# Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol

Version 1.0
February 20, 2020

Published By:

Association of Musical Electronics Industry AMEI
and
MIDI Manufacturers Association MMA

M2-104-UM

http://www.midi.org

\

M2-104-UM v1.0 Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol

# Table of Contents

## Figures

## Tables

# 1. Introduction

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.

## 1.1 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 2.1.1.1***.
- 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 ***[MMA01]***.

## 1.2 References

[MMA01] ***The Complete MIDI 1.0 Detailed Specification, Document Version 96.1, Third Edition***, Association of Musical Electronics Industry, http://www.amei.or.jp/, and MIDI Manufacturers Association, https://www.midi.org/.

[MMA02] ***MIDI Capability Inquiry (MIDI-CI)***, Association of Musical Electronics Industry, http://www.amei.or.jp/, and MIDI Manufacturers Association, https://www.midi.org/.

[MMA03] ***Confirmation of Approval for MIDI Standard CA-031, CC #88 High Resolution Velocity Prefix***, Association of Musical Electronics Industry, http://www.amei.or.jp/, and MIDI Manufacturers Association, https://www.midi.org/.

[MMA04] ***Defaults for Sound Controllers***, Recommended Practice RP-021, MIDI Manufacturers Association, https://www.midi.org/.

[MMA05] ***Redefinition of CC91 and CC93***, Recommended Practice RP-023, MIDI Manufacturers Association, https://www.midi.org/.

## 1.3 Terminology

***Note:** Terminology from MIDI 1.0 is the same as defined in the Complete MIDI 1.0 Detailed Specification **[MMA01]** and is not included here.*

**Alternate Translation Mode**: A non-standardized translation of MIDI 1.0 Protocol to MIDI 2.0 Protocol, or MIDI 2.0 Protocol to MIDI 1.0 Protocol, as described in ***Appendix D.4*** of this specification.

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

**Attribute Data, Attribute Type:** In the MIDI 2.0 Protocol's Note On and Note Off messages, optional fields that support additional expression. See ***Section 4.2.1***, ***Section 4.2.2***, and ***Section 4.2.13***.

**Data Message:** MIDI Message defined in ***Section 4.4 System Exclusive (7-Bit) Messages***, ***Section 4.5 System Exclusive 8 (8-Bit) Messages***, or ***Section 4.6 Mixed Data Set Message***.

**Default Translation Mode**: A standardized translation of the MIDI 1.0 Protocol to the MIDI 2.0 Protocol, or from the MIDI 2.0 Protocol to the MIDI 1.0 Protocol, conforming to the rules in ***Appendix D.1*** through ***Appendix D.3*** of this specification.

**JR:** Jitter Reduction.

**MIDI 1.0 Protocol:** Version 1.0 of the MIDI Protocol as originally specified in ***[MMA01]***. The native format for the MIDI 1.0 Protocol is a byte stream, but it has been adapted for many different transports. The UMP format for MIDI 1.0 Protocol messages is defined in Section 4 of this specification.

**MIDI 1.0 Protocol Device:** Hardware or software device that sends and/or receives MIDI Messages in the MIDI 1.0 Protocol, using any transport. See also Non-UMP MIDI 1.0 System.

**MIDI 1.0 Specification:** See ***[MMA01]***.

**MIDI 2.0**: The MIDI environment that encompasses all of MIDI 1.0 Protocol, MIDI-CI, Universal MIDI Packet, MIDI 2.0 Protocol, MIDI 2.0 Messages, and other extensions to MIDI as described in this specification.

**MIDI 2.0 Protocol:** Version 2.0 of the MIDI Protocol, as defined in this specification. The native format for MIDI 2.0 messages is UMP as defined in ***Section 4*** of this specification.

**MIDI 2.0 Protocol Device:** Hardware or software device that sends and receives MIDI Messages in the MIDI 2.0 Protocol.

**MIDI-CI Protocol Negotiation:** A two-way exchange of MIDI-CI messages to determine, through negotiation, which MIDI protocol the two devices will use to communicate. See ***[MMA02]***.

**MIDI Message:** (1) A complete MIDI 1.0 Protocol message, irrespective of the transport employed (if any), as specified in ***The Complete MIDI 1.0 Detailed Specification [MMA01]***, including any updates, addenda, or errata); or
(2) A complete MIDI 1.0 Protocol message or a complete MIDI 2.0 Protocol message in UMP Format, irrespective of the transport employed (if any), as specified in ***Section 3.2***, ***Section 3.3***, and ***Section 4*** of this specification.
In this definition, 'complete' means that all specified fields are incorporated, including any reserved bits, and conform to the message's defined format. Note that for some transports a single MIDI Message might span multiple transport packets, or a single transport packet might contain multiple MIDI Messages.

**MIDI Tuning Standard:** The mechanism for controlling musical tuning (intonation) as specified in the System Exclusive Messages section of the ***MIDI 1.0 Detailed Specification***, Document Version 4.2, which is published in ***The Complete MIDI 1.0 Detailed Specification [MMA01]***.

**Mixed Data Set:** Mixed Data Set messages can carry any data payload, without the 7-bit restriction of the MIDI 1.0 Protocol. See ***Section 4.6***.

**MMA:** MIDI Manufacturers Association. Authority for MIDI specifications worldwide except Japan.

**MMA/AMEI:** MIDI Manufacturers Association and Association for Musical Electronics Industry.

**Non-UMP MIDI 1.0 System**: Any combination of MIDI 1.0 Protocol Devices, transports, applications, or other system components that does not implement the UMP Format.

> ***Note:*** *At the time of writing this specification, there is no plan to use the UMP Format on the MIDI 1.0 5-pin DIN transport. Unless/until that plan changes, 5-pin DIN will only support the MIDI 1.0 byte stream data format.*

**Per-Note Controller**: Any of the following MIDI Messages: Poly Pressure, Per-Note Registered Controllers, Per-Note Assignable Controllers, or Per-Note Pitch Bend.

**Profile:** MMA/AMEI specification with purpose-specific definition of MIDI functionality, as defined in the MIDI-CI specification ***[MMA02]*** or its updates, addenda, or errata.

**Receiver:** MIDI 1.0 Protocol Device or MIDI 2.0 Protocol Device that receives MIDI Messages from a Sender and parses them.

**Sender:** MIDI 1.0 Protocol Device or MIDI 2.0 Protocol Device that creates MIDI Messages and transmits them to a Receiver.

**Translator:** A MIDI 1.0 Protocol Device or MIDI 2.0 Protocol Device which is capable of translating MIDI 1.0 Protocol messages to the MIDI 2.0 Protocol, and/or translating MIDI 2.0 Protocol messages to the MIDI 1.0 Protocol.

**UMP:** Universal MIDI Packet.

**UMP Format:** Data format for fields and messages in the Universal MIDI Packet.

**UMP MIDI 1.0 Device**: any device that sends or receives MIDI 1.0 Protocol messages using the UMP. 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 MMA/AMEI elects to officially support UMP. See ***Section 2*** of this specification for detailed definition.

**Utility Message:** MIDI Message composed of one or more of the UMPs defined in ***Section 4.8***.

## 1.4 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 1 Words Relating to Specification Conformance**

| Word | Reserved For | Relation to Spec Conformance |
|---|---|---|
| **shall** | Statements of requirement | Mandatory.<br>A conformant implementation conforms to all 'shall' statements. |
| **should** | Statements of recommendation | Recommended but not mandatory.<br>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.<br>An implementation that does not conform to some or all 'may' statements is still conformant, providing 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 2 Words Not Relating to Specification Conformance**

| Word | Reserved For | Notes |
|---|---|---|
| **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.<br>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.<br>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.<br>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.<br>Not used for statements of conformance permission (see 'may' above). |

# 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 using the Message formats defined in ***Section 4.1***
- All MIDI 2.0 Protocol Channel Voice Messages using the Message formats defined in ***Section 4.2***
- The System Common, System Real Time, System Exclusive, System Exclusive 8, Mixed Data Set, and Utility messages using the Message formats defined in ***Section 4.3*** through ***Section 4.8***.

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 Data 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.

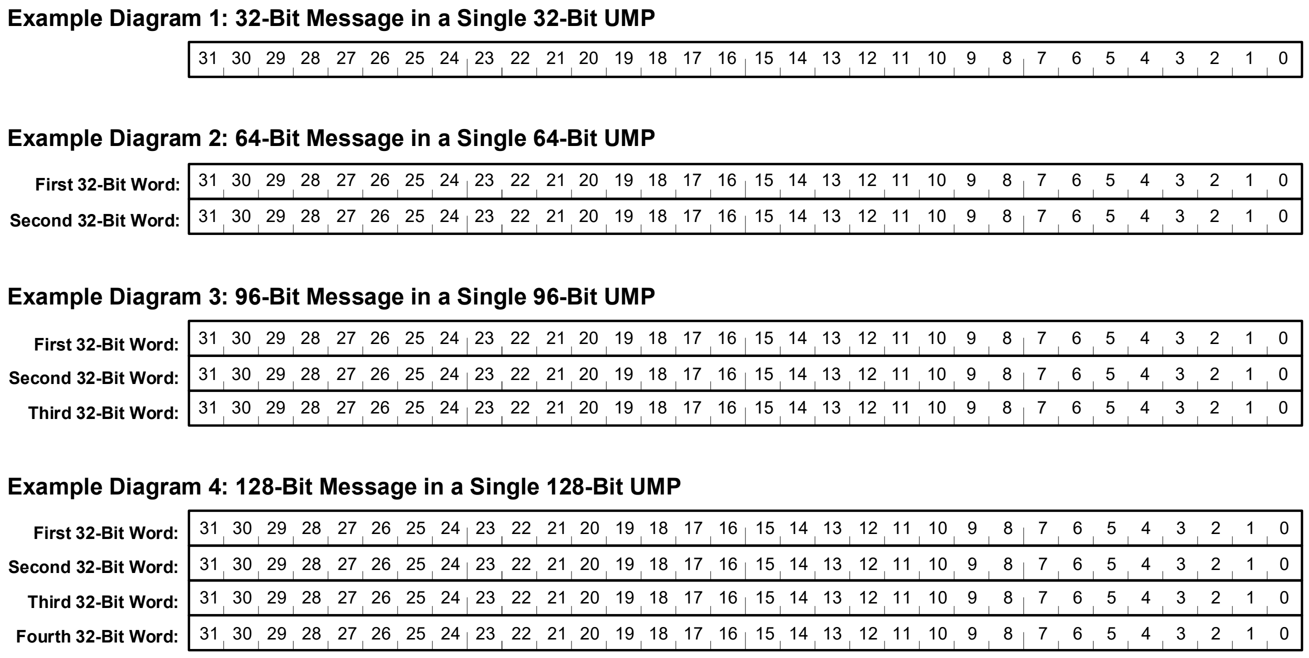


**Figure 1 Example UMP Format Diagrams**

#### 2.1.1.1 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 1.1***, it is expected that 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 Universal Fields

Every UMP shall contain the **Message Type**, **Group**, and **Status** fields.

**message type**

The most significant 4 bits in every UMP shall contain the **Message Type** field, detailed in ***Section 2.1.4***. It 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**

A 4-bit **Group** field is next, addressing every UMP Format MIDI Message (and every UMP comprising any given MIDI Message) to one of 16 Groups.

Per ***Section 3***, each Group shall communicate using only one MIDI Protocol (currently either the MIDI 1.0 Protocol or the MIDI 2.0 Protocol) at a time. MIDI Protocols shall not be mixed for any given Group.

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, Data Messages, and JR Timestamps) shall apply to, and shall affect, all MIDI Channels within that Group. In addition, each Group shall be separate and independent from any other Group in terms of its response to System Messages.

**status**

A **Status** field is next. 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.

Within each Message Type multiple messages are defined, distinguished from one another by the Status field. 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.

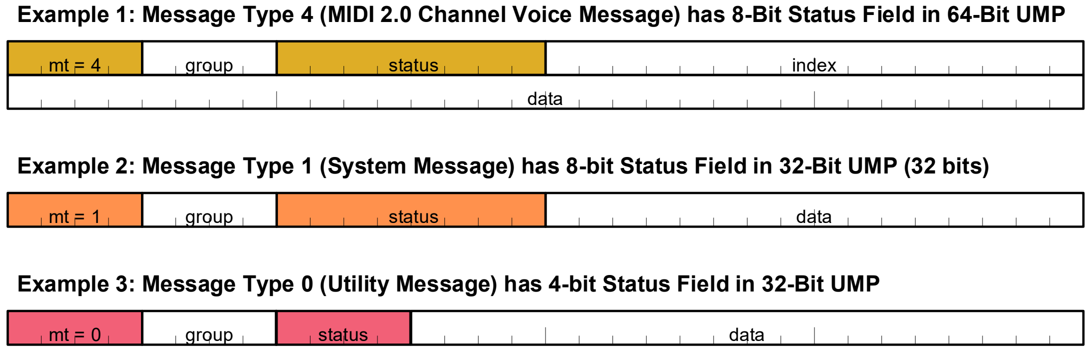

Example 1: Message Type 4 (MIDI 2.0 Channel Voice Message) has 8-Bit Status Field in 64-Bit UMP
mt = 4
group
status
index
data
Example 2: Message Type 1 (System Message) has 8-bit Status Field in 32-Bit UMP (32 bits)
mt = 1
group
status
data
Example 3: Message Type 0 (Utility Message) has 4-bit Status Field in 32-Bit UMP
mt = 0
group
status
data

**Figure 2 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 3 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 | |
| 0xE | 128 bits | |
| 0xF | 128 bits | |

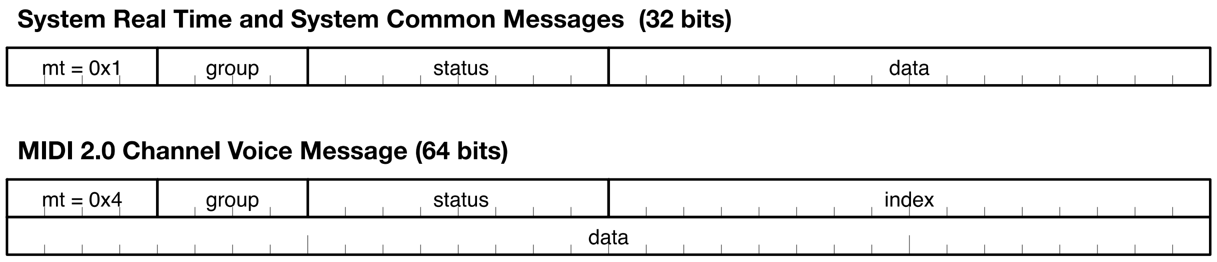

System Real Time and System Common Messages (32 bits)
mt = 0x1
group
status
data
MIDI 2.0 Channel Voice Message (64 bits)
mt = 0x4
group
status
index
data

**Figure 3 UMP Formats for Example Message Types**

### Reserved Message Types

Per ***Section 2.1.3***, Reserved Message Types marked Reserved in the table above 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.

