

Common Rules for MIDI-CI Property Exchange

MIDI Association Document: M2-103-UM

Document Version 1.2
Draft Date 2024-11-12

Published 2024-11-20

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



PREFACE

MIDI Association Document M2-103-UM Common Rules for MIDI-CI Property Exchange

The MIDI Capability Inquiry (MIDI-CI) specification defines mechanisms and a set of Universal System Exclusive messages used for Property Exchange. However, it does not define all the rules for Properties or devices that implement Property Exchange. This document, the Common Rules for MIDI-CI Property Exchange, complements MIDI-CI by defining a set of design rules for all Property Exchange Resources and Properties.

© 2020-2024 Association of Musical Electronic Industry (AMEI) (Japan)

© 2020-2024 MIDI Manufacturers Association Incorporated (MMA) (Worldwide except Japan)

ALL RIGHTS RESERVED. NO PART OF THIS DOCUMENT MAY BE REPRODUCED OR TRANSMITTED IN ANY FORM OR BY ANY MEANS, ELECTRONIC OR MECHANICAL, INCLUDING INFORMATION STORAGE AND RETRIEVAL SYSTEMS, WITHOUT PERMISSION IN WRITING FROM THE MIDI MANUFACTURERS ASSOCIATION.



<http://www.amei.or.jp>



<https://www.midi.org>

Version History

Table 1 Version History

Publication Date	Version	Changes
2020-02-20	1.0	Initial release
2020-12-08	1.1	Fixed an error in field layout order in the message shown in PE Message Format. Moved example device from prior Appendix A to the introduction area of the document. Minor Editorial Improvements.
2024-11-12	1.2	Added Flow Control mechanisms Added simpleAction and selectorAction Roles Expanded text on cacheTime Property Update text to reflect released PE Resources Minor Editorial Improvements.

Contents

Version History	3
Contents	4
Figures	6
Tables	6
1 References and Document Terminology	9
1.1 References	9
1.1.1 Normative References.....	9
1.2 Terminology	10
1.2.1 Definitions	10
1.2.2 Reserved Words and Specification Conformance	13
2 Introduction	14
2.1 Executive Summary.....	14
2.2 Background.....	14
3 Property Exchange Overview	15
3.1 Property Exchange Core Concepts and Mechanisms	15
3.2 Steps to Use Property Exchange.....	16
3.3 Property Exchange Message Format	16
3.4 Content of Examples in This Specification	17
4 Example Minimal Implementation	18
5 Property Exchange: MIDI-CI SysEx Messages	21
5.1 Property Exchange Inquiries and Replies.....	21
5.2 Property Data Set in Multiple Chunks.....	22
5.3 Request ID.....	22
6 Property Data Formats	24
6.1 General Rules for JSON Data in the Property Data Field	24
6.1.1 7-Bit Encoding of JSON String Values	24
6.1.2 Example of Unicode Escape Sequence in a String Value.....	24
6.1.3 CommonMark.....	25
6.1.4 Manufacturer-specific Properties.....	25
6.1.5 Non-JSON Data	25
6.1.6 Compression and Encoding Property Data	25
6.1.7 Mcoded7: 8-Bit to 7-Bit encoding	26
6.2 Order of Processing for Compression and/or Encoding	26
6.2.1 Uncompressed and Unencoded JSON Property Data	26
6.2.2 Compressed and Encoded JSON Property Data	27
6.2.3 Encoded Non-JSON Property Data	27
6.2.4 Compressed and Encoded Non-JSON Property Data.....	27
7 Header Data	28
7.1 Header Format Restrictions	28
7.1.1 JSON Header Data Property Additional Rules	28
7.2 Common Header Properties in a Request.....	28
7.3 Common Header Properties in a Reply	29
7.4 Reporting Status and Errors: Reply Header.....	29
7.4.1 Table of status codes used in a Reply Message	29

7.4.2	Other Error Mechanisms	30
7.4.3	cacheTime Header Property	30
7.5	Extra Header Property for Using Property Data which is Not JSON Data	31
7.6	Header Properties to be Defined in the Future	31
8	Resources	32
8.1	Resources Data Formats	32
8.2	Using Resources and Property Data	32
8.3	MMA/AMEI Defined Resources	32
8.4	Custom, Manufacturer/Device-Specific Resources	33
8.5	Resource ID	33
8.6	List Resources: Retrieving an Array of Objects	33
8.6.1	Property Data in a Reply to a List Resource Inquiry	33
8.6.2	Pagination	33
8.7	Simple Property Resources	35
9	Resources and Resource Data in a Device	36
9.1	ResourceList – Used Before Other PE Inquiries	36
9.2	Foundational Resources Defined in other Specifications:	36
9.3	Other Resources Defined in other Specifications:	36
10	Full and Partial SET Inquiries	37
10.1	Full SET Example:	37
10.2	Partial SET Examples:	38
11	Subscribing to Property Data	40
11.1	Extra Request Header Properties for Subscriptions	40
11.1.1	Selecting "partial", "full", or "notify"	42
11.2	Replying to Subscription Messages	42
11.3	Setting the Property Data when Sending the "command" Property with the Property Value "full"	42
11.3.1	Example: Subscription to a Simple Property Resource	42
11.4	Example Set of Transactions with Initiator Subscribing to Responder Resources	43
11.5	Subscription End	46
12	Terminating Transactions and Transaction Timeouts	47
12.1	Terminate Transaction	47
12.1.1	MIDI-CI NAK: Responder Terminates Transaction	47
12.1.2	MIDI-CI NAK: Initiator Terminates Transaction	47
12.1.3	Notify: Terminate Inquiry	47
12.2	Timeout	48
12.2.1	MIDI-CI ACK: Timeout Wait	48
12.2.2	Notify: Timeout Wait (deprecated)	49
12.2.3	MIDI-CI NAK: Timeout Has Occurred	49
12.2.4	Notify: Timeout Has Occurred	49
13	Resource Link Properties	51
13.1	Using a "role" Property of a Link	52
13.1.1	Role: simpleAction	53
13.1.1.1	Example Copy and Paste in ProgramList	54
13.1.2	Role: selectorAction	55

