Messages may be implemented using Alternate Translation Modes (see Section ***D.4***).

D.2.9 Messages That Cannot Be Translated to Non-UMP MIDI 1.0 Systems

When not using the UMP Format, the following MIDI 2.0 Protocol messages shall not be used with MIDI 1.0:

- System Exclusive 8
- Mixed Data Set
- Utility Messages

D.3 MIDI 1.0 to MIDI 2.0 Default Translation

D.3.1 Note On/Off

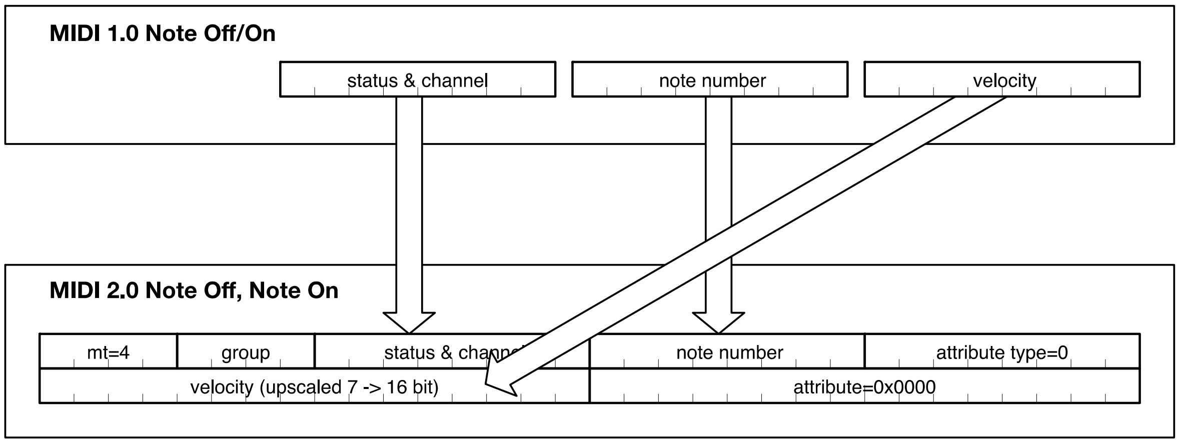


Figure 102 Translate MIDI 1.0 Note On and Note Off to MIDI 2.0

MIDI 1.0 Note On and Note Off

A MIDI 1.0 Protocol Note On message with a Velocity of 0x00 is special (i.e., is equal to Note Off), and shall be translated to a MIDI 2.0 Protocol Note Off message with Velocity 0x8000. For Velocity values from 1-127, use the upscaling method described in Section D.1.3.

MIDI Velocity = 0x01: translates to 0x0200

Attribute Type and Attribute Value: When MIDI 1.0 Protocol Note On and Note Off messages translate to MIDI 2.0 Protocol Note On and Note Off messages, the Attribute Type shall be set to 0x00 and the Attribute Value shall be set to 0x0000, unless a MIDI-CI Profile specification that is in effect specifies a different translation for the Attribute Type and Attribute Value fields.

D.3.2 Poly Pressure

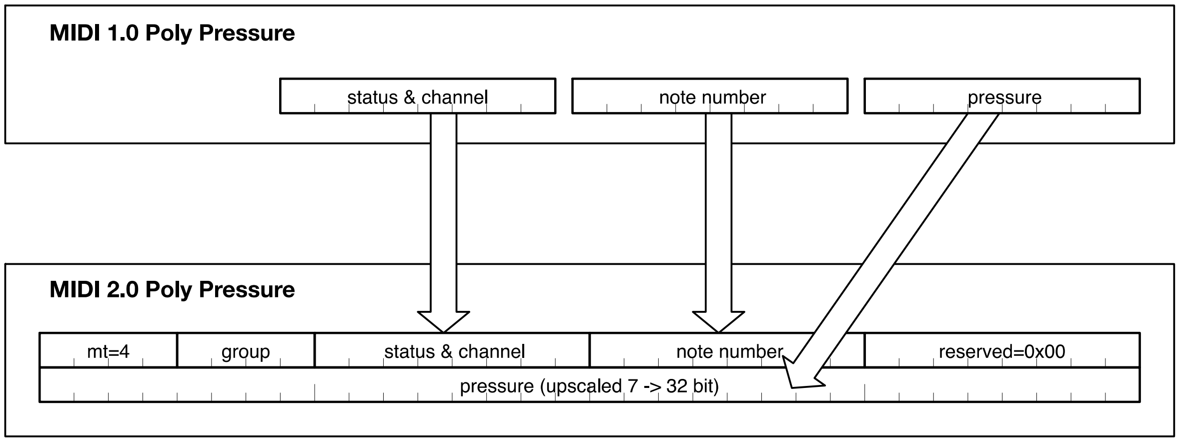


Figure 103 Translate MIDI 1.0 Poly Pressure to MIDI 2.0

D.3.3 Control Change, RPN, and NRPN

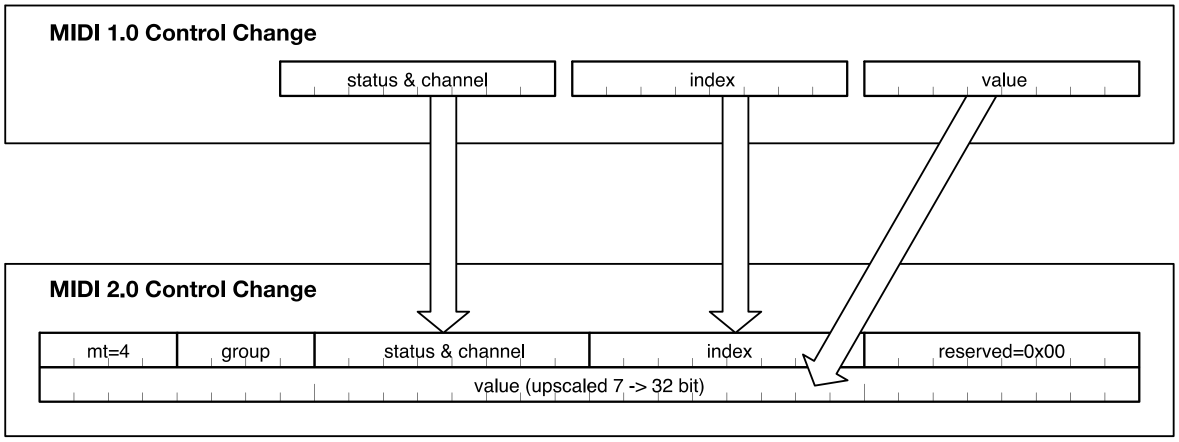


Figure 104 Translate MIDI 1.0 Control Change to MIDI 2.0

14 Bit Control Change Messages

14 Bit Control Change Messages made up of a MSB (1-31) and an LSB (33-63) value should not be merged and translated into one 32 bit value. These messages should remain as 2 independent messages.

MIDI 1.0 Increment and Decrement Message

MIDI 1.0 Protocol Increment (CC 96) and Decrement (CC 97) messages are translated to Control Change messages in the MIDI 2.0 Protocol. They have no RPN/NRPN related function in the MIDI 2.0 Protocol. They should not be translated to MIDI 2.0 Relative Registered Controller and Relative Assignable Controller Messages.

Control Change Messages for RPN/NRPN

Individual use of controllers CC 6, 38, 98, 99, 100, and 101 do not translate to the MIDI 2.0 Protocol, unless they are properly formed RPN/NRPN messages. The Default Translation shall hold the latest values for controllers CC 6, 98, 99, 100, and 101. An RPN/NRPN should be sent when one of the following occurs:

- a CC 38 is received
- a subsequent CC 6 is received
- a CC 98, 99, 100, and 101 is received, indicating the last RPN/NRPN message has ended and a new one has started

Note: As some MIDI 1.0 Devices do not transmit CC 38, some translators may wish to include a short timeout after receiving a CC 6. if any of the above triggers do not occur, to avoid loss of RPN.NRPN messages.

RPN/NRPN Null Function, where both the MSB and LSB is set to 0x7F, is not translated.