On a per-Group basis, the Sender and Receiver shall cooperatively select exactly one MIDI Protocol at a time using either the MIDI-CI mechanism ***[MMA02]*** or additional means, as described in ***Section 3.1.2***. As a result, no Group shall use more than one MIDI Protocol at a time.

If the devices support independent selection of MIDI Protocol on a per-Group basis, then the UMP stream for that 16-Group MIDI connection might contain a mixture of different MIDI Protocols.

> ***Note:*** *In other words: A device is free to implement the MIDI 1.0 Protocol on one or more Groups while implementing the MIDI 2.0 Protocol on one or more other Groups, but no device shall send both MIDI 1.0 Protocol messages and MIDI 2.0 Protocol messages on the same Group.*

### 3.1.1 Groups, Ports, and Virtual MIDI Cables

When connecting to Non-UMP MIDI 1.0 Systems, or MIDI 2.0 systems that integrate MIDI 1.0 Protocol Devices, each of the 16 Groups may be treated like one virtual MIDI cable interleaved in a shared stream of 16 virtual MIDI cables. Each Group can be represented as a virtual MIDI port that connects to a virtual MIDI cable.

### 3.1.2 Selecting a MIDI Protocol for a Group

MIDI-CI Protocol Negotiation ***[MMA02]*** is the MIDI standard method for discovering or selecting MIDI Protocols on a per-Group basis. Some devices, interfaces, APIs, or transports might have additional means for discovering or selecting protocols on a per Group basis to fit the needs of a particular MIDI system.

#### 3.1.2.1 MIDI-CI Protocol Negotiation

MIDI-CI Protocol Negotiation may be used by MIDI Devices to agree to switch between using MIDI 1.0 Protocol messages and using MIDI 2.0 Protocol messages, on a per-Group basis. MIDI-CI is also used for selecting optional features (Extensions), including JR Timestamps. MIDI-CI Protocol Negotiation requires that the transport between the two devices be capable of using the UMP Format.

MIDI-CI Protocol Negotiation messages describe the available protocols with a set of 5 Protocol Bytes:

**Table 4 MIDI-CI Protocol Negotiation Protocol Bytes**

| Protocol Byte | Field | To Describe MIDI 1.0 | To Describe MIDI 2.0 |
|---|---|---|---|
| **1** | **Protocol Type** | 0x01: MIDI 1.0 | 0x02: MIDI 2.0 |
| **2** | **Version** | 0x00: MIDI 1.0 | 0x00: MIDI 2.0, v1.0 |
| **3** | **Extensions** | See ***Section 3.2.3*** | See ***Section 3.3.3*** |
| **4** | **Reserved** | Set to 0x00 | Set to 0x00 |
| **5** | | Set to 0x00 | Set to 0x00 |

Details of selecting the MIDI 1.0 Protocol vs. the MIDI 2.0 Protocol are shown in ***Section 3.2.3*** and ***Section 3.3.3***, respectively.

## 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

***Note:*** *However, 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 MIDI 1.0 Devices may (vs. shall not) use each of those messages.

### 3.2.3 Protocol Negotiation to the MIDI 1.0 Protocol

***Note:*** *For convenience, this Section repeats information from the MIDI-CI Specification* ***[MMA02]****. In the event of any disagreement,* ***[MMA02]*** *shall take precedence.*

**Protocol Byte 1: <u>Protocol Type</u>**

For the MIDI 1.0 Protocol, the type number is set to 0x01.

**Protocol Byte 2: <u>Version</u>**

For the MIDI 1.0 Protocol, the version number is set to 0x00.

**Protocol Byte 3: <u>Extensions</u>**

If the two devices agreeing to a MIDI-CI Protocol Negotiation are connected by a transport that supports the UMP Format, then there are defined extensions available for using the MIDI 1.0 Protocol. The Extensions field is a bitmap of flags, each representing one extension or optional feature.

The current version of MIDI-CI defines two extensions. Further extensions might be defined by MMA/AMEI in future revisions of the MIDI 1.0 Protocol or the UMP Format specification.

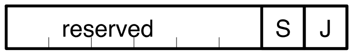


**Figure 4 MIDI-CI MIDI 1.0 Protocol Extensions Bitmap Field**

- **S: Size of UMP extension flag.** When MIDI 1.0 Protocol Devices use the UMP Format, they shall always be capable of handling UMPs of up to 64 bits in size (8 bytes).
  - When S = 0, message UMPs exchanged shall not exceed 64 bits in size.
  - When S = 1, message UMPs of 96 bits (12 bytes) and 128 bits (16 bytes) in size may also be exchanged. This larger size is necessary to support SysEx 8 and Mixed Data Set messages.
- **J: Jitter Reduction Timestamps extension flag.** When J = 1, Jitter Reduction Timestamps are supported and shall be used (i.e., shall be sent) preceding every MIDI 1.0 Protocol UMP.

Devices that report S = 0 and J = 1 shall be capable of handling UMPs up to 64 bits in size, plus 32 bits for a JR Timestamp UMP, for a total combined size of 96 bits (12 bytes).

Devices that report S = 1 and J = 1 shall be capable of handling UMPs of 128 bits in size, plus 32 bits for a JR Timestamp UMP, for a total combined size of 160 bits (20 bytes).

## 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 within a Group 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

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.

### 3.3.3 Protocol Negotiation to the MIDI 2.0 Protocol

***Note:*** *For convenience, this Section repeats information from the MIDI-CI Specification* ***[MMA02]****. In the event of any disagreement,* ***[MMA02]*** *shall take precedence.*

**Protocol Byte 1: <u>Protocol Type</u>**

For the MIDI 2.0 Protocol, the type number is set to 0x02.

**Protocol Byte 2: <u>Version</u>**

For Version 1.0 of the MIDI 2.0 Protocol, the version number is set to 0x00.

**Protocol Byte 3: <u>Extensions</u>**

The Extensions field is a bitmap of extension flags or optional features.

In this version of the MIDI 2.0 Protocol, the only extension defined is Jitter Reduction Timestamps. Further extensions might be defined by MMA/AMEI in future revisions of the MIDI 2.0 Protocol or the UMP Format specification.

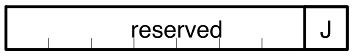


**Figure 5 MIDI-CI MIDI 2.0 Protocol Extensions Bitmap Field**

- **J: Jitter Reduction Timestamps extension flag.** When J = 1, Jitter Reduction Timestamps shall be supported/used (i.e., sent) preceding every MIDI 2.0 Protocol UMP.

***Note:*** *Devices that use the MIDI 2.0 Protocol in the UMP Format shall be capable of handling UMPs of up to 128 bits (16 bytes) in size. Devices that report J = 1 shall be capable of handling UMPs of 128 bits in size, plus 32 bits for a JR Timestamp UMP, for a total combined size of 160 bits (20 bytes).*

# 4. MIDI Messages in UMP Format

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

- ***Section 4.1 MIDI 1.0 Channel Voice Messages***
- ***Section 4.2 MIDI 2.0 Channel Voice Messages***
- ***Section 4.3 System Common and System Real Time Messages***
- ***Section 4.4 System Exclusive (7-Bit) Messages***
- ***Section 4.5 System Exclusive 8 (8-Bit) Messages***
- ***Section 4.6 Mixed Data Set Message***
- ***Section 4.8 Utility Messages***

See also:

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

## 4.1 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 6***, 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 7***, 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.

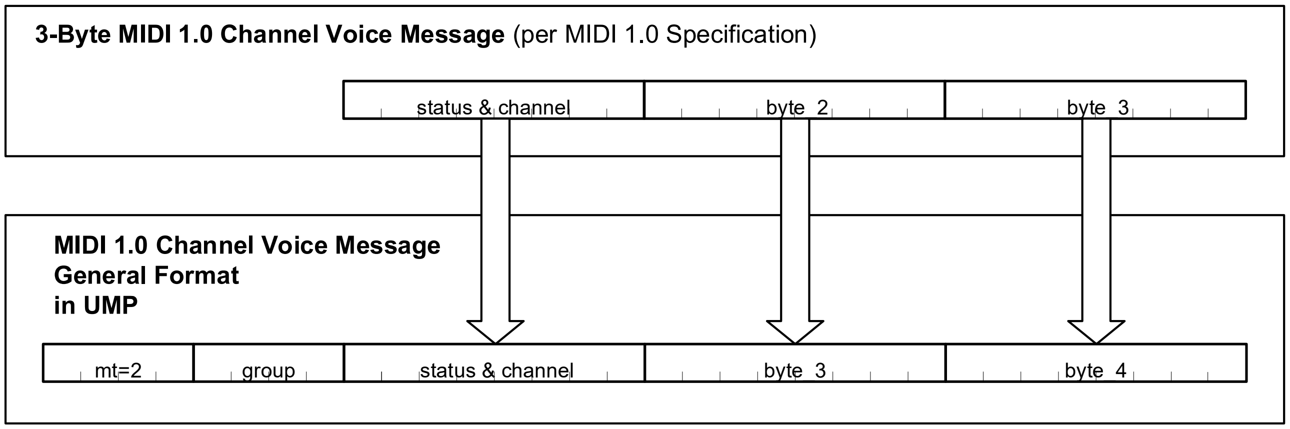

3-Byte MIDI 1.0 Channel Voice Message (per MIDI 1.0 Specification)
status & channel
byte 2
byte 3
MIDI 1.0 Channel Voice Message General Format in UMP
mt=2
group
status & channel
byte 3
byte 4

**Figure 6 MIDI 1.0 3-Byte Channel Voice Message General Format**

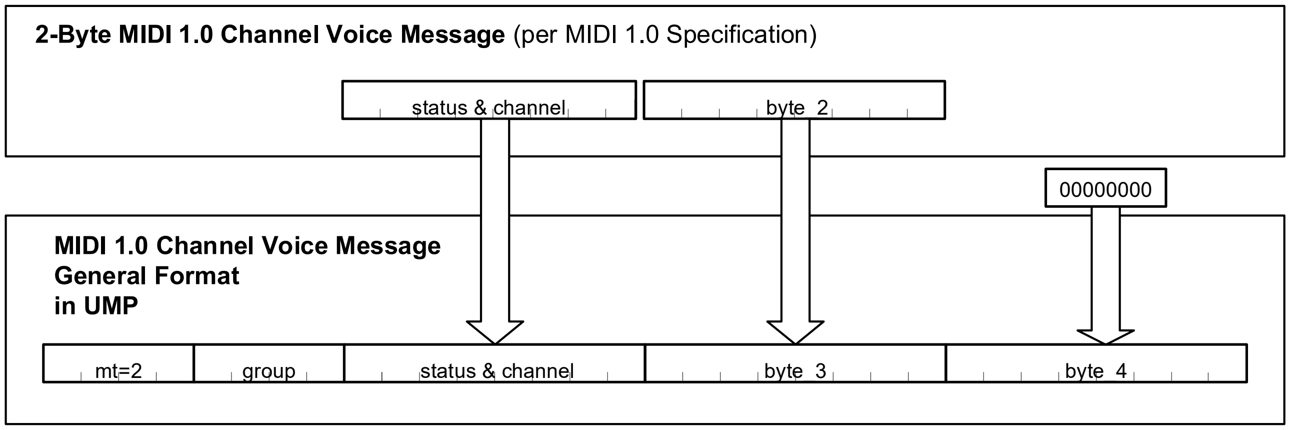

2-Byte MIDI 1.0 Channel Voice Message (per MIDI 1.0 Specification)
status & channel
byte 2
00000000
MIDI 1.0 Channel Voice Message General Format in UMP
mt=2
group
status & channel
byte 3
byte 4

**Figure 7 MIDI 1.0 2-Byte Channel Voice Message General Format**

### 4.1.1 MIDI 1.0 Note Off Message

For fundamental functions of Note Off see the MIDI 1.0 Specification ***[MMA01]***.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| mt=2 | group | 1 | 0 | 0 | 0 | channel | r | note number | r | velocity |

**Figure 8 MIDI 1.0 Note Off Message**

### 4.1.2 MIDI 1.0 Note On Message

For fundamental functions of Note On see the MIDI 1.0 Specification ***[MMA01]***.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| mt=2 | group | 1 | 0 | 0 | 1 | channel | r | note number | r | velocity |

**Figure 9 MIDI 1.0 Note On Message**

### 4.1.3 MIDI 1.0 Poly Pressure Message

For fundamental functions of Poly Pressure (Polyphonic Aftertouch) see the MIDI 1.0 Specification ***[MMA01]***.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| mt=2 | group | 1 | 0 | 1 | 0 | channel | r | note number | r | data |

**Figure 10 MIDI 1.0 Poly Pressure Message**

### 4.1.4 MIDI 1.0 Control Change Message

For fundamental functions of Control Change see the MIDI 1.0 Specification ***[MMA01]***.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| mt=2 | group | 1 | 0 | 1 | 1 | channel | r | index | r | data |

**Figure 11 MIDI 1.0 Control Change Message**

### 4.1.5 MIDI 1.0 Program Change Message

For fundamental functions of Program Change see the MIDI 1.0 Specification ***[MMA01]***.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mt=2 | group | 1 | 1 | 0 | 0 | channel | r | program | reserved |

**Figure 12 MIDI 1.0 Program Change Message**

### 4.1.6 MIDI 1.0 Channel Pressure Message

For fundamental functions of Channel Pressure (Channel Aftertouch) see the MIDI 1.0 Specification ***[MMA01]***.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mt=2 | group | 1 | 1 | 0 | 1 | channel | r | data | reserved |

**Figure 13 MIDI 1.0 Channel Pressure Message**

### 4.1.7 MIDI 1.0 Pitch Bend Message

For fundamental functions of Pitch Bend see the MIDI 1.0 Specification ***[MMA01]***.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| mt=2 | group | 1 | 1 | 1 | 0 | channel | r | lsb data | r | msb data |

**Figure 14 MIDI 1.0 Pitch Bend Message**

## 4.2 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)

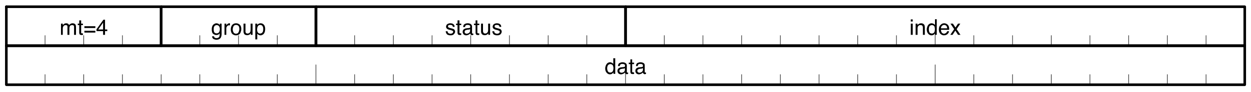


**Figure 15 MIDI 2.0 Channel Voice Message General Format**

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.