13.1.2.1 Example Selection of an Audio output on the ChannelList Resource from a Manufacturer defined List Resource.....	56
Step 1: Initiator Gets a ChannelList.....	57
Step 2: Initiator Displays to the User.....	58
Step 3: User Selects a selectorAction Button.....	58
Step 4: Initiator Displays to the User.....	59
Step 5: User Selects an Audio Output.....	59
13.2 Subscriptions and Linked Resources.....	59
14 Resource: ResourceList.....	60
14.1 Request ResourceList using Inquiry: Get Property Data.....	60
14.2 Property Data for ResourceList Returned via a Reply to Get Property Data Message.....	60
14.2.1 Additional Properties for List Resources.....	61
14.3 Minimized Listing of AMEI/MMA Defined Resources in ResourceList using Default Values.....	62
14.3.1 Example: ResourceList Object for the ChannelList Resource.....	62
14.4 Schema Property for Manufacturer Specific JSON Resources.....	63
14.4.1 Including a Schema Property in the ResourceList.....	63
14.4.2 Providing a Reference to an Expanded JSON Schema in the ResourceList.....	64
14.5 Using Columns for List Resources.....	65
14.6 ResourceList properties for non-JSON Resources.....	66
14.7 Extended Example of Property Data for ResourceList.....	67
14.8 JSON Schema for ResourceList.....	68
15 Flow Control.....	69
15.1 Flow Control MIDI-CI ACK Message.....	69
15.2 Using Flow Control with Inquiry: Get Property Data.....	70
15.3 Using Flow Control with Inquiry: Set Property Data.....	71
15.4 Sending a Retransmit MIDI-CI NAK Message.....	72
15.5 Timeouts when using Flow Control.....	73

Figures

Figure 1 Computer Connected to a Pedal Supporting Property Exchange.....	18
Figure 2 Example Using Request ID for Several Inquiries.....	23
Figure 3 Example of Initiator displaying simpleAction Roles.....	55
Figure 4 Example of Initiator displaying selectAction Roles.....	58
Figure 5 Example of Initiator displaying selectAction Roles.....	59
Figure 6 Flow Diagram of using Flow Control with an Inquiry: Get Property Data.....	71
Figure 7 Flow Diagram of using Flow Control with an Inquiry: Set Property Data.....	72

Tables

Table 1 Version History.....	3
Table 2 Words Relating to Specification Conformance.....	13
Table 3 Words Not Relating to Specification Conformance.....	13
Table 4 PE Message Format.....	16

Table 5 Initiator Sends Inquiry: Get Property Data.....18
Table 6 Responder Sends Reply to Get Property Data19
Table 7 Initiator Sends Inquiry: Get Property Data19
Table 8 Responder Sends Reply to Get Property Data19
Table 9 Initiator Sends Inquiry: Get Property Data19
Table 10 Responder Sends Reply to Get Property Data19
Table 11 All MIDI-CI Messages used for Property Exchange21
Table 12 Encoding and Compression Types List.....26
Table 13 Properties used in a Request Header28
Table 14 Properties used in a Reply Header29
Table 15 Reply Status Codes29
Table 16 Extra Header Properties for non-JSON Data31
Table 17 Initiator Sends Inquiry: Get Property Data32
Table 18 Responder Sends Reply to Get Property Data32
Table 19 Header Properties used with Pagination33
Table 20 Reply Header Properties used with Pagination34
Table 21 Initiator Sends Inquiry: Get Property Data34
Table 22 Responder Sends Reply to Get Property Data34
Table 23 Initiator Sends Inquiry: Get Property Data35
Table 24 Responder Sends Reply to Get Property Data35
Table 25 Initiator Sends Inquiry: Get Property Data35
Table 26 Responder Sends Reply to Get Property Data35
Table 27 Initiator Sends Inquiry: Get Property Data37
Table 28 Responder Sends Reply to Get Property Data37
Table 29 Initiator Sends Inquiry: Set Property Data Message37
Table 30 Responder Sends Reply to Set Property Data Message.....38
Table 31 Initiator Sends Inquiry: Get Property Data38
Table 32 Responder Sends Reply to Get Property Data38
Table 33 Initiator Sends Inquiry: Set Property Data Message38
Table 34 Responder Sends Reply to Set Property Data Message.....38
Table 35 Initiator Sends Inquiry: Set Property Data Message39
Table 36 Responder Sends Reply to Set Property Data Message.....39
Table 37 Initiator Sends Inquiry: Get Property Data39
Table 38 Responder Sends Reply to Get Property Data39
Table 39 Property Fields used for Subscription40
Table 40 Step 1: Initiator Sends a Subscription Message42
Table 41 Responder Sends Reply toSubscription Message.....42
Table 42 Step 2: Responder Sends Subscription Message using a "full" command42
Table 43 Initiator Sends Reply to Subscription Message.....43
Table 44 Initiator Sends Inquiry: Get Property Data43
Table 45 Responder Sends Reply to Get Property Data43
Table 46 Initiator Sends a Subscription Message43
Table 47 Responder Sends Reply toSubscription Message.....43
Table 48 Responder Sends Subscription Message using a "full" command44
Table 49 Initiator Sends Reply to Subscription Message.....44
Table 50 Responder Sends Subscription Message using a "full" command.....44
Table 51 Initiator Sends Reply to Subscription Message.....44
Table 52 Initiator Sends Inquiry: Get Property Data44
Table 53 Responder Sends Reply to Get Property Data44
Table 54 Responder Sends Subscription Message using a "full" command.....45