Then, if the Translator has all the data needed to make a valid RPN or NRPN, it shall send the MIDI 2.0 Protocol message as follows:

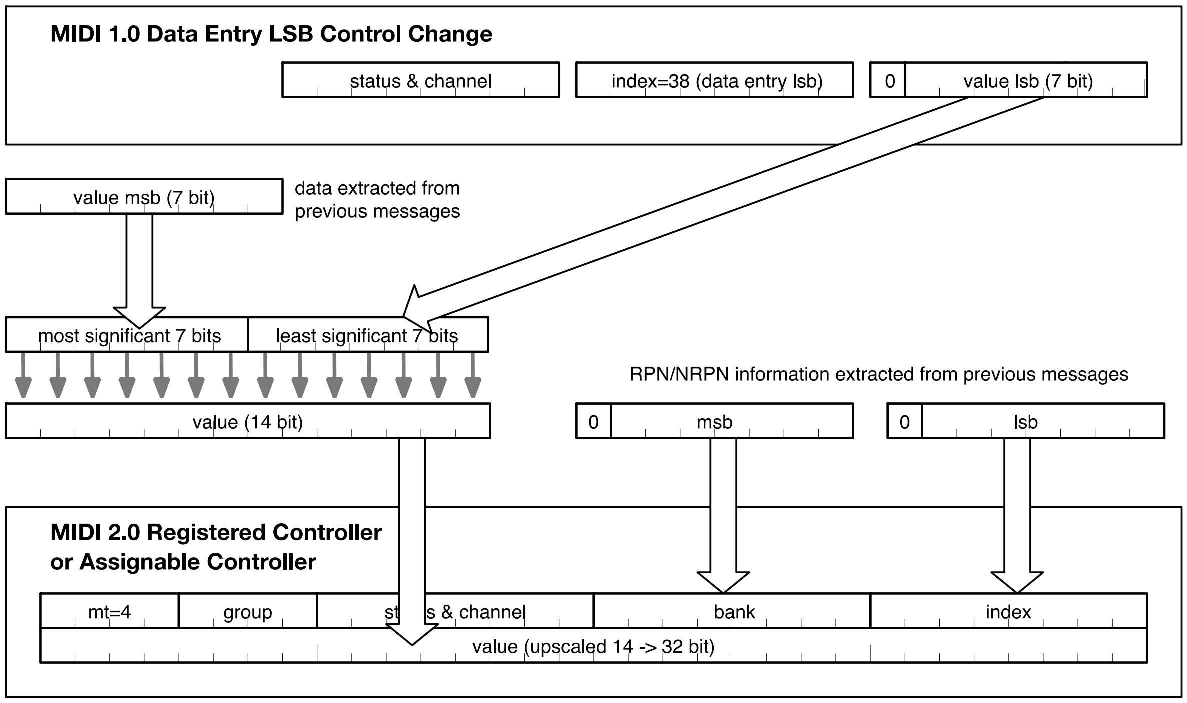

MIDI 1.0 Data Entry LSB Control Change
status & channel
index=38 (data entry lsb)
0
value lsb (7 bit)
value msb (7 bit)
data extracted from previous messages
most significant 7 bits
least significant 7 bits
value (14 bit)
RPN/NRPN information extracted from previous messages
0
msb
0
lsb
MIDI 2.0 Registered Controller or Assignable Controller
mt=4
group
status & channel
bank
index
value (upscaled 14 -> 32 bit)

Figure 105 Translate MIDI 1.0 Data Entry LSB Control Change to MIDI 2.0

Bank Select Control Change

Individual use of controllers CC 0 and CC 32 shall not translate to the MIDI 2.0 Protocol, unless they are used in a MIDI 2.0 Protocol Program Change message with the Bank Valid bit set.

D.3.4 Program Change and Bank Select

When translating MIDI 1.0 Protocol Program Change Messages to the MIDI 2.0 Protocol, include the current valid Bank Select values in the MIDI 2.0 Protocol Program Change message. If there is no current Bank Select value associated with the Program Change, then in the MIDI 2.0 Protocol message set the Bank Valid bit to 0 and fill the Bank Select fields with zeroes.

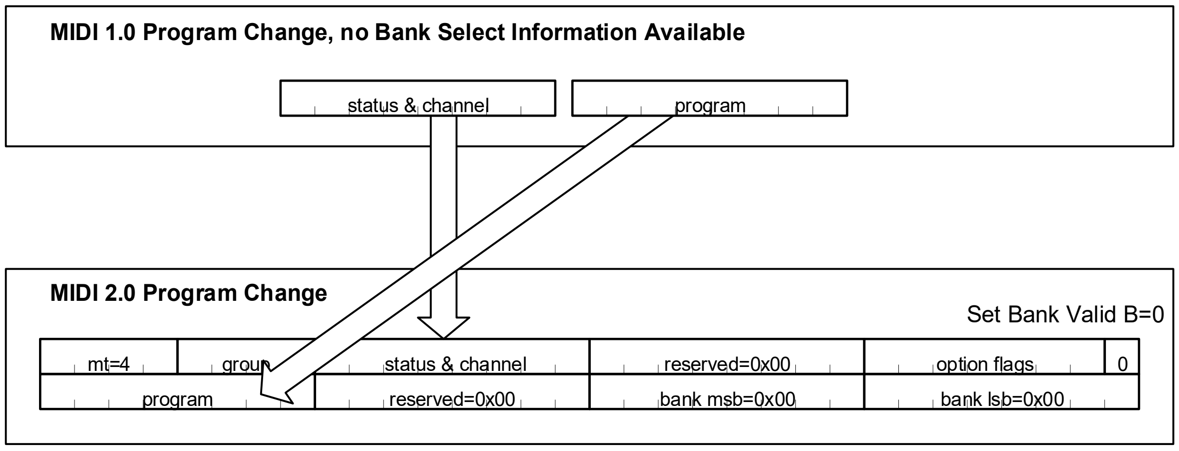


Figure 106 Translate MIDI 1.0 Program Change to MIDI 2.0 (No Bank)

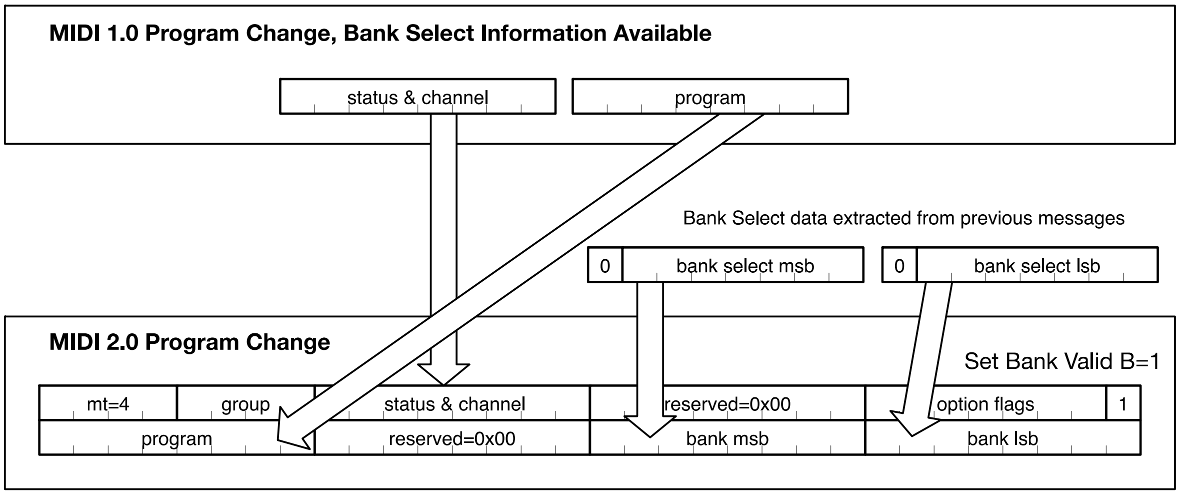


Figure 107 Translate MIDI 1.0 Bank and Program Change to MIDI 2.0

D.3.5 Channel Pressure

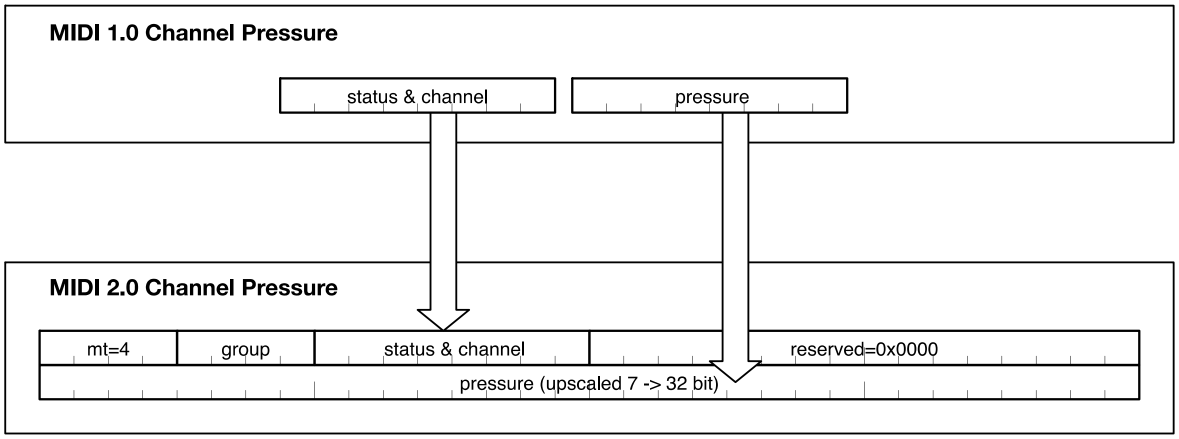

MIDI 1.0 Channel Pressure
status & channel
pressure
MIDI 2.0 Channel Pressure
mt=4
group
status & channel
reserved=0x0000
pressure (upscaled 7 -> 32 bit)