### 4.2.1 MIDI 2.0 Note Off Message

For fundamental functions of Note Off see the MIDI 1.0 Specification ***[MMA01]***.

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

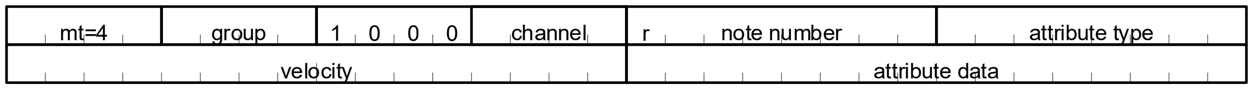


**Figure 16 MIDI 2.0 Note Off Message**

For more information, see:

***Section 4.2.2 MIDI 2.0 Note On Message***
***Section 4.2.13 MIDI 2.0 Note On/Off: Attribute Type & Attribute Data***

### 4.2.2 MIDI 2.0 Note On Message

For fundamental functions of Note On see the MIDI 1.0 Specification ***[MMA01]***.

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

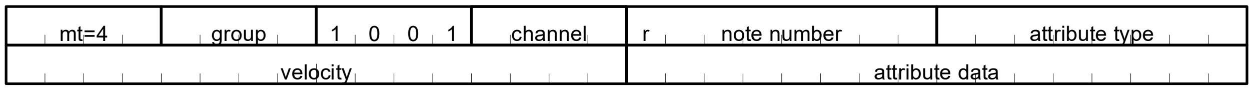


**Figure 17 MIDI 2.0 Note On Message**

**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)**

For more information, see:

***Section 4.2.13 MIDI 2.0 Note On/Off: Attribute Type & Attribute Data***
***Section 4.2.14 MIDI 2.0 Notes and Pitch***

### 4.2.3 MIDI 2.0 Poly Pressure Message

For fundamental functions of Poly Pressure (Polyphonic Aftertouch) see the MIDI 1.0 Specification ***[MMA01]***.

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

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mt=4 | group | 1 | 0 | 1 | 0 | channel | r | note number | reserved |
| data | | | | | | | | | |

**Figure 18 MIDI 2.0 Poly Pressure Message**

### 4.2.4 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:

- The 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***.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mt=4 | group | 0 | 0 | 0 | 0 | channel | r | note number | Index |
| data | | | | | | | | | |

**Figure 19 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.*

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

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| mt=4 | group | 0 | 0 | 0 | 1 | channel | r | note number | Index |
| data | | | | | | | | | |

**Figure 20 MIDI 2.0 Assignable Per-Note Controller Message**

### 4.2.5 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.

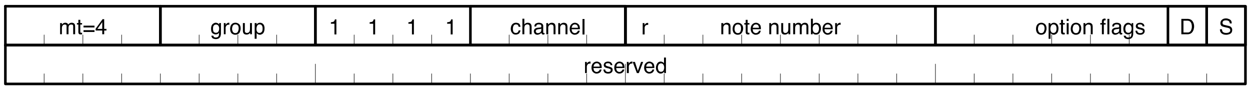


**Figure 21 MIDI 2.0 Per-Note Management Message**

**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.

### 4.2.6 MIDI 2.0 Control Change Message

For fundamental functions of Control Change see the MIDI 1.0 Specification ***[MMA01]***.

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

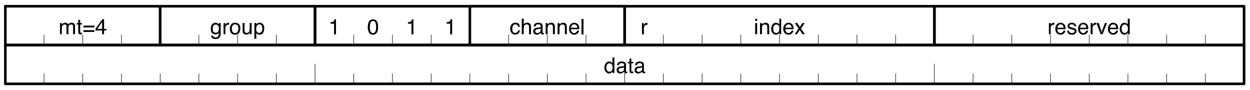


**Figure 22 MIDI 2.0 Control Change Message**

***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 4.2.7 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 4.2.9 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 4.2.7***). 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 4.2.9***).
- Devices receiving the MIDI 2.0 Protocol should ignore Control Change messages with indexes of 0, 6, 32, 38, 98, 99, 100, and 101.

### 4.2.7 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 ***Appendix 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).

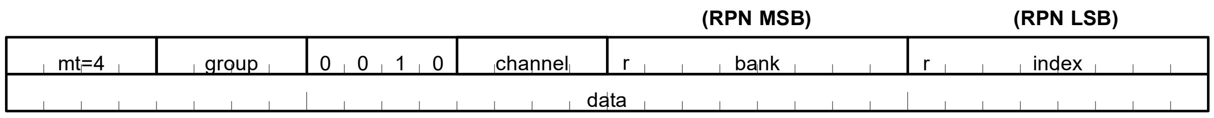


**Figure 23 MIDI 2.0 Registered Controller Message**

- **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).

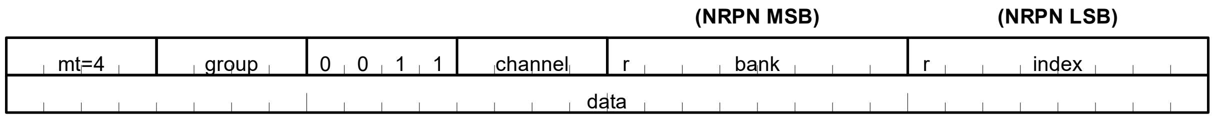


**Figure 24 MIDI 2.0 Assignable Controller Message**

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.

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

Registered Controller Messages and Assignable Controller Messages (defined above in ***Section 4.2.7***) 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.

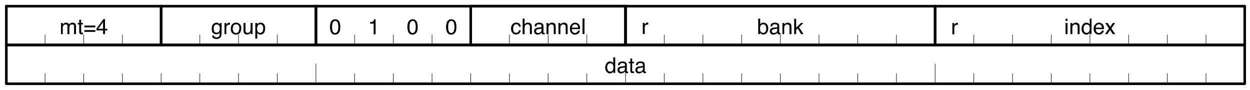


**Figure 25 MIDI 2.0 Relative Registered Controller Message**

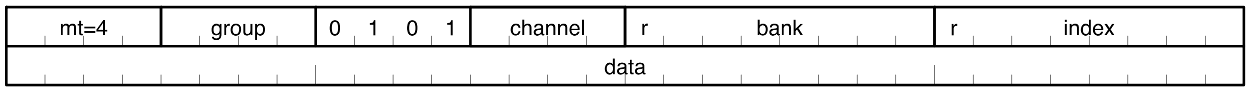


**Figure 26 MIDI 2.0 Relative Assignable Controller Message**

**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.

### 4.2.9 MIDI 2.0 Program Change Message

For fundamental functions of Program Change and Bank Select see the MIDI 1.0 Specification ***[MMA01]***.

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.

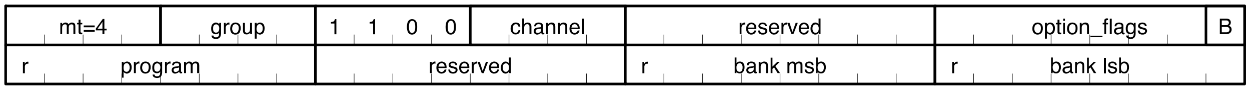


**Figure 27 MIDI 2.0 Program Change Message**

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.

### 4.2.10 MIDI 2.0 Channel Pressure Message

For fundamental functions of Channel Pressure (Channel Aftertouch) see the MIDI 1.0 Specification ***[MMA01]***.

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

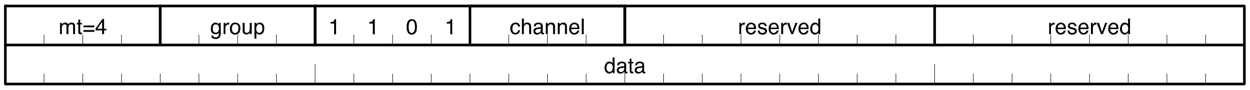


**Figure 28 MIDI 2.0 Channel Pressure Message**

### 4.2.11 MIDI 2.0 Pitch Bend Message

For fundamental functions of Pitch Bend see the MIDI 1.0 Specification ***[MMA01]***.

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.

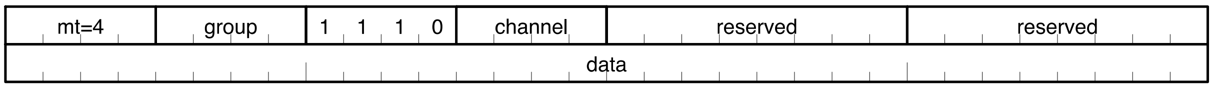


**Figure 29 MIDI 2.0 Pitch Bend Message**

### 4.2.12 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.

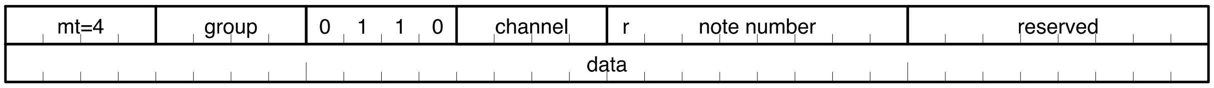


**Figure 30 MIDI 2.0 Per-Note Pitch Bend Message**

### 4.2.13 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 5 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<br>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 4.2.14.3*** |
| 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**

A MIDI-CI Profile ***[MMA02]*** might optionally specify its own use for the Attribute Type and Attribute data fields. When such a Profile is in use, the Sender shall set the Attribute Type field to 0x02, and the Sender and Receiver shall behave as required by the Profile. This mechanism is intended for Profiles that might be less commonly used and do not warrant the dedication of a whole MIDI 2.0 Attribute Type.

***Note:*** *Alternatively, a Profile might define another Attribute Type that is defined for more specific use by that one Profile only.*

The application of an Attribute Type value might be defined by MMA/AMEI in a MIDI-CI Profile specification. For example, a drum Profile might define an Attribute Type as “Strike Position” with the Attribute Data value declaring the position from center of drum/cymbal to outer edge. An orchestral string Profile might define Attribute values to be used as Articulation choice such as Arco, Pizzicato, Spiccato, Tremolo, etc. Such cases generally require assigning 1 of the 256 available Attribute Types for use by that Profile. Some Profiles might be able to share some common Attribute types.

**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 semitones. See ***Section 4.2.14.3*** for implementation details, including interaction with other messages that influence or determine pitch.

### 4.2.14 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:
    - Master 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.*

#### 4.2.14.1 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 ***[MMA01]***.

#### 4.2.14.2 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 semitones, based on default Note Number equal temperament scale
- **25 bits: Fractional Pitch** above Note Number (i.e., fraction of one semitone)

Pitch is a Q7.25 fixed-point unsigned integer that specifies a pitch in semitones. 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 ***[MMA01]*** in a 12-tone equal tempered scale with A=440 (Note number 69 [0x45]). The fractional part is a fraction of one semitone.

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.

#### 4.2.14.3 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 semitones, based on default Note Number equal temperament scale
- 9 bits: Fractional pitch above Note Number (i.e, fraction of one semitone)

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 semitones. 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 ***[MMA01]*** in a 12-tone equal tempered scale with A=440 (Note number 69 [0x45]). The fractional part is a fraction of a semitone. That has a resolution of 1/512 semitones, 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.

## 4.3 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 ***[MMA01]***.

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 4.4***. 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.

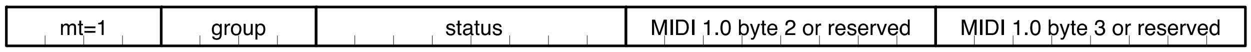


**Figure 31 System Message General Format**

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

**Table 6 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.*

## 4.4 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.

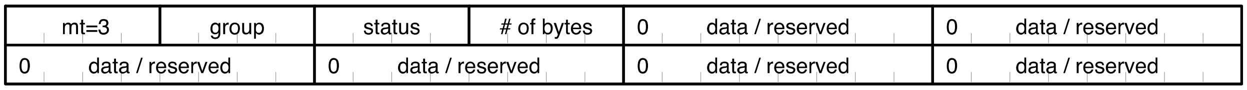


**Figure 32 System Exclusive (7-Bit) Message Format**

<u>**status**</u>

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

**Table 7 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<br>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.

<u>**# of bytes**</u>

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.*

### 4.4.1 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 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 and JR Clock Messages may be inserted between the UMPs of a System Exclusive message, in order to maintain timing synchronization.
- If any Message or UMP other than 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.

## 4.5 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 4.4**) 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 4.7***) for System Exclusive 8 messages.

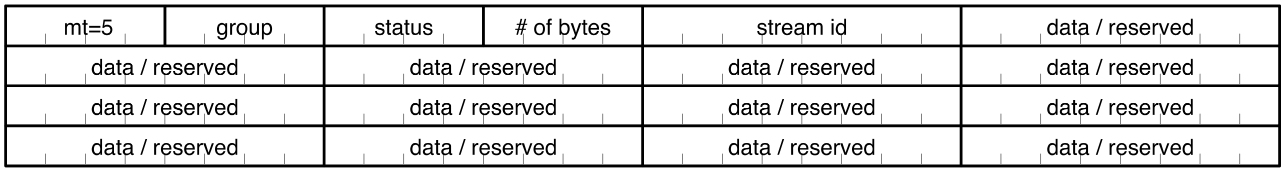


**Figure 33 System Exclusive 8 (8-Bit) Message Format**

**<u>status</u>**

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

**Table 8 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<br>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.

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, then the Sender shall first perform a MIDI-CI Property Exchange inquiry to determine how many simultaneous Stream IDs are supported by the Receiver (N). If either the Sender or the Receiver does not support Property Exchange 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 MIDI-CI Property Exchange inquiry to determine how many simultaneous Stream IDs are supported by the Receiver (N).