Table 55 Initiator Sends Reply to Subscription Message.....45

Table 56 Responder Sends Subscription Message using a "full" command.....45

Table 57 Initiator Sends Reply to Subscription Message.....45

Table 58 Initiator Subscription Message45

Table 59 Responder Sends Reply to Get Property Data45

Table 60 Initiator or Responder Sends Notify Message47

Table 61 MIDI-CI ACK Message used for Flow Control.....48

Table 62 Responder Sends Notify Message49

Table 63 Initiator Sends Notify Message49

Table 64 Link Properties51

Table 65 Example of Using Links With a "resId" Property51

Table 66 Initiator Sends an Inquiry: Get Property Data Message52

Table 67 Example of Using Links Without a "resId" Property52

Table 68 Link object Properties for use with a "simpleAction" Role53

Table 69 Example of ProgramList Resource using “simpleAction” Property.....54

Table 70 Initiator Sends an Inquiry: Set Property Data Message55

Table 71 Initiator Sends Reply to Inquiry: Set Property Data Message55

Table 72 Link object Properties for use with a "selectorAction" Role56

Table 73 Example of ChannelList Resource using “selectAction” Property57

Table 74 Initiator Sends an Inquiry: Get Property Data Message58

Table 75 Responder Sends Reply to Get Property Data58

Table 76 Initiator Sends an Inquiry: Set Property Data Message59

Table 77 Initiator Sends Reply to Subscription Message.....59

Table 78 Initiator Sends an Inquiry: Get Property Data Message60

Table 79 Link Properties60

Table 80 ResourceList Properties for List Resource.....61

Table 81 Example Transaction for ResourceList: Initiator Sends an Inquiry: Get Property Data Message.....61

Table 82 Responder Sends Reply to Get Property Data61

Table 83 declaring all the default values62

Table 84 Minimized object using all the default values63

Table 85 Minimized object using the default values for some Properties and declaring non-default values for other Properties.....63

Table 86 Example Resource List Object with a "schema" Property.....63

Table 87 Example ResourceList Object using a JSON Schema reference64

Table 88 Example of a JSON Schema reference: Initiator Sends an Inquiry: Get Property Data Message.....64

Table 89 Responder Sends Reply to Get Property Data64

Table 90 Example ResourceList Object Record Containing a "columns" Property65

Table 91 ChannelList Result for List Object Above65

Table 92 Example Non-JSON Resource66

Table 93 Example Non-JSON Resource67

Table 94 MIDI-CI ACK Message used for Flow Control.....69

Table 95 Example of using Flow Control an Inquiry: Get Property Data Message.....70

Table 96 Example of using Flow Control an Inquiry: Set Property Data Message71

Table 97 Retransmit MIDI-CI NAK Message72

1 References and Document Terminology

1.1 References

1.1.1 Normative References

- [COMM01] *CommonMark Spec*, Version 0.28, <https://spec.commonmark.org/0.28/>
- [ECMA01] *The JSON Data Interchange Syntax*, ECMA-404, <https://www.ecma-international.org/publications/standards/Ecma-404.htm>
- [JSON01] *JSON Schema*, Draft 4, 6, or 7 and subsequent revisions, <https://json-schema.org/>
- [MA01] *Complete MIDI 1.0 Detailed Specification*, Document Version 96.1, Third Edition, Association of Musical Electronics Industry, <http://www.amei.or.jp/>, and The MIDI Association, <https://www.midi.org/>
- [MA02] *M2-100-U MIDI 2.0 Specification Overview*, Version 1.1, Association of Musical Electronics Industry, <http://www.amei.or.jp/>, and The MIDI Association, <https://www.midi.org/>
- [MA03] *M2-101-UM MIDI Capability Inquiry (MIDI-CI)*, Version 1.2, Association of Musical Electronics Industry, <http://www.amei.or.jp/>, and The MIDI Association, <https://www.midi.org/>
- [MA04] *M2-102-U Common Rules for MIDI-CI Profiles*, Version 1.1, Association of Musical Electronics Industry, <http://www.amei.or.jp/>, and The MIDI Association, <https://www.midi.org/>
- [MA05] *M2-103-UM Common Rules for Property Exchange*, Version 1.1, Association of Musical Electronics Industry, <http://www.amei.or.jp/>, and The MIDI Association, <https://www.midi.org/>
- [MA06] *M2-104-UM Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol*, Version 1.1, Association of Musical Electronics Industry, <http://www.amei.or.jp/>, and The MIDI Association, <https://www.midi.org/>
- [MA07] *M2-105-UM*, MIDI-CI Property Exchange Foundational Resources, Version 1.1, Association of Musical Electronics Industry, <http://www.amei.or.jp/>, and The MIDI Association, <https://www.midi.org/>
- [RFC1950] *ZLIB Compressed Data Format Specification*, RFC1950, Version 3.3, <publisher>, <https://tools.ietf.org/html/rfc1950>
- [RFC6838] *Media Type Specifications and Registration Procedures*, RFC6838, <Version>, <publisher>, <https://tools.ietf.org/html/rfc6838>
- [RFC6901] *JavaScript Object Notation (JSON) Pointer*, RFC6901, <Version>, <publisher>, <https://tools.ietf.org/html/rfc6901>