Figure 108 Translate MIDI 1.0 Channel Pressure to MIDI 2.0

D.3.6 Pitch Bend

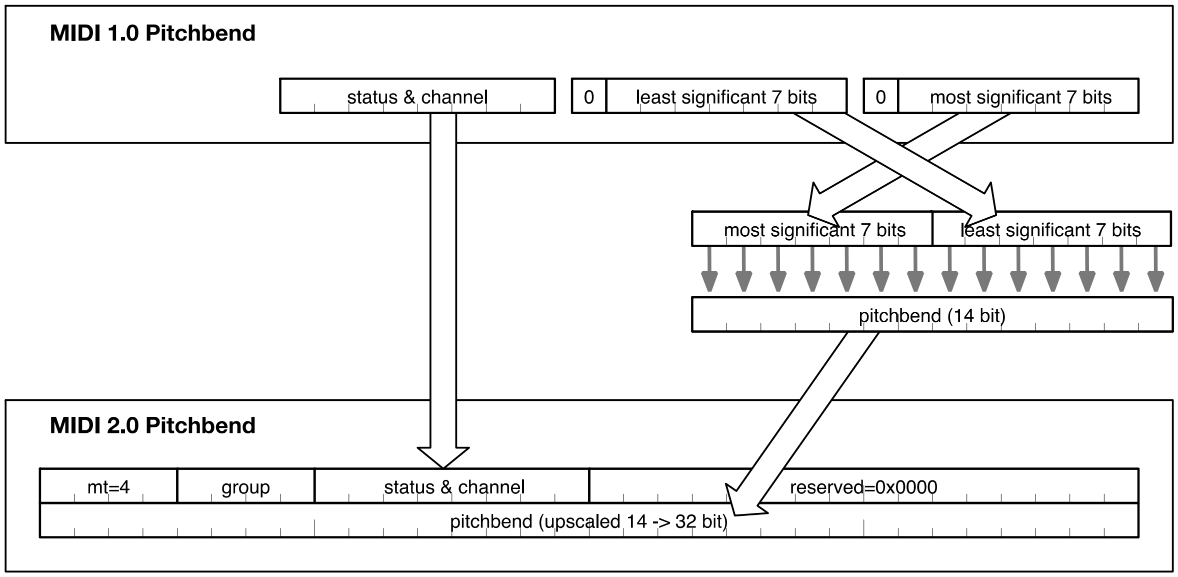


Figure 109 Translate MIDI 1.0 Pitch Bend to MIDI 2.0

Note that Pitch Bend values in the MIDI 1.0 Protocol are presented as Least Significant Byte (LSB) before Most Significant Byte (MSB) (little-endian).

D.3.7 System Messages

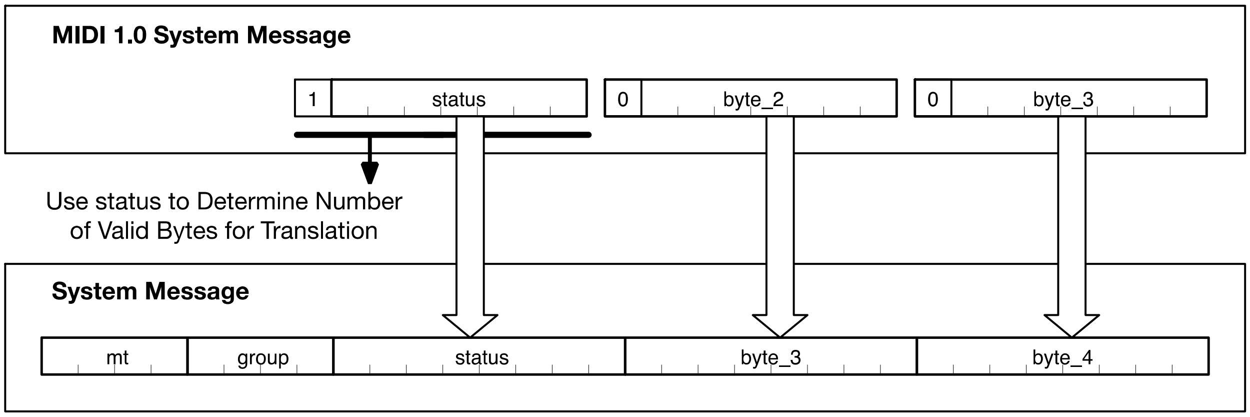
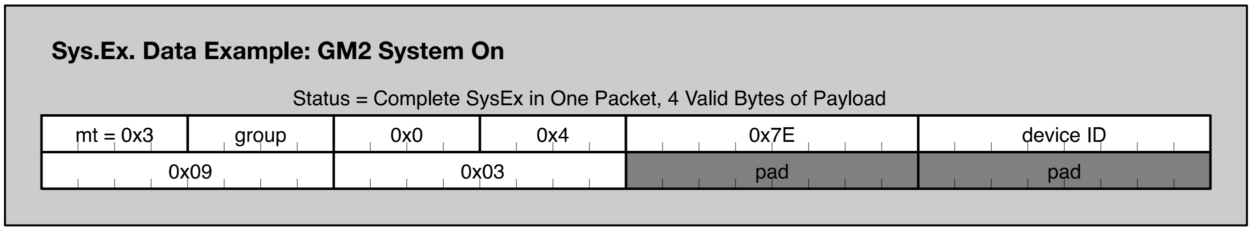


Figure 110 Translate MIDI 1.0 System Message to MIDI 2.0

System Exclusive

When translating a System Exclusive Message from the MIDI 1.0 Protocol to the MIDI 2.0 Protocol, the starting Byte of 0xF0 and ending byte of 0xF7 are discarded. Only the data between those bytes is placed into the payload of the MIDI 2.0 Protocol System Exclusive message. See example in ***Figure 111***.


Sys.Ex. Data Example: GM2 System On
Status = Complete SysEx in One Packet, 4 Valid Bytes of Payload
mt = 0x3
group
0x0
0x4
0x7E
device ID
0x09
0x03
pad
pad

0x7E:	Universal Non-Real Time SysEx header
device ID:	ID of target device (7F = all devices)
0x09:	sub-ID#1 = General MIDI message
0x03:	sub-ID#2 = General MIDI 2 On

Note: Status bytes 0xF0 Start and 0xF7 End used in the original MIDI 1.0 data format are not required and are not included in the message.

Figure 111 Translate MIDI 1.0 System Exclusive to MIDI 2.0 (Example)

D.4 Alternate Translation Modes

Devices are allowed to implement Alternate Translation Modes for special cases. Alternate Translation Modes can be marketed as features that bring added value. MIDI 2.0 Protocol Devices are not required to support Alternate Translation Modes.

A device with Alternate Translation Modes can still be compliant with the MIDI 2.0 specification, as long as the device has a configuration for the Default Translation.

Products with Alternate Translation Modes should inform the user that the Alternate Translation Mode is active.

D.4.1 Selecting an Alternate Translation Mode Using a Profile

Some MIDI 2.0 Protocol messages or parameters that do not have a direct equivalent in the MIDI 1.0 Protocol might be part of a MIDI-CI Profile for use in MIDI 2.0 Protocol Devices. The Profile specification might define an indirect equivalent (perhaps via System Exclusive, a compound message, MPE, or some other mechanism) for use in MIDI 1.0 Protocol Devices. Such Profiles might define a special case translation.

For example, a MIDI-CI Profile might define Per-Note Controllers in the MIDI 2.0 Protocol and MPE in the MIDI 1.0 Protocol. Then the Profile might define a translation. Devices that understand the Profile specification may choose to perform the alternate translations defined by that Profile.

D.4.2 Selecting Alternate Translation Modes Without a Profile

There can be useful alternate translations that are not defined by any MMA specification.