### 4.5.1 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.
- **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 4.4.1****).*

## 4.6 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 4.4**) for compatibility across a wide range of MIDI devices, or use System Exclusive 8 (8-Bit) Messages (**Section 4.5**) 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.

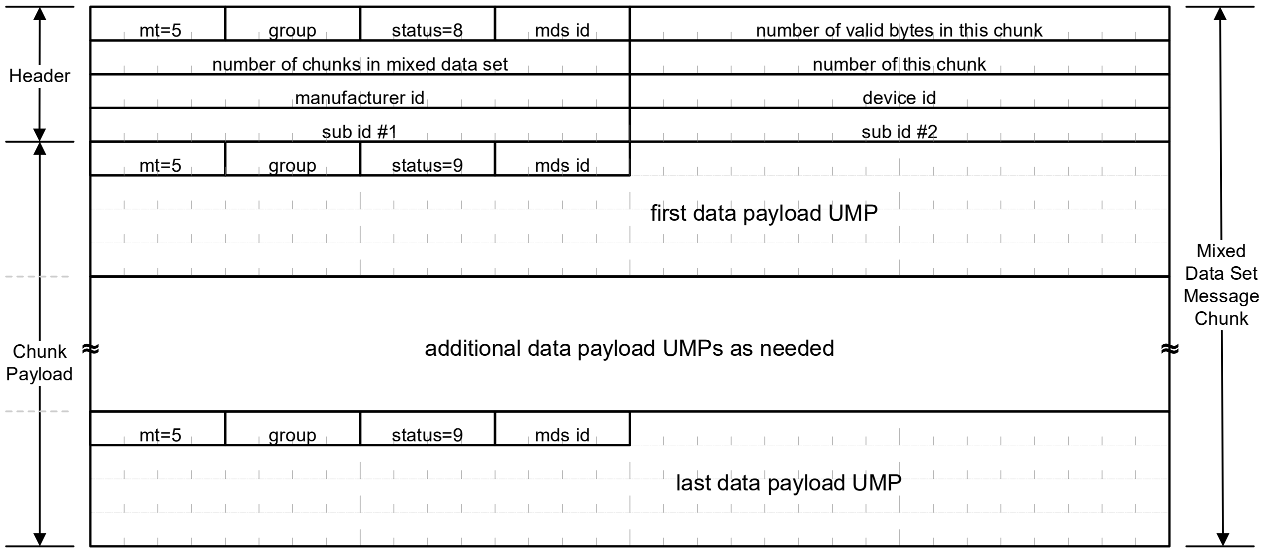


**Figure 34 Mixed Data Set Chunk Format**

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

**mt Message Type**

**0x5:** Data Messages

**group**

Group

**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 4.6.1** 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 4.7***.

**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 ***[MMA01]***.

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 ***[MMA01]***.

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 ***[MMA01]***.

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

### 4.6.1 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 4.4.1***.

## 4.7 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.

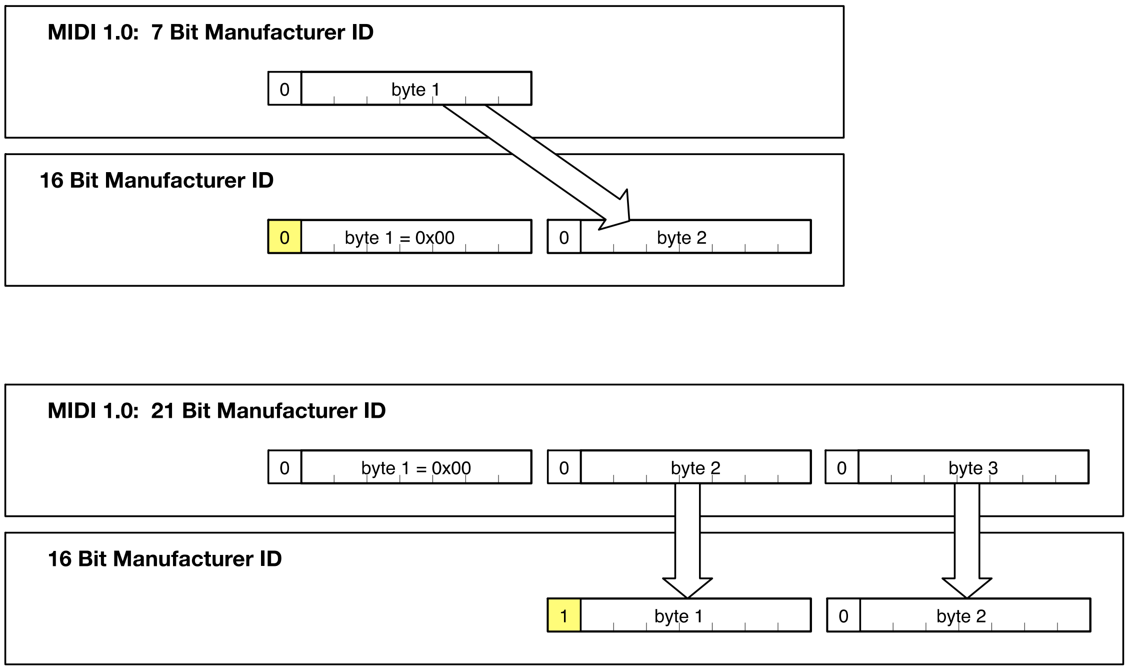


**Figure 35 Manufacturer ID Translations**

### 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 9 16-Bit Values for 7-Bit Special IDs**

| Special ID | 7-Bit Value | 16-Bit Value |
|---|---|---|
| **Non-Commercial / Research**<br>*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 10 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 |

## 4.8 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.

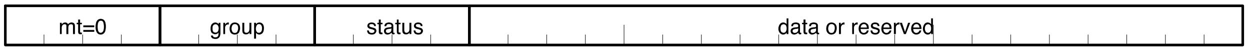


**Figure 36 Utility Message General Format**

### 4.8.1 NOOP

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

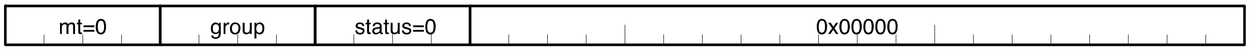


**Figure 37 NOOP Message Format**

### 4.8.2 Basic Timestamp Format

Timestamp messages in Message Type 0 can be either stand-alone messages, or prepended to any non-Timestamp Message. When the Timestamp is prepended to another message, the Timestamp message is sent in a separate UMP which is prepended to another UMP.

The Status field describes the application of the message and the contents, semantics, and application of the Timestamp Data field, whether stand alone or prepended to another message.

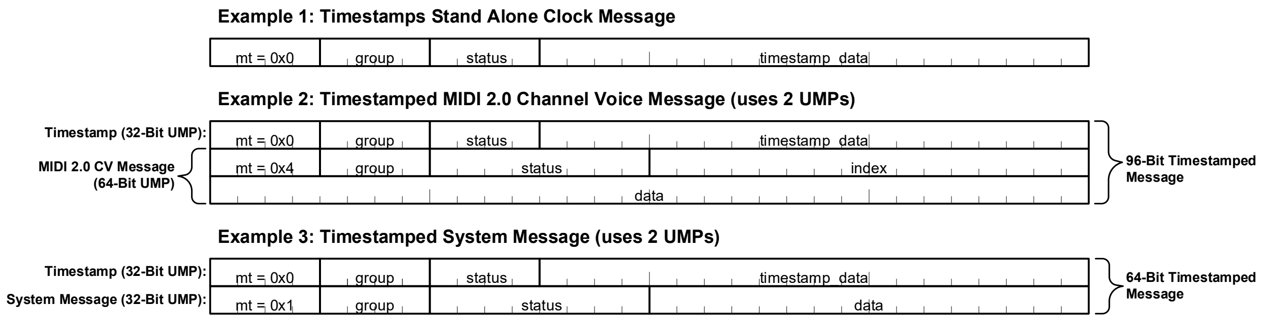

Example 1: Timestamps Stand Alone Clock Message
mt = 0x0
group
status
timestamp data
Example 2: Timestamped MIDI 2.0 Channel Voice Message (uses 2 UMPs)
Timestamp (32-Bit UMP):
mt = 0x0
group
status
timestamp data
MIDI 2.0 CV Message (64-Bit UMP)
mt = 0x4
group
status
index
data
96-Bit Timestamped Message
Example 3: Timestamped System Message (uses 2 UMPs)
Timestamp (32-Bit UMP):
mt = 0x0
group
status
timestamp data
System Message (32-Bit UMP):
mt = 0x1
group
status
data
64-Bit Timestamped Message

**Figure 38 Timestamp Format Examples**

### 4.8.3 Jitter Reduction (JR) Timestamps (and JR Clock)

#### Timestamps with Status Set to 0x1 and 0x2

This mechanism defines simple clock synchronization for jitter reduction:

- Sender sends clock messages from time to time so that the Receiver knows the current Sender's time
- Clock messages allow the Receiver to estimate the maximum jitter, and to continuously adapt to drift
- Sender can precisely specify the timestamp for every non-timestamp message: the render time (in Sender's time) of the following message(s)
- This is a simple, peer to peer mechanism, not a system-wide synchronization

Goals of JR Timestamps:

1. Capture a performance with accurate timing
2. Transmit MIDI Messages with accurate timing over a system that is subject to jitter
3. Does not depend on system-wide synchronization, master clock, or explicit clock synchronization between Sender and Receiver.

***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.*

### 4.8.4 MIDI-CI Protocol Negotiation and JR Timestamps

MIDI-CI Protocol Negotiation allows devices to agree to use a MIDI Protocol without JR Timestamps, or a MIDI Protocol with JR Timestamps. Using the MIDI-CI Protocol Negotiation mechanism, JR Timestamps are only used when both devices indicate support for JR Timestamps.

MIDI-CI Protocol Negotiation determines the choice of protocol and use of JR Timestamps in both directions between two devices. If JR Timestamps are being used between two devices, they shall be used bidirectionally.

If devices agree to use JR Timestamps, then the devices shall continue to use JR Timestamps with every message exchanged in both directions until a new MIDI-CI Protocol Negotiation is performed. If devices agree to use a MIDI Protocol without JR Timestamps, then neither device shall send JR Timestamps.

Each Group has its own JR Timestamp time domain, based on the time of the Sender using that Group. JR Timestamps are in the time domain of the Sender. While this is implementation specific, it is likely that a Sender will use a single, common source clock when sending to multiple Groups, so JR Timestamps would all be within the same time domain. If a Receiver cannot handle multiple JR Timestamps time domains, from multiple Senders on multiple Groups, then it should negotiate to use the protocol with JR Timestamps on only one Group.

If a device supports JR Timestamps, then it shall also support operation without them.

### 4.8.5 JR Clock Message Format

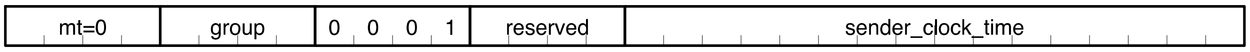


**Figure 39 JR Clock Message Format**

**status**

0x1, JR Clock

**reserved**

Reserved for future definition by MMA/AMEI. It shall be set to zero by the Sender, and ignored by the Receiver.

**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.

### 4.8.6 JR Timestamp Message Format

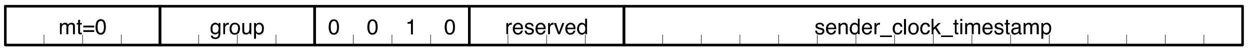


**Figure 40 JR Timestamp Message Format**

**status**

0x2, JR Timestamp

**reserved**

Reserved for future definition by the MMA/AMEI. It shall be set to zero by the Sender, and ignored by the Receiver.

**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).

### 4.8.7 JR Clock Mechanism

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.

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).

### 4.8.8 JR Timestamp Mechanism

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.

JR Timestamps are specified in the Sender's clock domain as communicated via JR Clock Messages. For real-time scheduling, the Receiver should convert the time for each message from the Sender's clock domain to the Receiver's clock 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).

#### 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 initiate a MIDI-CI Protocol Negotiation to switch the Sender to a protocol without JR Timestamps.

### 4.8.9 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.

#### 4.8.10 Translation to/from the MIDI 1.0 Protocol

This specification does not require all Translators to support JR Timestamps. If a Translator supports JR Timestamps the Translator relies on MIDI-CI Protocol Negotiation to determine whether to use JR Timestamps on each connection.

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. 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.

# 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 11 MIDI 2.0 Registered Per-Note Controllers**

| RPNC Number | Registered Per-Note Controller Name | Default | Reference |
|---|---|---|---|
| 1 | Modulation | – | – |
| 2 | Breath | – | – |
| 3 | Pitch 7.25 | – | ***Section 4.2.14.2*** |
| 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 ***[MMA04]*** |
| 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 ***[MMA05]*** |
| 92 | Effects 2 Depth *(formerly Tremolo Depth)* | – | – |
| 93 | Effects 3 Depth | Chorus Send Level | MMA RP-023 ***[MMA05]*** |
| 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 ***[MMA01]*** 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 ***[MMA01]*** 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 ***[MMA01]***.

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 41*** shows a typical example where the trailing envelope of Note A shares the Per-note Controllers that are also controlling Note B.