1.2 Terminology

1.2.1 Definitions

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

Category: A collection of MIDI-CI messages grouped together because they function together or address similar types of mechanisms.

Chunk: A single System Exclusive message that is one segment of a complete Property Exchange message which spans multiple System Exclusive messages.

Controller Message: Any MIDI Message from the following list:

MIDI 1.0 and MIDI 2.0 Protocol:

- Control Change
- Channel Pressure (Aftertouch)
- Poly Pressure (Key Aftertouch)
- Registered Controller (RPN)
- Assignable Controller (NRPN)
- Pitch Bend

MIDI 2.0 Protocol only:

- Per-note Registered Controller (including Relative versions)
- Per Note Assignable Controller (including Relative versions)
- Per Note Pitch Bend

Data Set: A complete Property Exchange message whether sent in one System Exclusive message in single Chunk or in multiple Chunks.

Destination: A Receiver to which the Sender intends to send MIDI messages.

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

Device ID: A one-byte field in Universal System Exclusive messages, as defined in the MIDI 1.0 Specification *[MA01]*, to indicate which device in the system is supposed to respond. The more specific application of Device ID in MIDI-CI messages is defined in the MIDI Capability Inquiry *[MA03]* specification. The use of “Device” in this context is not the same as a Device as defined above.

Foundational Resource: A Resource which provides core Properties of a Device which are critical or highly valuable to properly implement numerous other Resources.

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

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

Initiator: One of two MIDI-CI Devices with a bidirectional communication between them. Initiator has the management role of setting and negotiating parameters for interoperability between the two Devices. The primary goal of Initiator is usually (but not strictly required to be) configuring two Devices for subsequent communication from Initiator as MIDI transmitter to Responder as MIDI receiver. The role of Initiator and Responder may alternate between the two MIDI-CI Devices. Either MIDI-CI Device may initiate a MIDI Transaction (act as Initiator) at any time. Also see Responder.

Inquiry: A message sent by an Initiator to begin a Transaction.

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

List Resource: A specific type of Resource that provides a list of objects in a JSON array.

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

Mcoded7: Data transferred over System Exclusive must be restricted to 7-bit bytes. Mcoded7 is a method of converting 8-bit data into 7-bit data. This data encoding format was first defined (although without a name) for use with Sample Dump in the MIDI 1.0 Specification. It is also defined in the M2-103-UM Common Rules for MIDI-CI Property Exchange specification [\[MA05\]](#).

MIDI 1.0 Protocol: Version 1.0 of the MIDI Protocol as originally specified in [\[MA01\]](#) and extended by MA and AMEI with numerous additional MIDI message definitions and Recommended Practices. The native format for the MIDI 1.0 Protocol is a byte stream, but it has been adapted for many different transports. MIDI 1.0 messages can be carried in UMP packets. The UMP format for the MIDI 1.0 Protocol is defined in the M2-104-UM Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol specification [\[MA06\]](#).

MIDI 1.0 Specification: Complete MIDI 1.0 Detailed Specification, Document Version 96.1, Third Edition [\[MA01\]](#).

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

MIDI 2.0 Protocol: Version 2.0 of the MIDI Protocol. The native format for MIDI 2.0 Protocol messages is UMP as defined in M2-104-UM Universal MIDI Packet (UMP) Format and MIDI 2.0 Protocol specification [\[MA06\]](#).

MIDI-CI: MIDI Capability Inquiry [\[MA03\]](#), a specification published by The MIDI Association and AMEI.

MIDI-CI Device: A Device that has the ability to act as a Responder that replies to inquiries received from an Initiator. The ability to act as an Initiator is recommended but optional.

MIDI-CI Transaction: A Transaction using a set of MIDI-CI messages that includes an Inquiry sent by an Initiator and a reply to the Inquiry returned by the Responder. The Responder's reply to an Inquiry might be a single message that satisfies the Inquiry, a set of multiple messages that satisfy the Inquiry, or an error message. See also Transaction.