Devices may also enter Alternate Translation Modes by means other than “Profile enable”. The device should notify the user that an Alternate Translation Mode is in use.

For example, a MIDI 2.0 Protocol Device could receive a System Exclusive message that enables MPE mode, and this would enable an Alternate Translation Mode translation for MPE note allocation.

Appendix E: System Exclusive (7-Bit) and System Exclusive 8 (8-Bit) Message Examples

E.1 Table of System Exclusive Message UMPs

Table 24 UMPs for System Exclusive (7-Bit) Messages

	Byte Number									
Message	**1**		**2**		**3**	**4**	**5**	**6**	**7**	**8**
UMP Type	**MT**	**GR**	**Status**	**#bytes**	**Data**					
Complete SysEx	0x3	gr	0x0	0x0*	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
Complete SysEx	0x3	gr	0x0	0x1	0ddddddd	Reserved	Reserved	Reserved	Reserved	Reserved
Complete SysEx	0x3	gr	0x0	0x2	0ddddddd	0ddddddd	Reserved	Reserved	Reserved	Reserved
Complete SysEx	0x3	gr	0x0	0x3	0ddddddd	0ddddddd	0ddddddd	Reserved	Reserved	Reserved
Complete SysEx	0x3	gr	0x0	0x4	0ddddddd	0ddddddd	0ddddddd	0ddddddd	Reserved	Reserved
Complete SysEx	0x3	gr	0x0	0x5	0ddddddd	0ddddddd	0ddddddd	0ddddddd	0ddddddd	Reserved
Complete SysEx	0x3	gr	0x0	0x6	0ddddddd	0ddddddd	0ddddddd	0ddddddd	0ddddddd	0ddddddd
SysEx Start	0x3	gr	0x1	0x0*	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
SysEx Start	0x3	gr	0x1	0x1	0ddddddd	Reserved	Reserved	Reserved	Reserved	Reserved
SysEx Start	0x3	gr	0x1	0x2	0ddddddd	0ddddddd	Reserved	Reserved	Reserved	Reserved
SysEx Start	0x3	gr	0x1	0x3	0ddddddd	0ddddddd	0ddddddd	Reserved	Reserved	Reserved
SysEx Start	0x3	gr	0x1	0x4	0ddddddd	0ddddddd	0ddddddd	0ddddddd	Reserved	Reserved
SysEx Start	0x3	gr	0x1	0x5	0ddddddd	0ddddddd	0ddddddd	0ddddddd	0ddddddd	Reserved
SysEx Start	0x3	gr	0x1	0x6	0ddddddd	0ddddddd	0ddddddd	0ddddddd	0ddddddd	0ddddddd
SysEx Continue	0x3	gr	0x2	0x0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
SysEx Continue	0x3	gr	0x2	0x1	0ddddddd	Reserved	Reserved	Reserved	Reserved	Reserved
SysEx Continue	0x3	gr	0x2	0x2	0ddddddd	0ddddddd	Reserved	Reserved	Reserved	Reserved
SysEx Continue	0x3	gr	0x2	0x3	0ddddddd	0ddddddd	0ddddddd	Reserved	Reserved	Reserved
SysEx Continue	0x3	gr	0x2	0x4	0ddddddd	0ddddddd	0ddddddd	0ddddddd	Reserved	Reserved
SysEx Continue	0x3	gr	0x2	0x5	0ddddddd	0ddddddd	0ddddddd	0ddddddd	0ddddddd	Reserved
SysEx Continue	0x3	gr	0x2	0x6	0ddddddd	0ddddddd	0ddddddd	0ddddddd	0ddddddd	0ddddddd
SysEx End	0x3	gr	0x3	0x0	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
SysEx End	0x3	gr	0x3	0x1	0ddddddd	Reserved	Reserved	Reserved	Reserved	Reserved
SysEx End	0x3	gr	0x3	0x2	0ddddddd	0ddddddd	Reserved	Reserved	Reserved	Reserved
SysEx End	0x3	gr	0x3	0x3	0ddddddd	0ddddddd	0ddddddd	Reserved	Reserved	Reserved
SysEx End	0x3	gr	0x3	0x4	0ddddddd	0ddddddd	0ddddddd	0ddddddd	Reserved	Reserved
SysEx End	0x3	gr	0x3	0x5	0ddddddd	0ddddddd	0ddddddd	0ddddddd	0ddddddd	Reserved
SysEx End	0x3	gr	0x3	0x6	0ddddddd	0ddddddd	0ddddddd	0ddddddd	0ddddddd	0ddddddd

** Some values for #bytes are not valid as long as messages are required to contain ID Number (Manufacturer ID), which is true for all System Exclusive messages at the time of the drafting of this specification. These values are only included in the table in case future MMA/AMEI specifications define the use of short messages without ID Number.*

E.2 Complete System Exclusive Message Examples

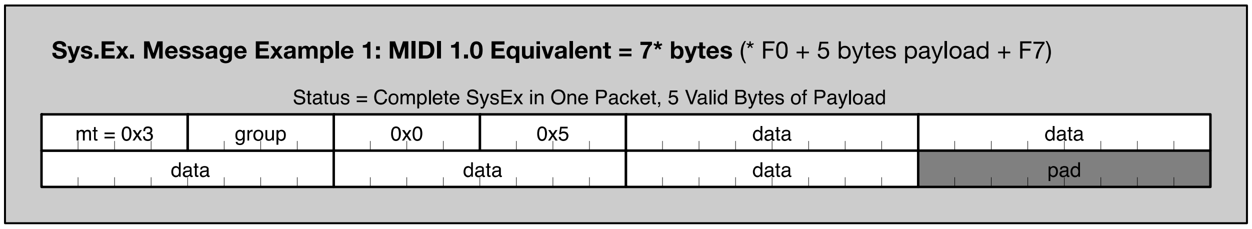

Sys.Ex. Message Example 1: MIDI 1.0 Equivalent = 7* bytes (* F0 + 5 bytes payload + F7)
Status = Complete SysEx in One Packet, 5 Valid Bytes of Payload
mt = 0x3
group
0x0
0x5
data
data
data
data
data
pad

Figure 112 MIDI 2.0 System Exclusive Message Example 1

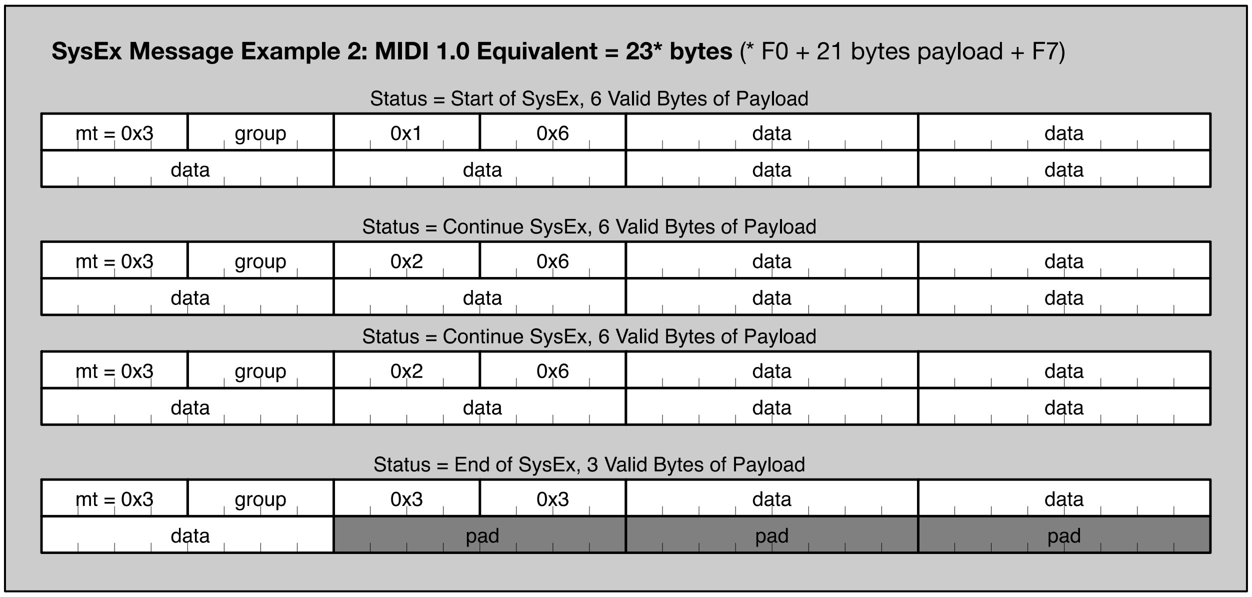
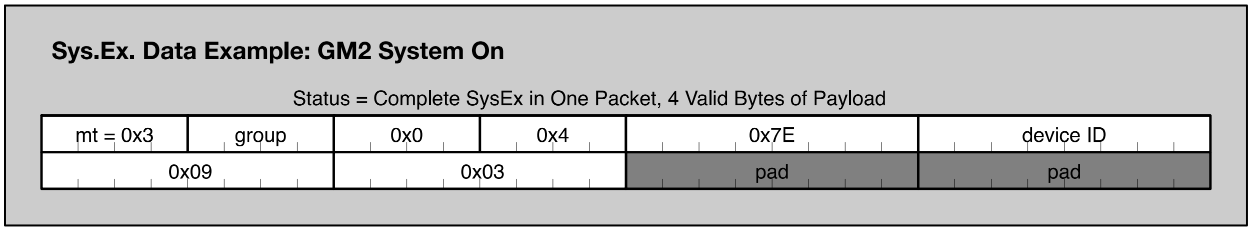