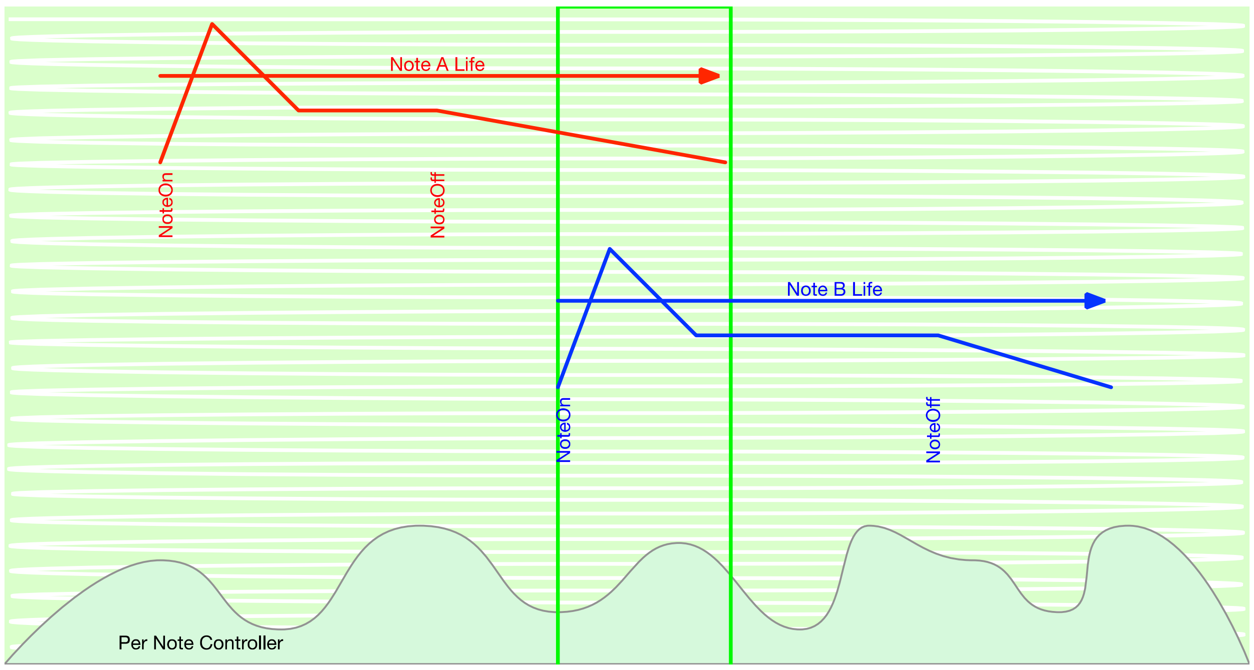


**Figure 41 Two Notes of Same Note Number Share Per-Note Controllers**

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 42***). 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.

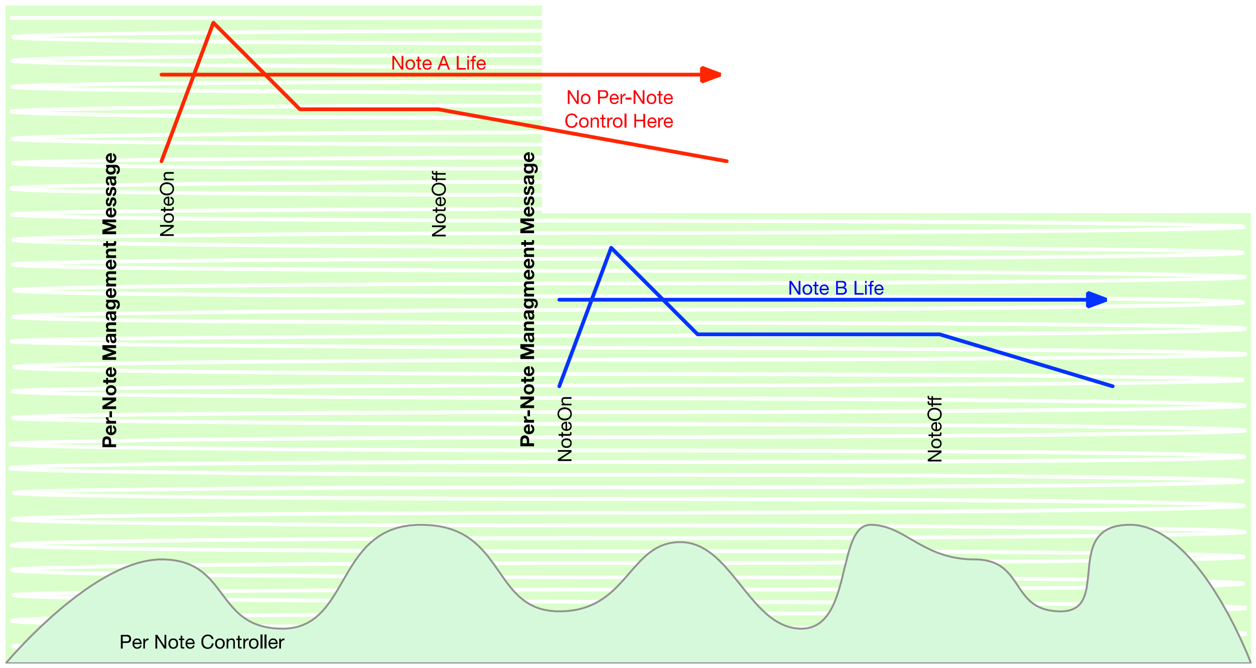


**Figure 42 Only the Note After the Per-Note Management Message has Per-Note Control**

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 43 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

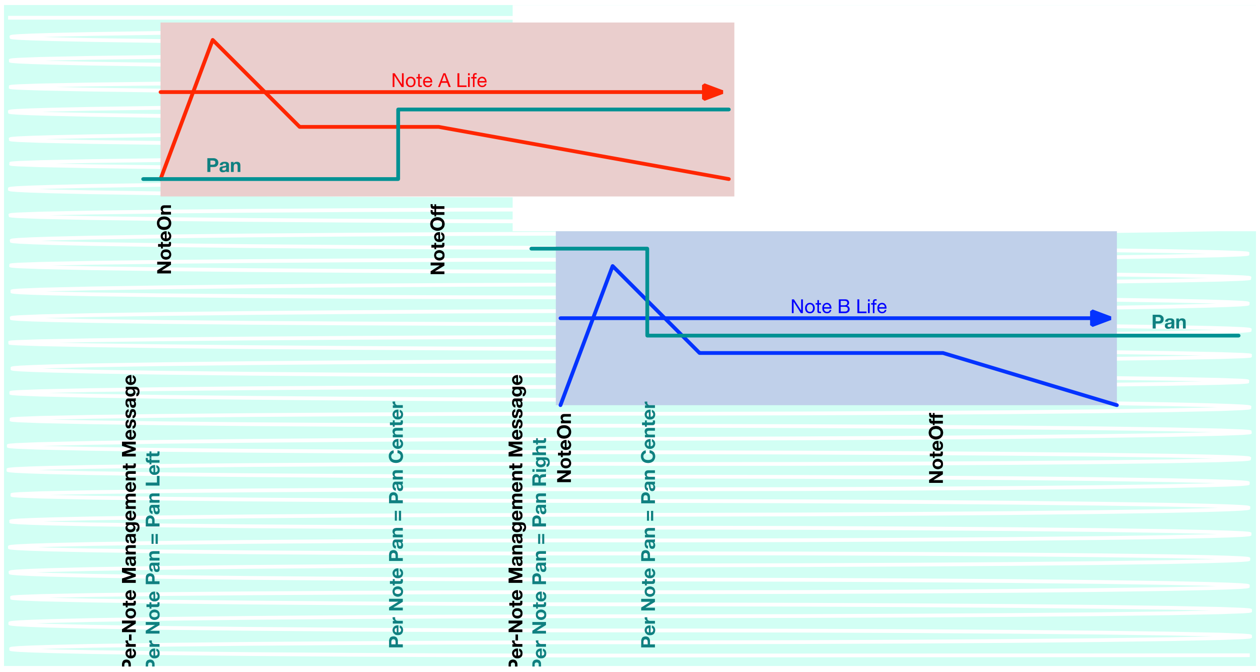

Note A Life
Pan
NoteOn
NoteOff
Per-Note Management Message
Per Note Pan = Pan Left
Per Note Pan = Pan Center
Note B Life
Pan
Per-Note Management Message
Per Note Pan = Pan Right
NoteOn
Per Note Pan = Pan Center
NoteOff

**Figure 44 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 Semitone 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, in order 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.

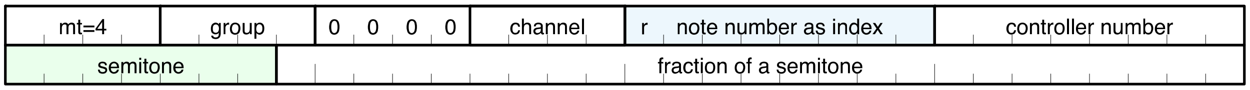


**Figure 45 MIDI 2.0 Registered Per-Note Controller Message with Controller #3 (Pitch 7.25)**

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.

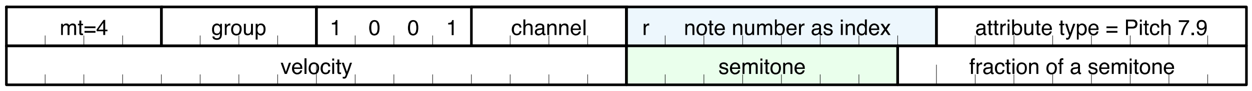


**Figure 46 MIDI 2.0 Note On Message with Attribute #3 (Pitch 7.9)**

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.

### 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 is translated to Minimum/Lowest
- Maximum/Highest value 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 12 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 Upscaling Translation Methods:

For upscaling values to higher resolution, use this algorithm:

- 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.

**Pseudo Code for the Upscaling Algorithm**

(Optimized for readability, not efficiency.)

```
scaleUp(srcVal, srcBits, dstBits) {
        // simple bit shift
        uint scaleBits = (dstBits - srcBits);
        uint bitShiftedValue = srcVal << scaleBits;
        uint srcCenter = 2^(srcBits-1);
        if (srcVal <= srcCenter ) {
                return bitShiftedValue;
        }
        // expanded bit repeat scheme
        uint repeatBits = srcBits - 1;
        uint repeatMask = (2^repeatBits) - 1;
        uint repeatValue = srcVal & repeatMask;
        if (scaleBits > repeatBits) {
                repeatValue <<= scaleBits - repeatBits;
        } else {
                repeatValue >>= repeatBits - scaleBits;
        }
        while (repeatValue != 0) {
                bitShiftedValue |= repeatValue;
                repeatValue >>= repeatBits;
        }
        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.

### Pseudo Code for Scaling Up from 7-Bit to 16-Bit

```
uint16 convert7to16(uint7 value7) {
    uint16 bitShiftedValue = value7 << 9;
    if (value7 <= 64) {
        return bitShiftedValue;
    }
    // use bit repeat bits from extended value7
    uint6 repeatValue6 = value7 & 0x3F;
    return bitShiftedValue
              | (repeatValue6 << 3)
              | (repeatValue6 >> 3);
}
```

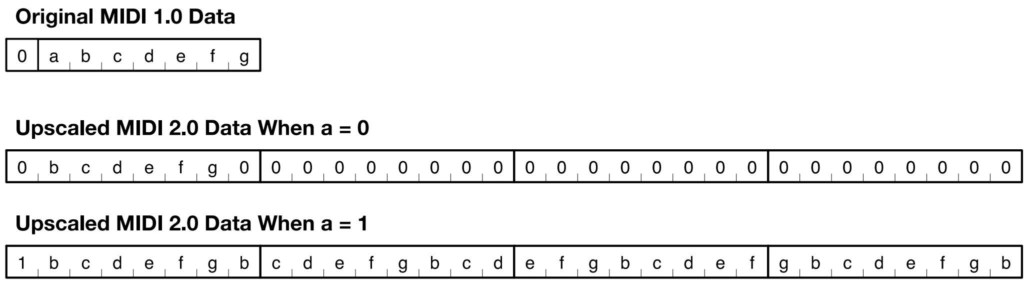

Original MIDI 1.0 Data
0 a b c d e f g
Upscaled MIDI 2.0 Data When a = 0
0 b c d e f g 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Upscaled MIDI 2.0 Data When a = 1
1 b c d e f g b c d e f g b c d e f g b c d e f g b c d e f g b

**Figure 47 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.

### Pseudo Code for Downscaling Algorithm

```
scaleDown(srcVal, srcBits, dstBits) {
      // simple bit shift
      uint 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

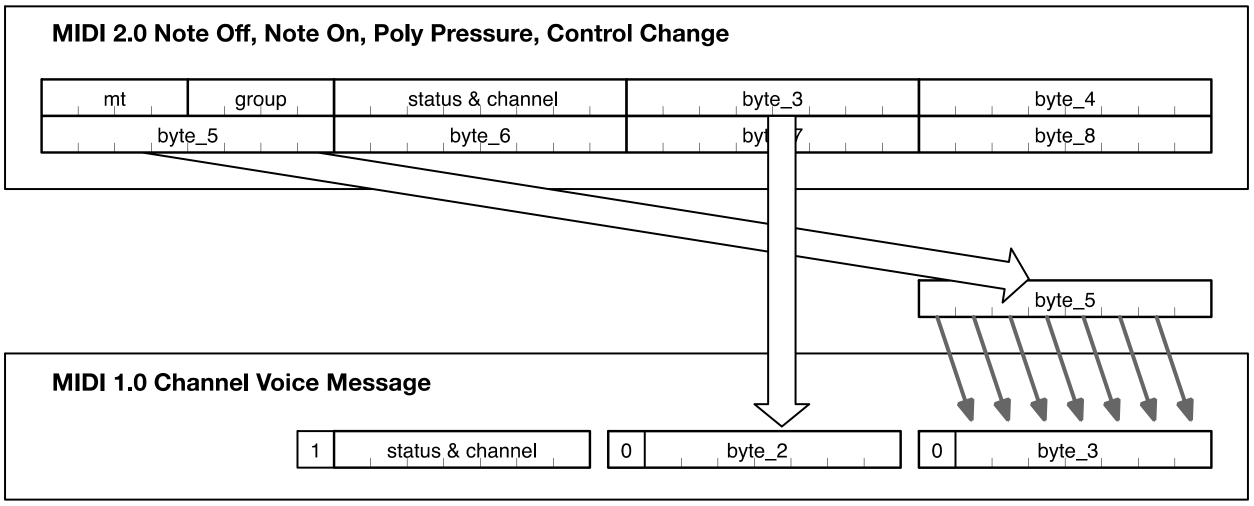


**Figure 48 Translate MIDI 2.0 Note Off, Note On, Poly Pressure, and Control Change to MIDI 1.0**

#### 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 0, which has the same function as a Note Off. Therefore, if the translated MIDI 1.0 value of the Velocity is 0, replace the value with 1. If translation to MIDI 1.0 High Resolution Velocity Prefix (using Control Change #88, see MMA/AMEI CA#031 ***[MMA03]***) is supported, then the minimum combined value for the 14-bit velocity is 0x0080.