MIDI Endpoint: A Device which is an original source of MIDI messages or final consumer of MIDI messages. Supports only MIDI 1.0 or is switchable between MIDI 1.0 and MIDI 2.0 Protocol messages.

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

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

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

MMA: See MIDI Manufacturers Association.

MUID (MIDI Unique Identifier): A 28-bit random number generated by a Device used to uniquely identify the Device in MIDI-CI messages sent to or from that Device.

PE: Property Exchange.

Process Inquiry: A set of MIDI-CI Transactions by which one Device may discover the current state of supported MIDI Messages in another device.

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

Property: A JSON key-value pair used by Property Exchange, for example "channel": 1.

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

Property Key: The key in a JSON key-value pair used by Property Exchange.

Property Value: The value in a JSON key-value pair used by Property Exchange.

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

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

Resource: A defined collection of one or more PE Properties with an associated inquiry to access its Properties.

Responder: One of two MIDI-CI Devices with a bidirectional communication between them. The Responder is the Device that receives an Inquiry message from an Initiator Device as part of a MIDI-CI Transaction and acts based on negotiation messages managed by the Initiator Device. Also see Initiator.

Resource: A defined collection of one or more PE Properties with an associated inquiry to access its Properties.

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

Simple Property Resource: A specific type of PE Resource that defines only a single Property. The reply includes only a Property Value (without a Property Key). See [\[MA05\]](#).

Source: A Source is a Sender which originates or generates MIDI messages. A Source does not include a Sender which is retransmitting messages which originated in another MIDI Device.

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

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

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

UMP: Universal MIDI Packet, see [\[MA06\]](#).

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

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

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

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

1.2.2 Reserved Words and Specification Conformance

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

Table 2 Words Relating to Specification Conformance

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

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

Table 3 Words Not Relating to Specification Conformance

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

2 Introduction

2.1 Executive Summary

Property Exchange is used to Discover, Get, and Set many properties including but not limited to device configuration settings, a list of controllers and destinations, a list of programs with names and other metadata, manufacturer, model number, and version. This specification defines rules which are commonly applied to all Property Exchange Transactions. The rules also define specific details for using the Property Exchange messages which are defined in the MIDI Capability Inquiry (MIDI-CI) specification *[MA03]*.

Property Exchange can enable devices to auto map controllers, edit programs, change state and also provide visual editors to DAW's without any prior knowledge of the device and without specifically crafted software. This means that devices could work on Windows, macOS, Linux, iOS, Android, embedded operating systems, and web browsers and may provide tighter interactions with DAWs and hardware controllers.

Property Exchange provides generalized access to device properties that might otherwise only be accessible through custom applications. Custom applications may continue to provide a more unique experience. However, the generalized mechanisms of Property Exchange allow functionality between devices and applications that are not custom designed. Devices that use Property Exchange are more likely to continue to function through underlying system upgrades, with future devices unknown today, or on new platforms that appear in the marketplace.

2.2 Background

The MIDI-Capability Inquiry (MIDI-CI) specification (part of MIDI 2.0) defines an architecture that allows devices with bidirectional communication to agree to use MIDI 2.0 capabilities. These new capabilities extend MIDI 1.0 while carefully protecting backward compatibility. MIDI-CI features “fall back” mechanisms so that if a device does not support new features, MIDI continues to work as defined by MIDI 1.0.

Property Exchange is a set of MIDI-CI messages used to access a wide range of properties in MIDI devices. The exchange of properties takes place between a MIDI-CI Initiator and a MIDI-CI Responder.

This Common Rules for Property Exchange document provides a complement to the MIDI-CI specification by defining details of the Property Exchange mechanism and rules for the data payload in MIDI-CI Property Exchange messages. Further Property Exchange specifications define schemas and various data payloads that use the rules in MIDI-CI and this document to achieve specific tasks.

3 Property Exchange Overview

3.1 Property Exchange Core Concepts and Mechanisms

Inquiry and Reply

Property Exchange is a dialogue between two MIDI Devices using Inquiry messages and matching Reply messages. These two Devices are the MIDI-CI Initiator and the MIDI-CI Responder.

Start Up

1. Before two Devices can use Property Exchange, the Initiator enacts a Discovery Transaction with the Responder. See the MIDI-CI specification *[MA03]* for use of the Discovery Transaction.
2. The two Devices use an Inquiry: Property Exchange Capabilities message and reply to discover basic Property Exchange support capabilities of each of the two Devices.
3. An Initiator sends an Inquiry: Get Property Data message with a “**ResourceList**” request to discover which Property Exchange features a Responder supports.

Get and Set

The core functions of Property Exchange are Getting and Setting a wide range of properties or values of properties of an attached device using an Inquiry: Get Property Data message and an Inquiry: Set Property Data message. These inquiries are messages issued by the Initiator, with reply messages being returned by the Responder.

Subscriptions

The Initiator may subscribe to a Resource from the Responder (if the Responder reports that the specific Resource is subscribable). When the subscribed Resource is changed on the Responder, the Responder informs the Initiator by a Subscription message. The subscription mechanism can also be used to keep the understanding of a Resource synchronized between Initiator and Responder.

Resources

Inquiries use Resources that are defined by either the MMA/AMEI or the device manufacturer to describe the Properties that can be retrieved and updated by Property Exchange. Resource definitions include the intent, intelligence required to use them, and how they relate with other Resources. Device manufacturers may decide which Resources to implement. However, response to **ResourceList** (see Section 14) is mandatory for all devices that support Property Exchange.