SysEx Message Example 2: MIDI 1.0 Equivalent = 23* bytes (* F0 + 21 bytes payload + F7)
Status = Start of SysEx, 6 Valid Bytes of Payload
mt = 0x3
group
0x1
0x6
data
data
data
data
data
data
Status = Continue SysEx, 6 Valid Bytes of Payload
mt = 0x3
group
0x2
0x6
data
data
data
data
data
data
Status = Continue SysEx, 6 Valid Bytes of Payload
mt = 0x3
group
0x2
0x6
data
data
data
data
data
data
Status = End of SysEx, 3 Valid Bytes of Payload
mt = 0x3
group
0x3
0x3
data
data
data
pad
pad
pad

Figure 113 MIDI 2.0 System Exclusive Message Example 2


Sys.Ex. Data Example: GM2 System On
Status = Complete SysEx in One Packet, 4 Valid Bytes of Payload
mt = 0x3
group
0x0
0x4
0x7E
device ID
0x09
0x03
pad
pad

0x7E:	Universal Non-Real Time SysEx header
device ID:	ID of target device (7F = all devices)
0x09:	sub-ID#1 = General MIDI message
0x03:	sub-ID#2 = General MIDI 2 On

Note: Status bytes 0xF0 Start and 0xF7 End used in the original MIDI 1.0 data format are not required and are not included in the message.

Figure 114 MIDI 2.0 System Exclusive Message Example 3

E.3 Table of System Exclusive 8 (8-Bit) Message UMPs

Table 25 UMPs for System Exclusive 8 Messages

	Byte Number															
Message	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**

UMP Type	MT	GR	Status	Size	Data													
SysEx8 Complete	5	grp	0x0	0x1*	StreamID	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Complete	5	grp	0x0	0x2*	StreamID	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Complete	5	grp	0x0	0x3	StreamID	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Complete	5	grp	0x0	0x4	StreamID	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Complete	5	grp	0x0	0x5	StreamID	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Complete	5	grp	0x0	0x6	StreamID	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Complete	5	grp	0x0	0x7	StreamID	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Complete	5	grp	0x0	0x8	StreamID	data	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Complete	5	grp	0x0	0x9	StreamID	data	data	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Complete	5	grp	0x0	0xA	StreamID	data	data	data	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd
SysEx8 Complete	5	grp	0x0	0xB	StreamID	data	data	data	data	data	data	data	data	data	data	rsvd	rsvd	rsvd
SysEx8 Complete	5	grp	0x0	0xC	StreamID	data	data	data	data	data	data	data	data	data	data	data	rsvd	rsvd
SysEx8 Complete	5	grp	0x0	0xD	StreamID	data	data	data	data	data	data	data	data	data	data	data	data	rsvd
SysEx8 Complete	5	grp	0x0	0xE	StreamID	data	data	data	data	data	data	data	data	data	data	data	data	data
SysEx8 Start	5	grp	0x1	0x1*	StreamID	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Start	5	grp	0x1	0x2*	StreamID	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Start	5	grp	0x1	0x3	StreamID	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Start	5	grp	0x1	0x4	StreamID	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Start	5	grp	0x1	0x5	StreamID	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Start	5	grp	0x1	0x6	StreamID	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Start	5	grp	0x1	0x7	StreamID	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Start	5	grp	0x1	0x8	StreamID	data	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Start	5	grp	0x1	0x9	StreamID	data	data	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Start	5	grp	0x1	0xA	StreamID	data	data	data	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd
SysEx8 Start	5	grp	0x1	0xB	StreamID	data	data	data	data	data	data	data	data	data	data	rsvd	rsvd	rsvd
SysEx8 Start	5	grp	0x1	0xC	StreamID	data	data	data	data	data	data	data	data	data	data	data	rsvd	rsvd
SysEx8 Start	5	grp	0x1	0xD	StreamID	data	data	data	data	data	data	data	data	data	data	data	data	rsvd
SysEx8 Start	5	grp	0x1	0xE	StreamID	data	data	data	data	data	data	data	data	data	data	data	data	data
SysEx8 Continue	5	grp	0x2	0x1	StreamID	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Continue	5	grp	0x2	0x2	StreamID	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Continue	5	grp	0x2	0x3	StreamID	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Continue	5	grp	0x2	0x4	StreamID	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Continue	5	grp	0x2	0x5	StreamID	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Continue	5	grp	0x2	0x6	StreamID	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Continue	5	grp	0x2	0x7	StreamID	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Continue	5	grp	0x2	0x8	StreamID	data	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Continue	5	grp	0x2	0x9	StreamID	data	data	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 Continue	5	grp	0x2	0xA	StreamID	data	data	data	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd
SysEx8 Continue	5	grp	0x2	0xB	StreamID	data	data	data	data	data	data	data	data	data	data	rsvd	rsvd	rsvd
SysEx8 Continue	5	grp	0x2	0xC	StreamID	data	data	data	data	data	data	data	data	data	data	data	rsvd	rsvd
SysEx8 Continue	5	grp	0x2	0xD	StreamID	data	data	data	data	data	data	data	data	data	data	data	data	rsvd
SysEx8 Continue	5	grp	0x2	0xE	StreamID	data	data	data	data	data	data	data	data	data	data	data	data	data
SysEx8 End	5	grp	0x3	0x1	StreamID	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 End	5	grp	0x3	0x2	StreamID	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 End	5	grp	0x3	0x3	StreamID	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 End	5	grp	0x3	0x4	StreamID	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 End	5	grp	0x3	0x5	StreamID	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 End	5	grp	0x3	0x6	StreamID	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 End	5	grp	0x3	0x7	StreamID	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 End	5	grp	0x3	0x8	StreamID	data	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 End	5	grp	0x3	0x9	StreamID	data	data	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd	rsvd
SysEx8 End	5	grp	0x3	0xA	StreamID	data	data	data	data	data	data	data	data	data	rsvd	rsvd	rsvd	rsvd
SysEx8 End	5	grp	0x3	0xB	StreamID	data	data	data	data	data	data	data	data	data	data	rsvd	rsvd	rsvd
SysEx8 End	5	grp	0x3	0xC	StreamID	data	data	data	data	data	data	data	data	data	data	data	rsvd	rsvd
SysEx8 End	5	grp	0x3	0xD	StreamID	data	data	data	data	data	data	data	data	data	data	data	data	rsvd
SysEx8 End	5	grp	0x3	0xE	StreamID	data	data	data	data	data	data	data	data	data	data	data	data	data
SysEx8 End Incomplete	5	grp	0x3	0xF**	StreamID	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd	rsvd

** Some values for #bytes are not valid as long as messages are required to contain ID Number (manufacturer ID), which is true for all System Exclusive 8 messages at the time of the drafting of this specification. They are only included in the table in case future MMA/AMEI specifications define the use of short messages without ID Number.*

*** 0xF is not a valid size. This indicates that a System Exclusive 8 message is terminating unexpectedly with no data.*

Appendix F: All Defined UMP Formats

F.1 4-Byte UMP Formats

F.1.1 Message Type 0x0: Utility

Table 26 4-Byte UMP Formats for Message Type 0x0: Utility

Message	Byte 1		Byte 2	Byte 3		Byte 4
	MT	GR	Status	Data		
UTILITY						
NOOP	0x0	reserved	0000	0000 00000000 00000000		
JR Clock	0x0	reserved	0001	reserved	tttttttt tttttttt	
JR Timestamp	0x0	reserved	0010	reserved	tttttttt tttttttt	
Delta Clockstamp Ticks Per Quarter Note	0x0	reserved	0011	reserved	tttttttt tttttttt	
Delta Clockstamp	0x0	reserved	0100	tttt tttttttt tttttttt		