### D.2.2 Channel Pressure

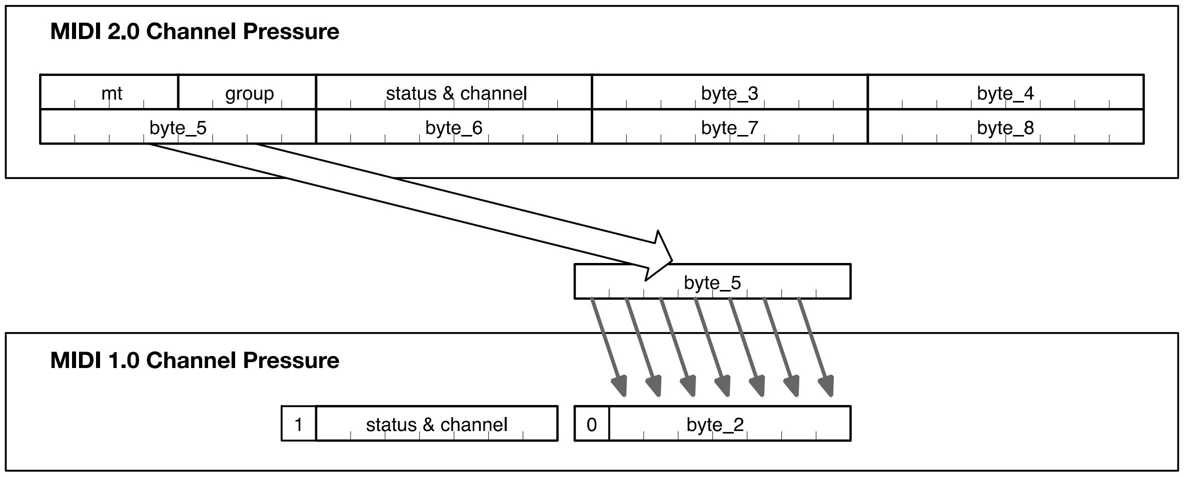


**Figure 49 Translate MIDI 2.0 Channel Pressure to MIDI 1.0**

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

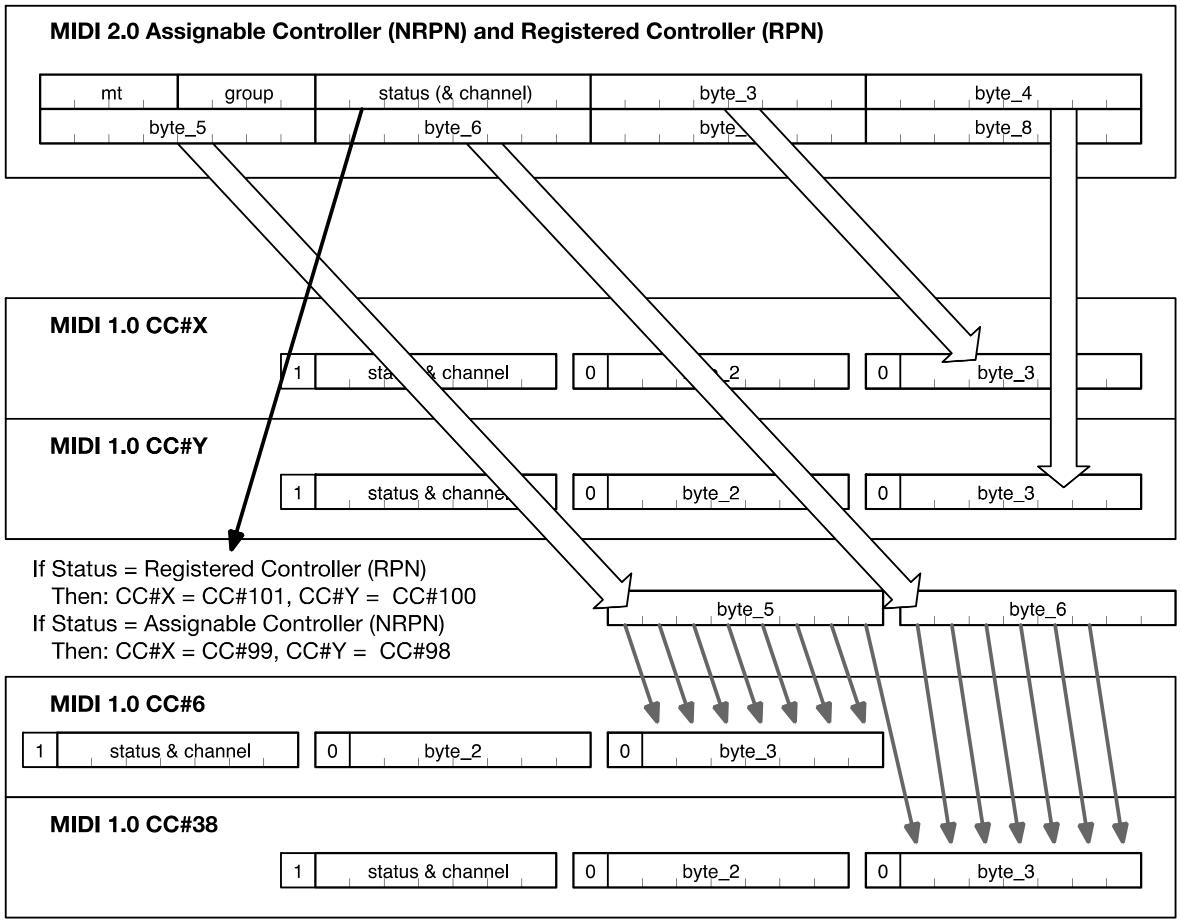


**Figure 50 Translate MIDI 2.0 Assignable (NRPN) and Registered (RPN) Controller to MIDI 1.0**

#### 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

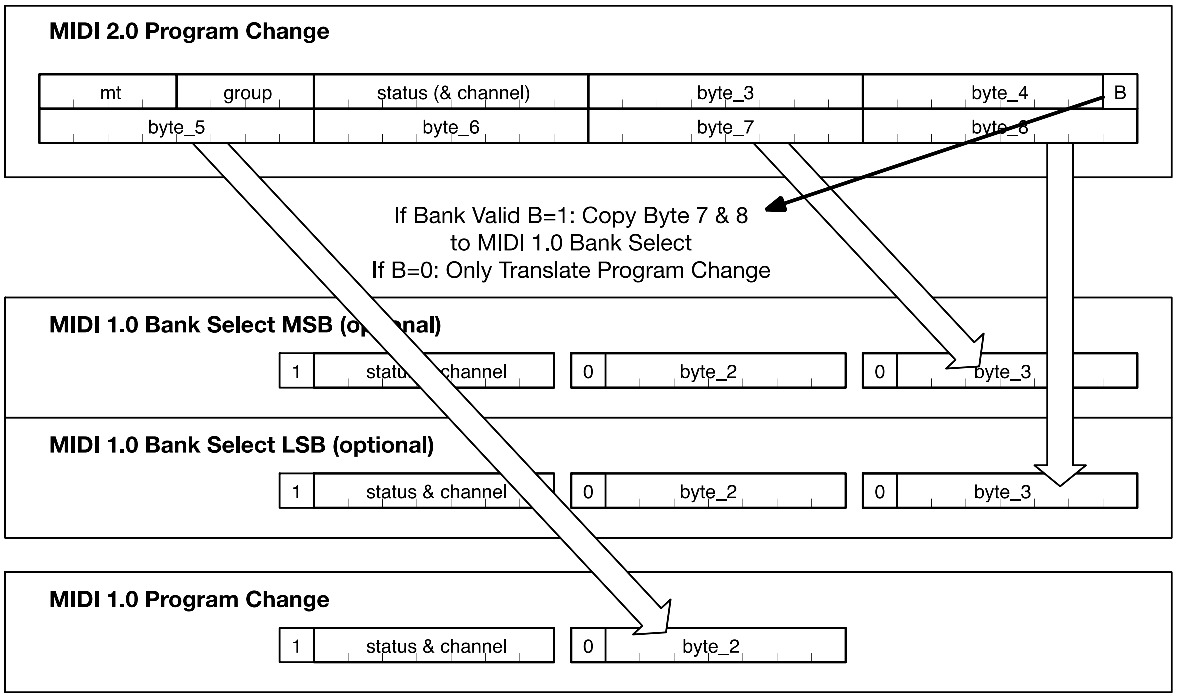


**Figure 51 Translate MIDI 2.0 Program Change to MIDI 1.0**

### 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:

```
Bank Select MSB
Bank Select LSB
Program Change
```

### D.2.5 Pitch Bend

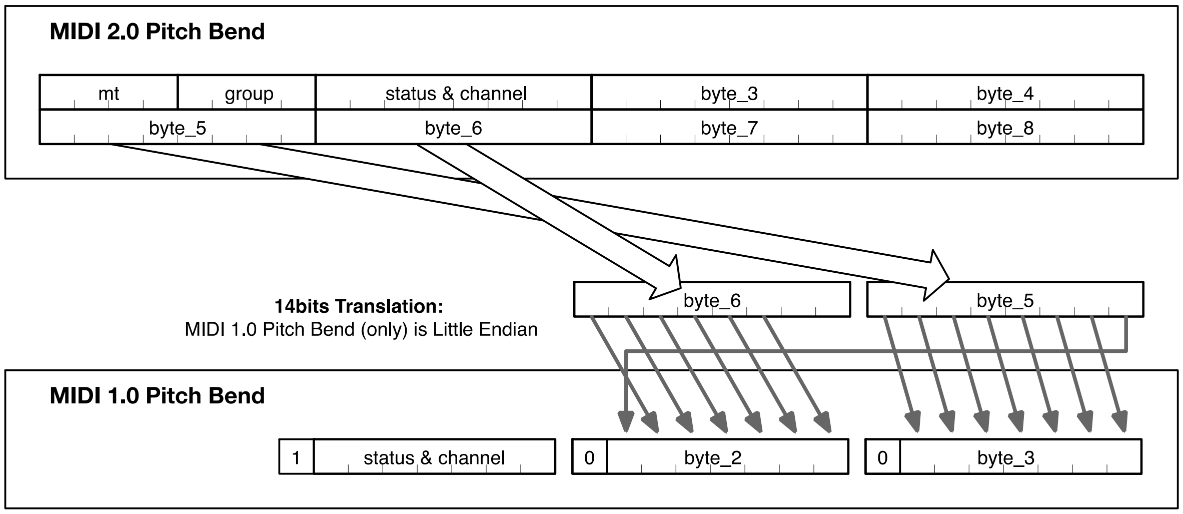


**Figure 52 Translate MIDI 2.0 Pitch Bend to MIDI 1.0**

Note that Pitch Bend values in the MIDI 1.0 Protocol are presented as Little Endian.