Property Data

When devices Get, Set or Subscribe to a Resource, the Property Data for the Resource is the targeted data. The Resource defines the content of the Property Data.

Error Messages

Property Exchange defines a set of informative error messages that are returned in an associated Reply message if an error occurred.

Property Exchange also defines a Notify message to report general errors when it is not possible to send a more informative error message via a Reply message.

MIDI-CI also provides a NAK message that can be used with Property Exchange.

Data Formats – System Exclusive, JSON, and Other Data

Property Exchange is sent via MIDI-CI Universal System Exclusive messages. System Exclusive SubID#2 of the System Exclusive determines the broad function of the message. The payload of the message includes a Header

Data field and a Property Data Field. Header Data is JSON compliant, with defined restrictions. The Property Data may be by default JSON data with defined restrictions, or any other data format depending on the definition of the Resource being used.

Use MIDI Messages Whenever Possible

Property Exchange has a "MIDI messages first" approach. If a method of changing a setting can be accomplished by using a common MIDI message (such as a Program Change or Control Change message) then the MIDI message method shall be used.

3.2 Steps to Use Property Exchange

1. Perform a MIDI-CI Discovery Transaction (See MIDI-CI *[MA03]* for details), exchanging information to:
 - A. Get the MUID of both devices
 - B. Ensure both devices support Property Exchange
 - C. Get Manufacturer SysEx ID of both devices
 - D. Get Device Family of both devices
 - E. Get Device Family Model Number of both devices
 - F. Get Software Revision Level of both devices
 2. Perform an Inquiry: Property Exchange Capabilities Transaction to get fundamental details of Property Exchange Support
 3. Perform an Inquiry: Get Property Data Transaction with a **ResourceList** Resource to discover support for the desired Property Data.
 4. Recommended but optional: Perform an Inquiry: Get Property Data Transaction with a **DeviceInfo** Resource to get Property Data with fundamental information about the Responder.
 5. Recommended but optional: Perform an Inquiry: Get Property Data Transaction with a **ChannelList** Resource to get Property Data with fundamental information about the Responder.
 6. Perform a Get, Set, or Subscribe Transaction with the desired Resource to use the associated Property Data.
- Repeat Step 6 whenever necessary to use other desired Resources (Steps 1 to 5 are only needed once if the Initiator can store the information discovered in those Steps).

3.3 Property Exchange Message Format

Property Exchange messages are in the following format:

Table 4 PE Message Format

Value	Parameter
F0	System Exclusive Start
7E	Universal System Exclusive
1 byte	Device ID: Source or Destination (depending on type of message): 7F: To/from Function Block
0D	Universal System Exclusive Sub-ID#1: MIDI-CI
1 byte	Universal System Exclusive Sub-ID#2: Category and Type of MIDI-CI Message: 0x32–3F: Property Exchange Messages

1 byte	MIDI-CI Message Version/Format
4 bytes	Source MUID (LSB first)
4 bytes	Destination MUID (LSB first)
1 byte	Request ID See MIDI-CI Specification [MA03] . Also see Section 5.3.
2 bytes	Number of Bytes for Header Data in this Chunk (nh)
nh bytes	Header Data, JSON compliant with defined restrictions. See Section 7.1
2 bytes	Number of Chunks in Data Set 0x0000 = Number of Chunks in Data Set is Unknown
2 bytes	Number of This Chunk (count starts from 0x0001) These fields track the separate but associated Inquiry and Reply Messages See MIDI-CI Specification [MA03] Also See Section 5.2
2 bytes	Number of Bytes for Property Data in this Chunk (nd)
nd bytes	Property Data: Resource determines the format of the data: JSON Compliant Data with defined restrictions Other Data Data compression is optional See Section 6.1
F7	End Universal System Exclusive

3.4 Content of Examples in This Specification

Examples throughout the remainder of this specification do not include all the fields of the message as shown above in Section 3.4. Most examples only show the contents of the Header Data Field and/or the contents of the Property Data Field.

Examples are designed to show implementation concepts and do not always use specification-defined content. Resources and Property Data shown may be fictitious or hypothetical examples.

Section 14 "Resource: ResourceList" does include a definition of a Resource and the associated Property Data. However, examples in this section may also include fictitious or hypothetical content to explain implementation concepts.

Refer to other AMEI/MMA specifications for defined Resources and Property Data.

4 Example Minimal Implementation

This section shows an example of implementing MIDI-CI Property Exchange in a very simple device. In this case the device is an effect pedal that can be controlled by MIDI. Software on the computer acts as the MIDI-CI Initiator and the effect pedal is the Responder.

Note: This is a hypothetical example to give a simplified overview of an implementation. The data as presented in this example should not be considered to be definitive for any Device.



Figure 1 Computer Connected to a Pedal Supporting Property Exchange

The effect pedal supports inquiries for 3 Resources:

- **ResourceList**
- **DeviceInfo**
- **AllCtrlList**

The effect pedal has a simple parser to recognize only these 3 inquiries. When it recognizes an inquiry, it replies with a preformed SysEx message fixed in ROM. The SysEx messages include the Property Data as described in the following tables:

Note: Tables in this example focus on the payload and omit other fields in the messages. See Section 3.4.