F.1.2 Message Type 0x1: System Common & System Real Time

Table 27 4-Byte UMP Formats for Message Type 0x1: System Common & System Real Time

Message	Byte 1		Byte 2	Byte 3	Byte 4
	MT	GR	Status	Data	
SYSTEM COMMON					
MIDI Time Code	0x1	gggg	11110001	0nnndddd	reserved
Song Position Pointer	0x1	gggg	11110010	0lllllll	0mmmmmmm
Song Select	0x1	gggg	11110011	0sssssss	reserved
Tune Request	0x1	gggg	11110110	reserved	reserved
SYSTEM REAL TIME					
Timing Clock	0x1	gggg	11111000	reserved	reserved
Start	0x1	gggg	11111010	reserved	reserved
Continue	0x1	gggg	11111011	reserved	reserved
Stop	0x1	gggg	11111100	reserved	reserved
Active Sensing	0x1	gggg	11111110	reserved	reserved
Reset	0x1	gggg	11111111	reserved	reserved

F.1.3 Message Type 0x2: MIDI 1.0 Channel Voice Messages

Table 28 4-Byte UMP Formats for Message Type 0x2: MIDI 1.0 Channel Voice Messages

Message	Byte 1		Byte 2	Byte 3	Byte 4
	MT	GR	Status	Index/Data	
MIDI 1.0 CHANNEL VOICE					
Note Off	0x2	gggg	1000nnnn	rkkkkkkk	rvvvvvvv
Note On	0x2	gggg	1001nnnn	rkkkkkkk	rvvvvvvv
Poly Pressure	0x2	gggg	1010nnnn	rkkkkkkk	rddddddd
Control Change	0x2	gggg	1011nnnn	rccccccc	rddddddd
Program Change	0x2	gggg	1100nnnn	rppppppp	reserved
Channel Pressure	0x2	gggg	1101nnnn	rddddddd	reserved
Pitch Bend	0x2	gggg	1110nnnn	rddddddd	rDDDDDDD

F.2 8-Byte UMP Formats

F.2.1 Message Type 0x3: 8-Byte Data Messages

Table 29 8-Byte UMP Formats for Message Type 0x3: 8-Byte Data Messages

Message	Byte 1		Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
	MT	GR	Status	Data or Pad/Reserved					
DATA									
Sys.Ex. in 1 UMP	0x3	gggg	0000bbbb	0data/pad	0data/pad	0data/pad	0data/pad	0data/pad	0data/pad
SvsEx Start	0x3	gggg	0001bbbb	0data/pad	0data/pad	0data/pad	0data/pad	0data/pad	0data/pad
SysEx Continue	0x3	gggg	0010bbbb	0data/pad	0data/pad	0data/pad	0data/pad	0data/pad	0data/pad
SvsEx End	0x3	gggg	0011bbbb	0data/pad	0data/pad	0data/pad	0data/pad	0data/pad	0data/pad

F.2.2 Message Type 0x4: MIDI 2.0 Channel Voice Messages

Template Instructions: COLOR KEY:	Does not translate to the MIDI 1.0 Protocol	Reserved for future use by MMA/AMEI. Pad with zeros.

Table 30 8-Byte UMP Formats for Message Type 0x4: MIDI 2.0 Channel Voice Messages

Message	Byte 1		Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
	MT	GR	Status	Index		Data			
MIDI 2.0 CHANNEL VOICE									

Note Off	0x4	gggg	1000nnnn	rkkkkkkk	AttributeType	VVVVVVVV	vvvvvvvv	AAAAAAAA	aaaaaaaa
Note On	0x4	gggg	1001nnnn	rkkkkkkk	AttributeType	VVVVVVVV	vvvvvvvv	AAAAAAAA	aaaaaaaa
Poly Pressure	0x4	gggg	1010nnnn	rkkkkkkk	reserved	DDDDDDDD	dddddddd	dddddddd	dddddddd
Registered Per-Note Ctrl.	0x4	gggg	0000nnnn	rkkkkkkk	cccccccc	DDDDDDDD	dddddddd	dddddddd	dddddddd
Assignable Per-Note Ctrl.	0x4	gggg	0001nnnn	rkkkkkkk	cccccccc	DDDDDDDD	dddddddd	dddddddd	dddddddd
Per-Note	0x4	gggg	1111nnn	rkkkkkkk	option flags	reserved	reserved	reserved	reserved
Control	0x4	gggg	1011nnnn	rccccccc	reserved	DDDDDDDD	dddddddd	dddddddd	dddddddd
Registered	0x4	gggg	0010nnnn	rbbbbbbb	rppppppp	DDDDDDDD	dddddddd	dddddddd	dddddddd
Assignable	0x4	gggg	0011nnnn	rbbbbbbb	rppppppp	DDDDDDDD	dddddddd	dddddddd	dddddddd
Relative	0x4	gggg	0100nnnn	rbbbbbbb	rppppppp	DDDDDDDD	dddddddd	dddddddd	dddddddd
Relative	0x4	gggg	0101nnnn	rbbbbbbb	rppppppp	DDDDDDDD	dddddddd	dddddddd	dddddddd
Program	0x4	gggg	1100nnnn	reserved	option flags	rppppppp	reserved	rBBBBBBB	rbbbbbbb
Channel	0x4	gggg	1101nnnn	reserved	reserved	DDDDDDDD	dddddddd	dddddddd	dddddddd
Pitch Bend	0x4	gggg	1110nnnn	reserved	reserved	DDDDDDDD	dddddddd	dddddddd	dddddddd
Per-Note	0x4	gggg	0110nnnn	rkkkkkkk	reserved	DDDDDDDD	dddddddd	dddddddd	dddddddd

F.3 16-Byte UMP Formats

F.3.1 Message Type 0x5: 16-Byte Data Messages (System Exclusive 8 and Mixed Data Set)

Table 31 16-Byte UMP Formats for Message Type 0x5: System Exclusive 8 and Mixed Data Set

Message	Byte 1		Byte 2		Byte 3	Byte 4-16
	MT	GR	Status	Low 4 bits		
SysEx8 in 1 UMP	0x5	gggg	0000	#bytes	stream id	data/pad
SysEx8 Start	0x5	gggg	0001	#bytes	stream id	data/pad
SysEx8 Continue	0x5	gggg	0010	#bytes	stream id	data/pad
SysEx8 End	0x5	gggg	0011	#bytes	stream id	data/pad
Mixed Data Set Header	0x5	gggg	1000	mds id	Header Fields	
Mixed Data Set Payload	0x5	gggg	1001	mds id	Payload Data	

F.3.1 Message Type 0xD: Flex Data Messages

Table 32 128 bit UMP Formats for Message Type 0xD: Flex Data Messages

	32bit word 1							32bit word 2				32bit word 3				32bit word 4			
	31-28	**27-24**	**23-22**	**21-20**	**19-16**	**15-8**	**7-0**	**31-24**	**23-16**	**15-8**	**7-0**	**31-24**	**23-16**	**15-8**	**7-0**	**31-24**	**23-16**	**15-8**	**7-0**
Message	**MT**	**Grp**	**Form**	**Addr**	**Ch.**	**SB**	**St**	**Data**											
Set Tempo Message	0xD	gggg	0	1	0000	0x00	0x00	Number of 10 Nanosecond units Per Quarter Note				reserved				reserved			
Set Time Signature	0xD	gggg	0	1	0000	0x00	0x01	Numerator, Denominator, Num of 1/32nd Notes				reserved				reserved			
Set Metronome	0xD	gggg	0	1	0000	0x00	0x02	Num Clocks/Primary click, Bar accents 1,2 and 3				Num Sub-division clicks 1 and 2		reserved		reserved			
Set Key Signature	0XD	gggg	0	aa	cccc	0x00	0x05	sharp/flats, tonic note		reserved		reserved				reserved			
Set Chord Name	0xD	gggg	0	aa	cccc	0x00	0x06	C.Shrp/Flts, Tonic, Type		alter 1/2 type & deg.		alter 3/4 type & deg		reserved		B.Shrp/Flts, Tonic, Type		alter 1/2 type & deg.	
Text Message Common Format	0XD	gggg	0-3	aa	cccc	0x01 - 0x02	ssss ssss	Data											