### D.2.6 System Messages

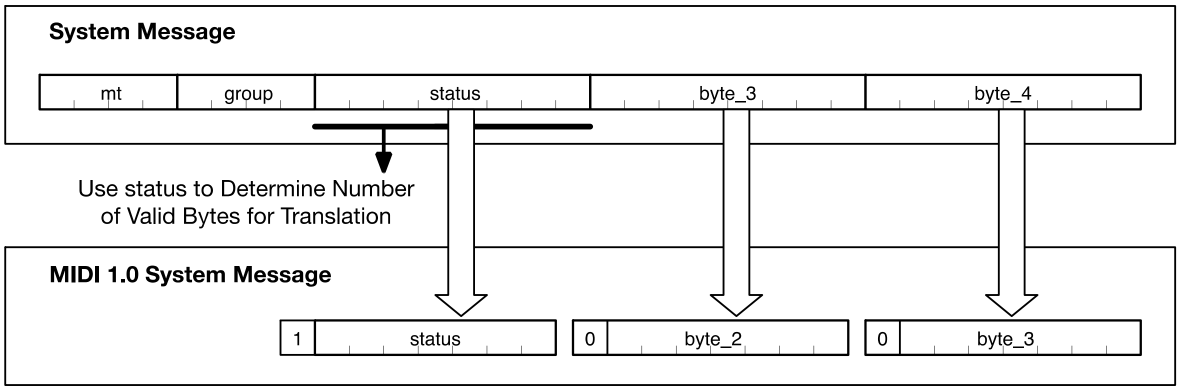


**Figure 53 Translate MIDI 2.0 System Message to MIDI 1.0**

### 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:

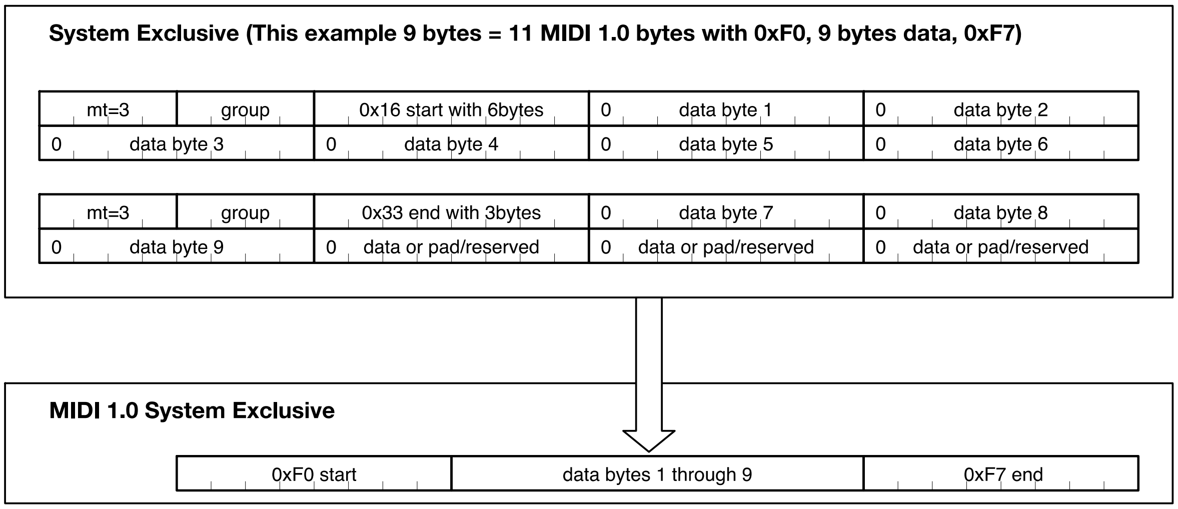


**Figure 54 Translate MIDI 2.0 System Exclusive to MIDI 1.0**

### 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

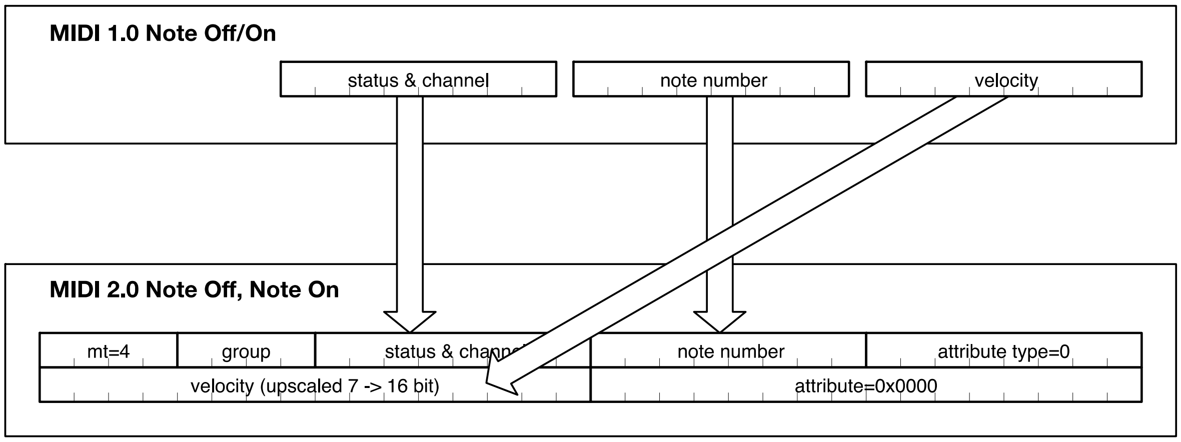


**Figure 55 Translate MIDI 1.0 Note On and Note Off to MIDI 2.0**

#### 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 0x0000.

**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

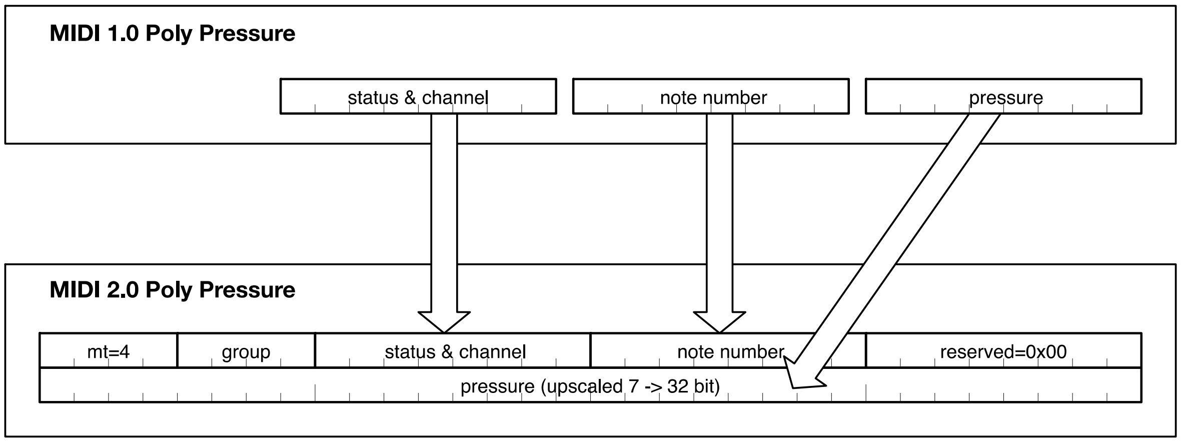


**Figure 56 Translate MIDI 1.0 Poly Pressure to MIDI 2.0**

### D.3.3 Control Change, RPN, and NRPN

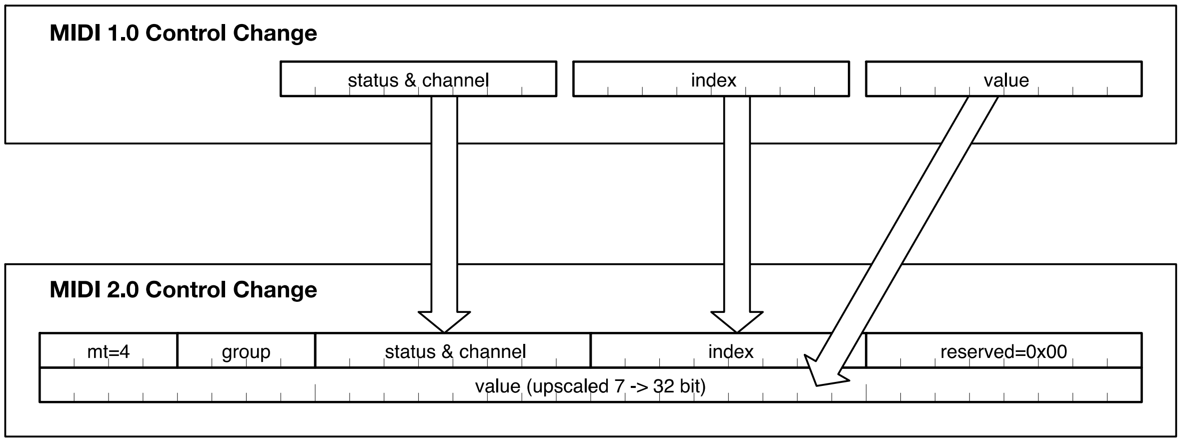


**Figure 57 Translate MIDI 1.0 Control Change to MIDI 2.0**

#### Control Change Messages for RPN/NRPN

- MIDI 1.0 Protocol Inc/Dec 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.
- 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 until a CC#38 is received. 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:

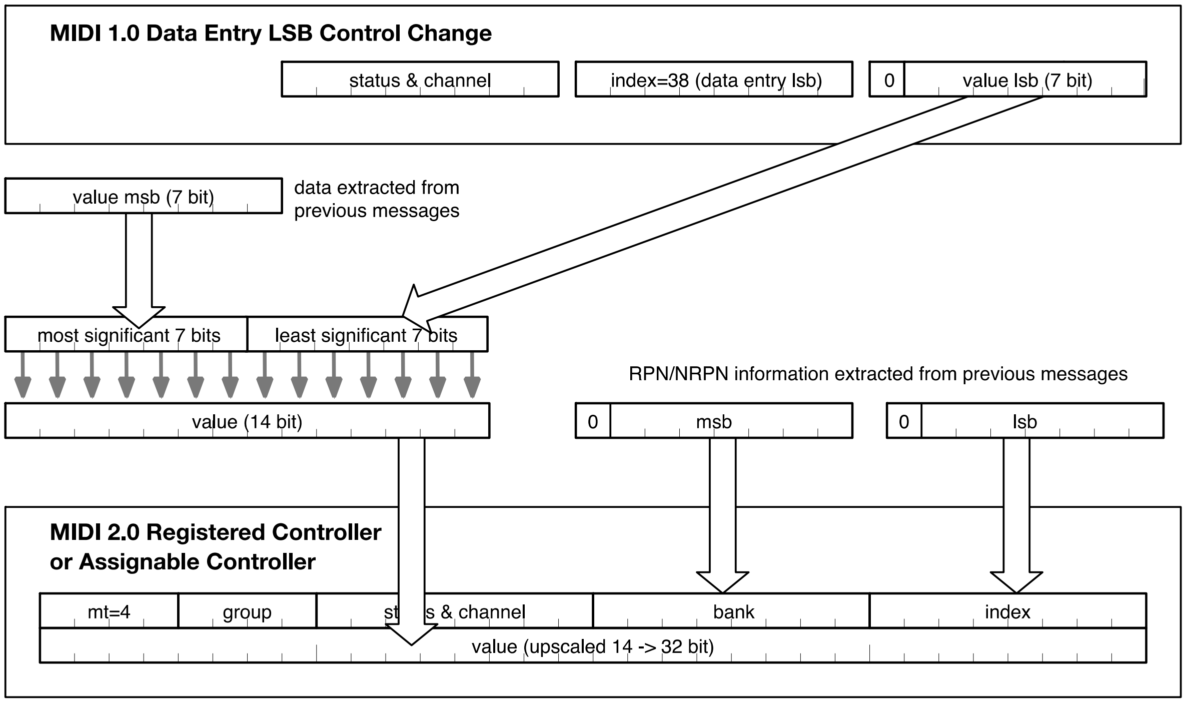

MIDI 1.0 Data Entry LSB Control Change
status & channel
index=38 (data entry lsb)
0
value lsb (7 bit)
value msb (7 bit)
data extracted from previous messages
most significant 7 bits
least significant 7 bits
RPN/NRPN information extracted from previous messages
value (14 bit)
0
msb
0
lsb
MIDI 2.0 Registered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

**Figure 58 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.

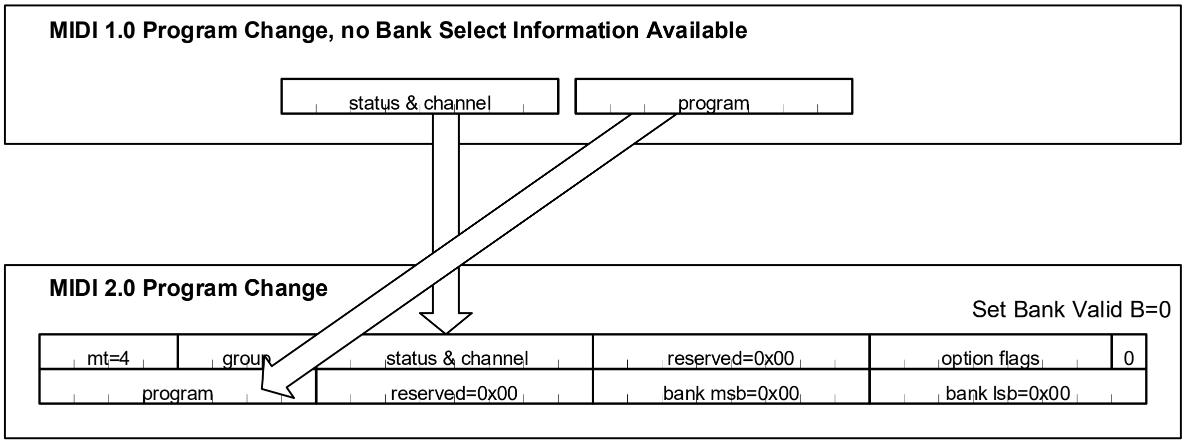


**Figure 59 Translate MIDI 1.0 Program Change to MIDI 2.0 (No Bank)**

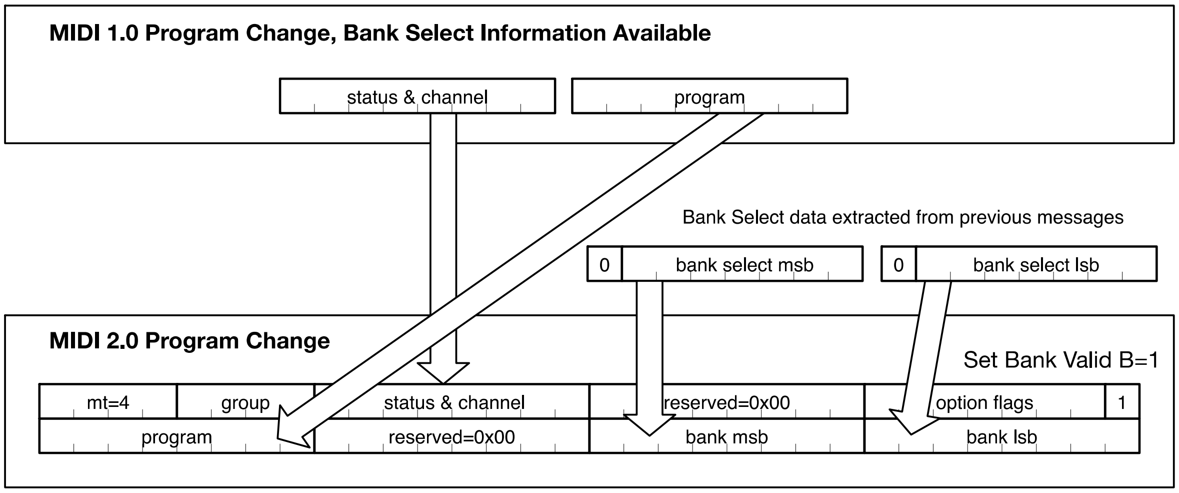


**Figure 60 Translate MIDI 1.0 Bank and Program Change to MIDI 2.0**

### D.3.5 Channel Pressure

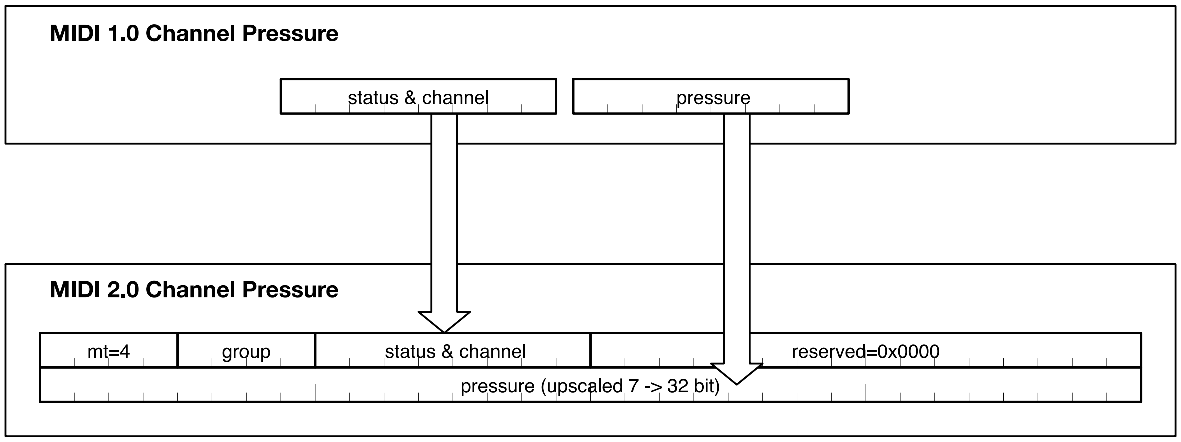

MIDI 1.0 Channel Pressure
status & channel
pressure
MIDI 2.0 Channel Pressure
mt=4
group
status & channel
reserved=0x0000
pressure (upscaled 7 -> 32 bit)