Action 1:

Perform a MIDI-CI Discovery Transaction, exchanging fundamental information between the software and the effect pedal (See MIDI-CI [\[MA03\]](#) for details).

Action 2:

The software sends an Inquiry: Property Exchange Capabilities message to request details of Property Exchange Support from the effect pedal. The effect pedal returns a Reply to Property Exchange Capabilities message.

Action 3:

Table 5 Initiator Sends Inquiry: Get Property Data

Header Data	{"resource": "ResourceList"}
Property Data	<i>none</i>

Table 6 Responder Sends Reply to Get Property Data

Header Data	<code>{"status":200}</code>
Property Data	<code>[{"resource": "DeviceInfo"}, {"resource": "AllCtrlList"}]</code>

Action 4:

Table 7 Initiator Sends Inquiry: Get Property Data

Header Data	<code>{"resource":"DeviceInfo"}</code>
Property Data	<i>none</i>

Table 8 Responder Sends Reply to Get Property Data

Header Data	<code>{"status":200}</code>
Property Data	<code>{ "manufacturerId": [125,0,0], "manufacturer": "Educational Use", "familyId": [0,0], "family": "Example Range", "modelId": [48,0], "model": "Example Pedal", "versionId": [0,0,1,0], "version": "1.0" }</code>

Action 5:

Table 9 Initiator Sends Inquiry: Get Property Data

Header Data	<code>{"resource":"AllCtrlList"}</code>
Property Data	<i>none</i>

Table 10 Responder Sends Reply to Get Property Data

Header Data	<code>{"status":200}</code>
Property Data	<code>[{ "title": "Level", "channel": 1, "ctrlType": "cc", "ctrlIndex": [75], "default": 4294967295 }]</code>

	<pre>}, { "title": "Tone", "channel": 1, "ctrlType": "cc", "ctrlIndex": [76], "default": 2147483648 }, { "title": "Distortion", "channel": 1, "ctrlType": "cc", "ctrlIndex": [77], "default": 2147483648 }]</pre>
--	--

5 Property Exchange: MIDI-CI SysEx Messages

5.1 Property Exchange Inquiries and Replies

MIDI-CI defines several different types of Inquiry messages differentiated by the Universal System Exclusive Sub Id #2.

Table 11 All MIDI-CI Messages used for Property Exchange

Sub ID #2	Message Type	Description
MIDI-CI Property Exchange Messages		
0x30	Inquiry: Property Exchange Capabilities	The Inquiry: Property Exchange Capabilities is used to request information about the Property Exchange Support.
0x31	Reply to Property Exchange Capabilities	The Reply to Property Exchange Capabilities reports the number of concurrent requests supported and the version of Property Exchange supported.
0x34	Inquiry: Get Property Data	The Inquiry: Get Property Data message is used to retrieve properties from the device.
0x35	Reply to Get Property Data	The results from the Inquiry: Get Property Data request.
0x36	Inquiry: Set Property Data	The Inquiry: Set Property Data message is used for the setting of information in a Resource.
0x37	Reply to Set Property Data	The results from the Inquiry: Set Property Data request.
0x38	Subscription	Subscriptions tell the Responder to please inform the Initiator or changes to the Resource Property Data.
0x39	Reply to Subscription	Reply to Subscription sends an acknowledgement that the Subscription Message has been received.
0x3F	Notify Message	Notify Messages informs an Initiator or Responder the status of a current Property Exchange Transaction. This has been superseded by MIDI-CI ACK and NAK messages.
MIDI-CI Management Messages		
0x7D	MIDI-CI ACK	MIDI-CI ACK informs an Initiator or Responder the status of a current Property Exchange Transaction.
0x7F	MIDI-CI NAK	MIDI-CI NAK informs an Initiator or Responder of an error for a current Property Exchange Transaction.

Note: See MIDI-CI specification for a complete list of all MIDI-CI messages

5.2 Property Data Set in Multiple Chunks

A Device may choose to send the Data Set of a Property Exchange as a single SysEx message or as a set of multiple SysEx messages known as “Chunks”.

Any message that contains Header Data only and does not contain any Property Data may not use the Chunking mechanism. Headers shall only be in the first chunk.

When a complete Property Exchange SysEx message, with its payload Resource Data, exceeds the size of the “Receivable Maximum SysEx Message Size” of the other Device (discovered in the initial Discovery Transaction between the devices) the sender shall break the message into multiple Chunks. A Device may also choose to send a message in multiple Chunks for its own design requirements.

If the Device chooses to send a Data Set in multiple Chunks, it specifies the “Number of Chunks in Data Set” and labels each Chunk with a sequential “Number of This Chunk”.

An additional Flow Control mechanism is defined in Section 15 to help reduce loss of chunks in Resources that have larger payloads.

See the MIDI-CI specification *[MA03]* for more details including values for special cases.