F.3.2 Message Type 0xF: UMP Stream Messages

Table 33 128 bit UMP Formats for Message Type 0xF: UMP Stream Messages

	32bit word 1					32bit word 2				32bit word 3				32bit word 4			
	31-28	**27-26**	**25-16**	**15-8**	**7-0**	**31-24**	**23-16**	**15-8**	**7-0**	**31-24**	**23-16**	**15-8**	**7-0**	**31-24**	**23-16**	**15-8**	**7-0**
Message	**MT**	**Form**	**Status**	**Data**													
Endpoint Discovery	0xF	0	0x00	ver. Maj	ver. Min	reserved			filter	reserved				reserved			
Endpoint Info Notification	0xF	0	0x01	ver. Maj	ver. Min	static, #fb blocks	reserve d	M2, M1 Sup.	JR Sup.	reserved				reserved			
Device Identity Notification	0xF	0	0x02	reserved			SysEx id bytes			Family id bytes		Model id bytes		Version bytes			
Endpoint Name Notification	0xF	0-3	0x03	name bytes													
Product Instance Id Notification	0xF	0-3	0x04	Product Instance Id Bytes													
Stream Configuration Request	0xF	0	0x05	prot,	jr	reserved											
Stream Configuration Notification	0xF	0	0x06	prot,	jr	reserved											
Function Block Discovery	0xF	0	0x10	fb #	filter	reserved				reserved				reserved			

Function Block Info Notification	0xF	0	0x11	act, fb #	UI, M1.0, dir	first group	# groups	MIDI-CI	# SyEx 8	reserved	reserved
Function Block Name Notification	0xF	0-3	0x12	name bytes							
Start of Clip	0xF	0	0x20	reserved							
End of Clip	0xF	0	0x21	reserved							

Appendix G: All Defined Messages

Table 34 All Defined Message Formats (in 5 parts)

Message Type	MIDI Message	Byte 1		Byte 2		Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Bytes 9-16 (64 bits)
		4 bit MT	4 bit Group	Status	Channel / Other							
UTILITY	NOOP	0x0	res.	0x0	20 bit 0x00000							
UTILITY	JR Clock	0x0	res.	0x1	reserved	16 bit 0xtttt						
UTILITY	JR Timestamp	0x0	res.	0x2	reserved	16 bit 0xtttt						
UTILTIY	Delta Clockstamp Ticks Per Quarter Note	0x0	res.	0x3	reserved	16 bit 0xtttt						
UTILTIY	Delta Clockstamp	0x0	res.	0x4	20 bit 0xttttt							
SYSTEM COMMON	MIDI Time Code	0x1	0xg	0xF1		7 bit time code 0xnd	reserved					
SYSTEM COMMON	Song Position Pointer	0x1	0xg	0xF2		7 bit position LSB 0xll	7 bit position MSB 0xmm					
SYSTEM COMMON	Song Select	0x1	0xg	0xF3		7 bit song# 0xss	reserved					
SYSTEM COMMON	Tune Request	0x1	0xg	0xF6		reserved	reserved					
SYSTEM REAL TIME	Timing Clock	0x1	0xg	0xF8		reserved	reserved					
SYSTEM REAL TIME	Start	0x1	0xg	0xFA		reserved	reserved					
SYSTEM REAL TIME	Continue	0x1	0xg	0xFB		reserved	reserved					
SYSTEM REAL TIME	Stop	0x1	0xg	0xFC		reserved	reserved					
SYSTEM REAL TIME	Active Sensing	0x1	0xg	0xFE		reserved	reserved					
SYSTEM REAL TIME	Reset	0x1	0xg	0xFF		reserved	reserved					
MIDI 1.0 CHANNEL VOICE	Note Off	0x2	0xg	0x8	0xn	7 bit note# 0xkk	7 bit velocity 0xvv					
MIDI 1.0 CHANNEL VOICE	Note On	0x2	0xg	0x9	0xn	7 bit note# 0xkk	7 bit velocity 0xvv					
MIDI 1.0 CHANNEL VOICE	Poly Pressure	0x2	0xg	0xA	0xn	7 bit note# 0xkk	7 bit pressure 0xpp					
MIDI 1.0 CHANNEL VOICE	Control Change	0x2	0xg	0xB	0xn	7 bit controller# 0xcc	7 bit value 0xvv					
MIDI 1.0 CHANNEL VOICE	Program Change	0x2	0xg	0xC	0xn	7 bit program# 0xpp	reserved					
MIDI 1.0 CHANNEL VOICE	Channel Pressure	0x2	0xg	0xD	0xn	7 bit chan pressure	reserved					
MIDI 1.0 CHANNEL VOICE	Pitch Bend	0x2	0xg	0xE	0xn	7 bit pitch bend LSB	7 bit pitch bend MSB					
DATA 64 BIT	SysEx in 1 Packet	0x3	0xg	0x0	0xb	7 bit data/pad	7 bit data/pad	7 bit data/pad	7 bit data/pad	7 bit data/pad	7 bit data/pad	
DATA 64 BIT	SysEx Start	0x3	0xg	0x1	0xb	7 bit data/pad	7 bit data/pad	7 bit data/pad	7 bit data/pad	7 bit data/pad	7 bit data/pad	

Message Type	**MIDI Message**	**Byte 1**		**Byte 2**		**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6**	**Byte 7**	**Byte 8**	**Bytes 9-16 (64 bits)**
		4 bit MT	**4 bit Group**	**Status**	**Channel / Other**							
DATA 64 BIT	SysEx Continue	0x3	0xg	0x2	0xb	7 bit data/pad	7 bit data/pad	7 bit data/pad	7 bit data/pad	7 bit data/pad	7 bit data/pad	
DATA 64 BIT	SysEx End	0x3	0xg	0x3	0xb	7 bit data/pad	7 bit data/pad	7 bit data/pad	7 bit data/pad	7 bit data/pad	7 bit data/pad	

Message Type	MIDI Message	Byte 1		Byte 2		Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Bytes 9-16 (64 bits)
		4 bit MT	4 bit Group	Status	Channel / Other							
MIDI 2.0 CHANNEL VOICE	Regist. Per-Note Ctrl.	0x4	0xg	0x0	0xn	7 bit note# 0xkk	8 bit controller# 0xcc	32 bit data 0xdddd dddd				
MIDI 2.0 CHANNEL VOICE	Assign. Per-Note Ctrl.	0x4	0xg	0x1	0xn	7 bit note# 0xkk	8 bit controller# 0xcc	32 bit data 0xdddd dddd				
MIDI 2.0 CHANNEL VOICE	Registered Ctrl. (RPN)	0x4	0xg	0x2	0xn	7 bit bank# 0xbb	7 bit index# 0xpp	32 bit data 0xdddd dddd				
MIDI 2.0 CHANNEL VOICE	Assignable Ctrl. (NRPN)	0x4	0xg	0x3	0xn	7 bit bank# 0xbb	7 bit index# 0xpp	32 bit data 0xdddd dddd				
MIDI 2.0 CHANNEL VOICE	Relative Regist. Ctrl.	0x4	0xg	0x4	0xn	7 bit bank# 0xbb	7 bit index# 0xpp	32 bit data 0xdddd dddd				
MIDI 2.0 CHANNEL VOICE	Relative Assign. Ctrl.	0x4	0xg	0x5	0xn	7 bit bank# 0xbb	7 bit index# 0xpp	32 bit data 0xdddd dddd				
MIDI 2.0 CHANNEL VOICE	Per-Note Pitch Bend	0x4	0xg	0x6	0xn	7 bit note# 0xkk	reserved	32 bit data 0xdddd dddd				
MIDI 2.0 CHANNEL VOICE	Note Off	0x4	0xg	0x8	0xn	7 bit note# 0xkk	attribute type	16 bit velocity 0xvvvvv		16 bit attribute value 0xaaaa		
MIDI 2.0 CHANNEL VOICE	Note On	0x4	0xg	0x9	0xn	7 bit note# 0xkk	attribute type	16 bit velocity 0xvvvvv		16 bit attribute value 0xaaaa		
MIDI 2.0 CHANNEL VOICE	Poly Pressure	0x4	0xg	0xA	0xn	7 bit note# 0xkk	reserved	32 bit data 0xdddd dddd				
MIDI 2.0 CHANNEL VOICE	Control Change	0x4	0xg	0xB	0xn	7 bit controller# 0xcc	reserved	32 bit data 0xdddd dddd				
MIDI 2.0 CHANNEL VOICE	Program Change	0x4	0xg	0xC	0xn	reserved	option flags	7 bit program 0xpp	reserved	7 bit bank MSB 0xBB	7 bit bank LSB 0xbb	
MIDI 2.0 CHANNEL VOICE	Channel Pressure	0x4	0xg	0xD	0xn	reserved	reserved	32 bit data 0xdddd dddd				
MIDI 2.0 CHANNEL VOICE	Pitch Bend	0x4	0xg	0xE	0xn	reserved	reserved	32 bit data 0xdddd dddd				
MIDI 2.0 CHANNEL VOICE	Per-Note Management	0x4	0xg	0xF	0xn	7 bit note# 0xkk	option flags	reserved	reserved	reserved	reserved	
DATA 128 BIT	SysEx8 in 1 Packet	0x5	0xg	0x0	#bytes	stream id	104 bit data/pad					
DATA 128 BIT	SysEx8 Start	0x5	0xg	0x1	#bytes	stream id	104 bit data/pad					
DATA 128 BIT	SysEx8 Continue	0x5	0xg	0x2	#bytes	stream id	104 bit data/pad					
DATA 128 BIT	SysEx8 End	0x5	0xg	0x3	#bytes	stream id	104 bit data/pad					
DATA 128 BIT	Mixed Data Set Header	0x5	0xg	0x8	mds id	112 bit header fields						
DATA 128 BIT	Mixed Data Set Payload	0x5	0xg	0x9	mds id	112 bit payload data/pad						