**Figure 61 Translate MIDI 1.0 Channel Pressure to MIDI 2.0**

### D.3.6 Pitch Bend

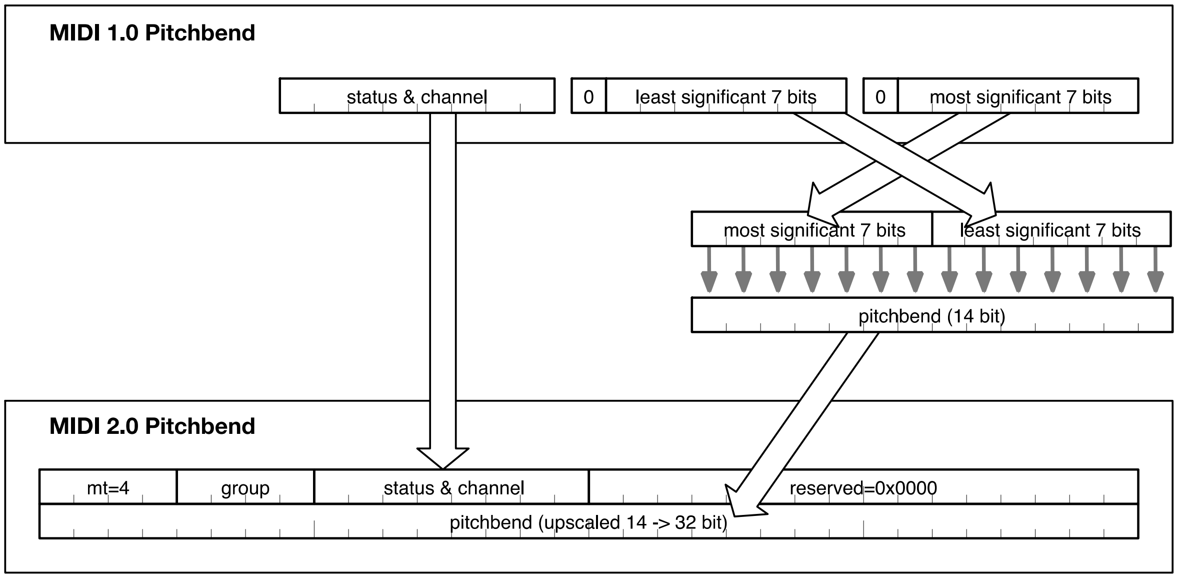


**Figure 62 Translate MIDI 1.0 Pitch Bend to MIDI 2.0**

***Note:*** *Pitch Bend values in the MIDI 1.0 Protocol are presented as Little Endian.*

### D.3.7 System Messages

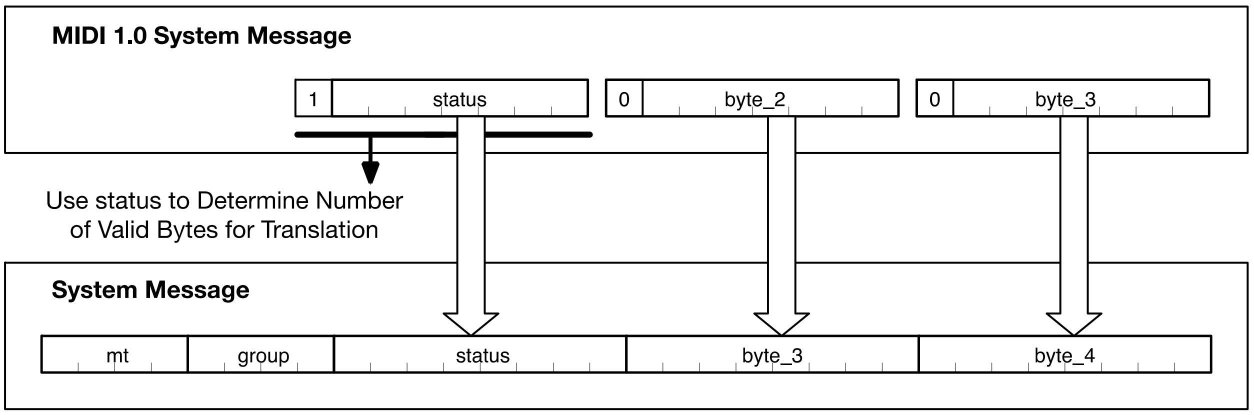
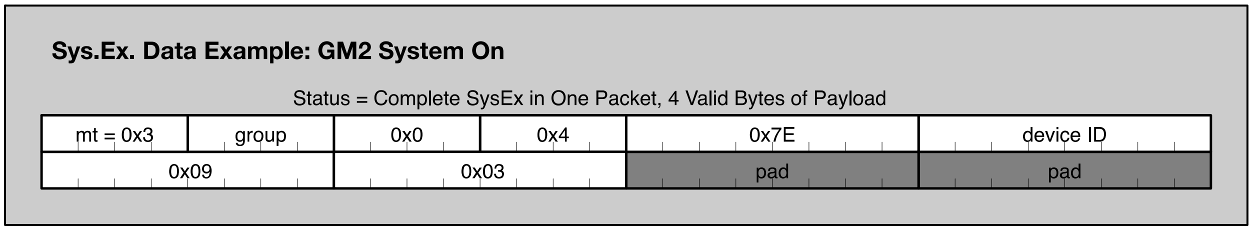


**Figure 63 Translate MIDI 1.0 System Message to MIDI 2.0**

#### 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 64***.


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 64 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 13 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

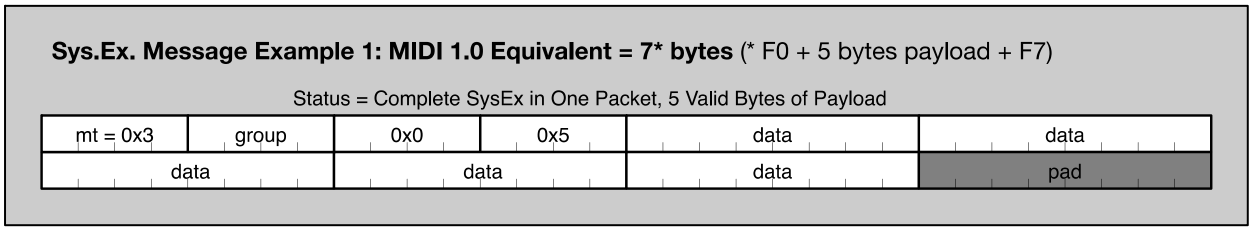

Sys.Ex. Message Example 1: MIDI 1.0 Equivalent = 7* bytes (* F0 + 5 bytes payload + F7)
Status = Complete SysEx in One Packet, 5 Valid Bytes of Payload
mt = 0x3
group
0x0
0x5
data
data
data
data
data
pad

**Figure 65 MIDI 2.0 System Exclusive Message Example 1**

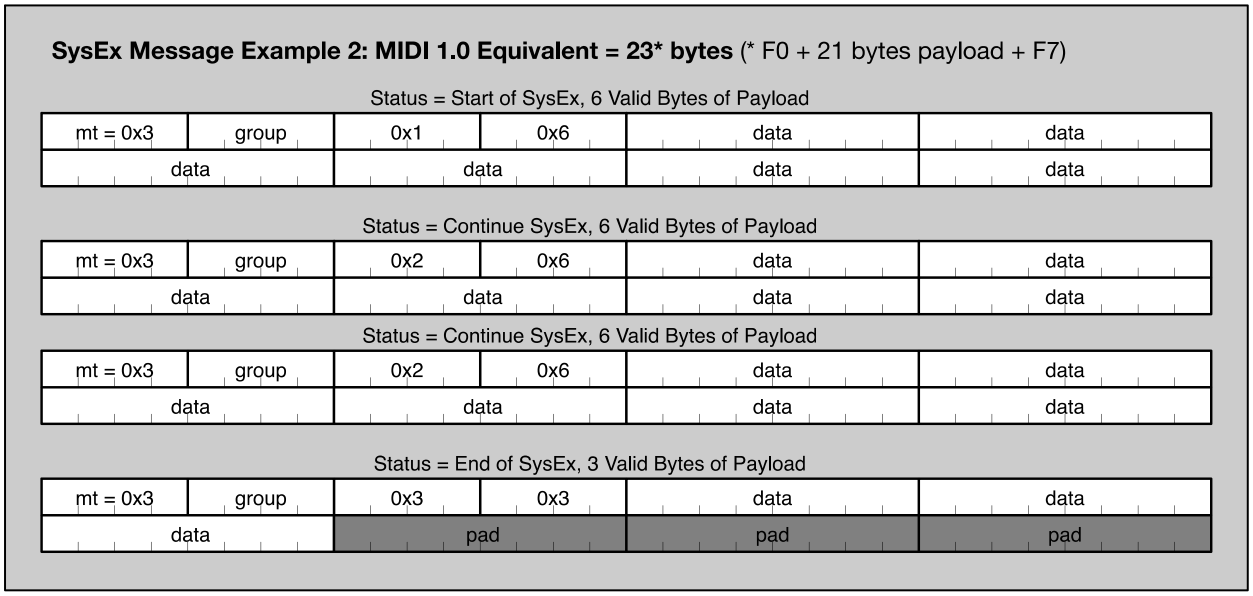
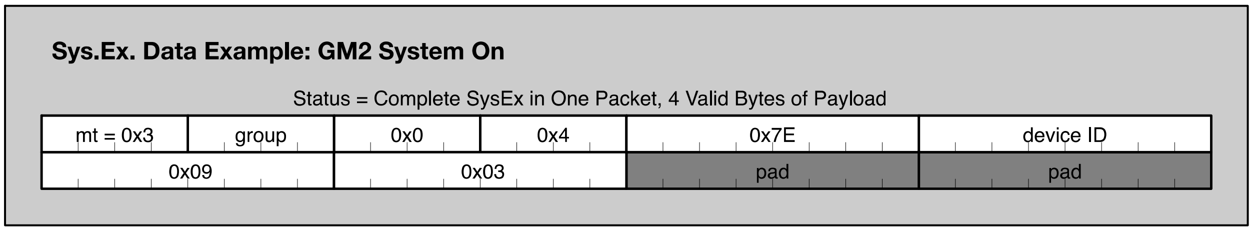

SysEx Message Example 2: MIDI 1.0 Equivalent = 23* bytes (* F0 + 21 bytes payload + F7)
Status = Start of SysEx, 6 Valid Bytes of Payload
mt = 0x3
group
0x1
0x6
data
Status = Continue SysEx, 6 Valid Bytes of Payload
0x2
0x6
Status = End of SysEx, 3 Valid Bytes of Payload
0x3
0x3
pad

**Figure 66 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 67 MIDI 2.0 System Exclusive Message Example 3**

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

**Table 14 UMPs for System Exclusive 8 (8-Bit) 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 15 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 | gggg | 0000 | 0000 00000000 00000000 | | |
| JR Clock | 0x0 | gggg | 0001 | reserv | ttttttt tttttttt | |
| JR Timestamp | 0x0 | gggg | 0010 | reserv | ttttttt tttttttt | |

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

**Table 16 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 17 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 18 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 |
| SysEx 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 |
| SysEx 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

| COLOR KEY: | Does not translate to the MIDI 1.0 Protocol | Reserved for future use by MMA/AMEI. Pad with zeros. |
|---|---|---|

**Table 19 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 Management | 0x4 | gggg | 1111nnn | rkkkkkkk | option flags | reserved | reserved | reserved | reserved |
| Control Change | 0x4 | gggg | 1011nnnn | rccccccc | reserved | DDDDDDDD | dddddddd | dddddddd | dddddddd |
| Registered Ctrl. (RPN) | 0x4 | gggg | 0010nnnn | rbbbbbbb | rppppppp | DDDDDDDD | dddddddd | dddddddd | dddddddd |
| Assignable Ctrl. (NRPN) | 0x4 | gggg | 0011nnnn | rbbbbbbb | rppppppp | DDDDDDDD | dddddddd | dddddddd | dddddddd |
| Relative Registered Ctrl | 0x4 | gggg | 0100nnnn | rbbbbbbb | rppppppp | DDDDDDDD | dddddddd | dddddddd | dddddddd |
| Relative Assignable Ctrl | 0x4 | gggg | 0101nnnn | rbbbbbbb | rppppppp | DDDDDDDD | dddddddd | dddddddd | dddddddd |
| Program Change | 0x4 | gggg | 1100nnnn | reserved | option flags | rppppppp | reserved | rBBBBBBB | rbbbbbbb |
| Channel Pressure | 0x4 | gggg | 1101nnnn | reserved | reserved | DDDDDDDD | dddddddd | dddddddd | dddddddd |
| Pitch Bend | 0x4 | gggg | 1110nnnn | reserved | reserved | DDDDDDDD | dddddddd | dddddddd | dddddddd |
| Per-Note Pitch Bend | 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 20 16-Byte UMP Formats for Message Type 0x5: System Exclusive 8 and Mixed Data Set**

| Message | Byte 1 | | Byte 2 | | Byte 3 | Bytes 4−16 |
|---|---|---|---|---|---|---|
| | MT | GR | Status | Low 4 Bits | | |
| **DATA** | | | | | | |
| 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 | |

# Appendix G All Defined Messages

| 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 | 0xg | 0x0 | 24 bit 0x00 0000 | | | | | | | |
| UTILITY | JR Clock | 0x0 | 0xg | 0x1 | reserved | 16 bit 0xtttt | | | | | | |
| UTILITY | JR Timestamp | 0x0 | 0xg | 0x2 | reserved | 16 bit 0xtttt | | | | | | |
| 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 | |
| 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 | 7 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 | 7 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 | 0x10 | 0xn | 7 bit note# 0xkk | reserved | 32 bit data 0xdddd dddd | | | | |
| MIDI 2.0 CHANNEL VOICE | Control Change | 0x4 | 0xg | 0x11 | 0xn | 7 bit controller# 0xcc | reserved | 32 bit data 0xdddd dddd | | | | |
| MIDI 2.0 CHANNEL VOICE | Program Change | 0x4 | 0xg | 0x12 | 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 | 0x13 | 0xn | reserved | reserved | 32 bit data 0xdddd dddd | | | | |
| MIDI 2.0 CHANNEL VOICE | Pitch Bend | 0x4 | 0xg | 0x14 | 0xn | reserved | reserved | 32 bit data 0xdddd dddd | | | | |
| MIDI 2.0 CHANNEL VOICE | Per-Note Management | 0x4 | 0xg | 0x15 | 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 | | | | | | |

| 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 Overview of Extensions to MIDI

***Note:*** *The lists below are overviews and are not exhaustive.*

## H.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.

## H.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