Message Type	MIDI Message	Byte 1		Byte 2		Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Bytes 9-16 (64 bits)
		4 bit MT	4 bit Group	Status	Channel / Other							
UMP Stream	Endpoint Discovery	0xF	0	status 0x0		UMP ver. Major	UMP ver. Minor	reserved			filter bitmap	res
UMP Stream	Endpoint Info Notification	0xF	0	status 0x1		UMP ver. Major	UMP ver. Minor	# of function blocks, M1, M2 JR support				res
UMP Stream	Device Identity Notification	0xF	0	status 0x2		reserved		SysEx, Family, Model and Version id's				
UMP Stream	Endpoint Name Notification	0xF	0-3	status 0x3		Name bytes						
UMP Stream	Product Instance Id Notification	0xF	0-3	status 0x4		Product Instance Id bytes						
UMP Stream	Stream Configuration Request	0xF	0	status 0x5		protocol	jr	reserved				
UMP Stream	Stream Configuration Notification	0xF	0	status 0x6		protocol	jr	reserved				
UMP Stream	Function Block Discovery	0xF	1	status 0x10		function block #	filter bitmap	reserved				
UMP Stream	Function Block Info Notification	0xF	0	status 0x11		function block #	Function Block data					res
UMP Stream	Function Block Name Notification	0xF	0-3	status 0x12		function block #	Name bytes					
UMP Stream	Start of Clip	0xF	0	status 0x20		reserved						
UMP Stream	End of Clip	0xF	0	status 0x21		reserved						

Message Type	MIDI Message	Byte 1		Byte 2			Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Bytes 9-16 (64 bits)
		4 bit MT	4 bit Gr	2 bit Form	2 bit Addr	Channel / Other	Status Bank	Status					
Flex Data	Set Tempo	0xD	0xg	0	1	0	0x00	0x00	Number of 10 Nanosecond units Per Quarter Note				res
Flex Data	Set Time Signature	0xD	0xg	0	1	0	0x00	0x01	Numerator, Denominator, Num of 1/32nd Notes			reserved	
Flex Data	Set Metronome	0xD	0xg	0	1	0	0x00	0x02	Num Clocks/Primary click, Bar accents 1,2 and 3				data
Flex Data	Set Key Signature	0xD	0xg	0	0xa	0xn	0x00	0x05	sharp/flats, tonic note		reserved		
Flex Data	Set Chord Name	0xD	0xg	0	0xa	0xn	0x00	0x06	data				
Flex Data	Text Message Common Format	0xD	0xg	0-3	0xa	0xn	0x01-0x02	0xvv	data				

Color Key	
	Does not translate to MIDI 1.0 Protocol
	Does not translate to MIDI 1.0 Protocol, but may be used by a UMP MIDI 1.0 Device
	Reserved for future use by the Association of Musical Electronics Industry and the MIDI Manufacturers Association. Pad with zeros.

Appendix H: MIDI 2.0 Addressing

Table 35 MIDI 2.0 Addressing

Message	**Destination**			
	UMP Stream	**Function Block**	**UMP Group**	**Channel**
UMP Stream Messages (MT 0xF)	• UMP Endpoint Messages • Function Block Messages • Protocol Selection			
Utility Messages (MT 0x0)	• NOOP • JR Timestamps • Delta Clockstamps			
MIDI-CI - System Exclusive Messages (MT 0x3)		If Source / Destination field = 0x7F: • Discovery • Profile Inquiry • Profile Inquiry Reply • Property Exchange • Process Inquiry Capabilities • MIDI-CI ACK/NAK	If Source / Destination field = 0x7E: • Protocol Negotiation (old) • Profile Inquiry Reply • MIDI-CI ACK/NAK	If Source / Destination field = 0x00-0x0F: • Profile Inquiry Reply - Single/Multi Channel • MIDI Message Report • MIDI-CI ACK/NAK
SysEx8/Mixed Data Set (MT 0x5)			See individual specifications	
Universal System Exclusive Messages other than MIDI-CI (MT 0x3)			See individual specifications	
UMP System Messages (MT 0x1)			All Messages	
Flex Data Messages (MT 0xD)			Based on Address field	Based on Address and Channel fields
UMP MIDI 1.0 CVM (MT 0x2)				All Messages
UMP MIDI 2.0 CVM (MT 0x4)				All Messages

Appendix I: Using USB MIDI Group Terminal Blocks and Function Blocks

The **USB Class Specification for MIDI Devices, Version 2.0, [USBIF02]** describes the use of Group Terminal Blocks to declare the entities that operate on a set of one or more Groups.

USB Devices that use Function Blocks and Group Terminal Blocks may cause an overlap of features or potential conflict of declared topology.

One strategy to avoid this conflict is to declare a single bidirectional Group Terminal Block using all 16 Groups. Multiple Function Blocks can exist in that Group Terminal Block without conflict. In the USB Group Terminal Block Descriptor, select a Protocol descriptor that supports 128 bit messages.

In case that the device only has a static Function Block configuration and supports only one protocol, it can mirror its Function Block configuration in the Group Terminal Block descriptors. This allows compatibility with USB MIDI 2 host drivers but at the same time allows the use of the new UMP 1.1 (stream) messages.

USB Devices that do not use the above strategies with their Group Terminal Blocks should consider the following:

- UMP Stream Messages (MT=0xF) may not be received by the UMP Endpoint.
- Utility Messages (MT=0x0) may only be received on Group 1.
- MIDI-CI v1.1 Protocol Negotiation (deprecated in MIDI-CI v1.2) may be required to change Protocols.
- A Group Terminal Block has specific rules for use of MIDI-CI. See ***[MA02]*** MIDI Capability Inquiry (MIDI-CI) specification (version 1.2 or higher) for details.

Appendix J: Overview of Extensions to MIDI

Note: The lists below are overviews and are not exhaustive.

J.1 Extensions Enabled by the Universal MIDI Packet Format

These extensions apply to both the MIDI 1.0 Protocol and the MIDI 2.0 Protocol:

- 16 Groups. Each Group has a set of System Messages and 16 Channels
- Messages expanded to 32, 64, 96, or 128-bit message UMPs
- Running Status is no longer used
- Adds a NOOP (no operation) message
- Adds optional Jitter Reduction Timestamps
- Adds new System Exclusive 8 Message without the 7-bit limitation of System Exclusive
- Adds new Mixed Data Set Message for carrying large data sets
- The Message Type field allows future extensibility. Many opcodes are available for new messages to be defined in the future by MMA/AMEI. The Message Type field also allows future definition of longer versions of existing messages to include more properties.
- New Message Type: UMP Stream Messages – Messages sent to the whole UMP Endpoint.

J.2 Further Extensions in the MIDI 2.0 Protocol

- Increases Resolution of Velocity in Note On and Note Off to 16 bits
- Adds 8-bit Articulation Type and 16-bit Articulation Data fields to Note On and Note Off
- Increases Resolution of Poly Pressure messages to 32 bits
- New Message: Registered Per-Note Controllers
- New Message: Assignable Per-Note Controllers
- New Message: Per-Note Management Message
- Increases Resolution of Control Change messages to 32 bits
- RPN and NRPN are now unified messages, and as a result are easier to use plus their resolution has been extended to 32 bits
- Relative Control of RPN/NRPN (Increment & Decrement) now easier to use and high resolution
- Renames RPN and NRPN to Registered Controllers and Assignable Controllers
- Program Change and Bank Select are combined into a single, unified message
- Increases Resolution of Channel Pressure messages to 32 bits
- Increases Resolution of Pitch Bend to 32 bits
- Adds Per-Note Pitch Bend Message
- New Message Type: Flex Data Messages – real time messages with limited variability of size.