5.3 Request ID

The Request ID is a number from 0 to 127 and has 3 functions:

1. The Request ID is used to associate multiple Chunks of a single PE message. Every Chunk of a message shall contain the same Request ID.
2. The Request ID is used to associate a reply to the inquiry that prompted a response. The reply to an inquiry message shall contain the Request ID that was sent in the inquiry.
3. The Request ID also allows the Device to support multiple messages being sent and received at one time. This can be useful to prevent a larger PE message which is split over many chunks from blocking smaller requests. For example, it may take some time for a Responder to gather and transfer a complete Data Set in response to an inquiry. In this case, an Initiator might want to make a 2nd inquiry before the response to the 1st inquiry has been completed.

Using the Inquiry Property Exchange Capabilities mechanism, each Device reports the

Number of Simultaneous Property Exchange Requests Supported. An Initiator or Responder shall not use more than the reported number of requests supported by the other device.

Some devices may have a Number of Simultaneous Property Exchange Requests Supported which declares a total limit shared among all connected Initiators. An Initiator needs to deal with the possibility that the number of Number of Simultaneous Property Exchange Requests Supported may be exhausted due to other requests from other Initiators. A Responder may reply with a "Too many requests" error (see Section 7.4.1) and the request will need to be retried.

If an Initiator has exhausted the Number of Simultaneous Property Exchange Requests Supported, it should not issue any new request until a previous Transaction has completed.

Request IDs may be reused after a Transaction is complete.

Request ID values are unique only to the connection between a specific Initiator and specific Responder, determined by the MUID of those 2 devices. The same Request ID value may be active on a connection between a different pair of MUIDs without incurring a collision.

Requests that don't receive a response will time out and the Request ID can be reused. See Section 12.2 for more details about timeouts.

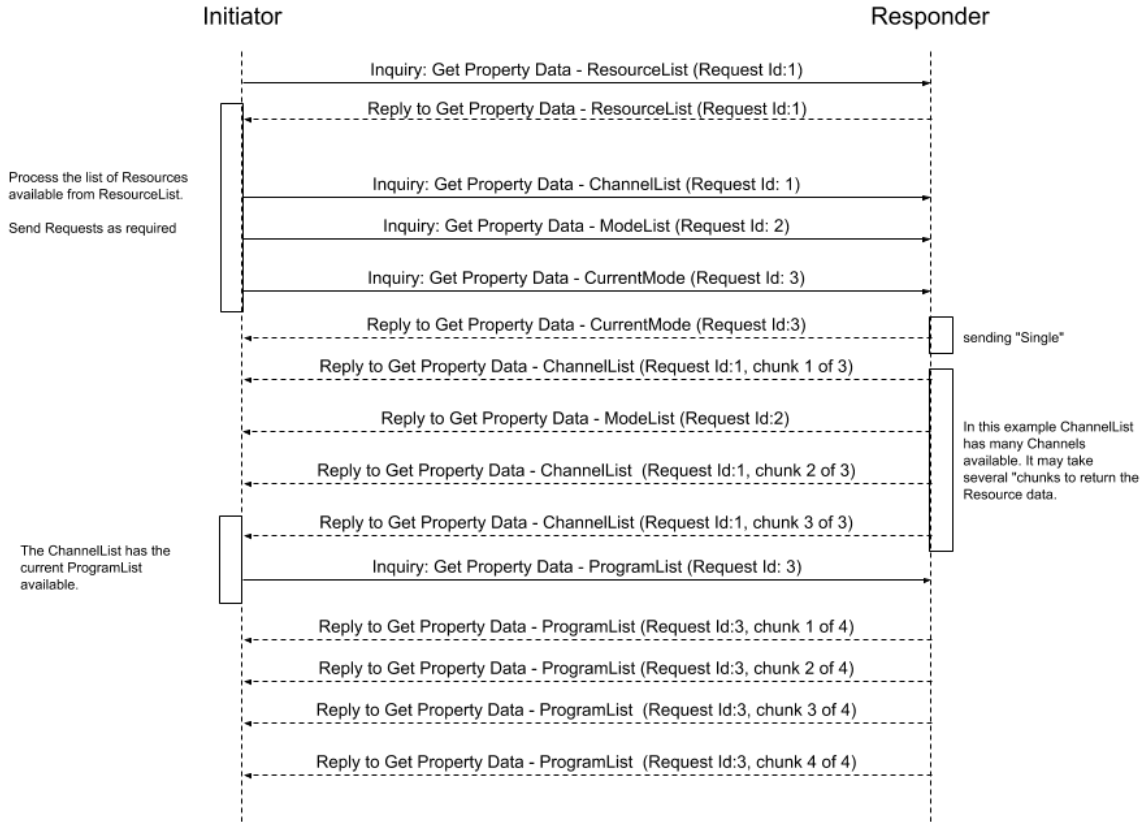


Figure 2 Example Using Request ID for Several Inquiries

See MIDI-CI [\[MA03\]](#) for more details.

6 Property Data Formats

Property Exchange SysEx messages have separate Header Data and Property Data fields.

By default, Property Data shall be JSON data.

However, if a Header Data field includes a "mediaType" Property, the associated Property Data may be non-JSON (see Section 6.2).

By default, the Property Data field is uncompressed. See Section 6.2.2 for details of optional compression and or encoding of the Property Data field.

6.1 General Rules for JSON Data in the Property Data Field

- Property names shall use camelCase ASCII strings ("a-z", "A-Z", "0-9" only).
- Property Names shall not be just numbers e.g. "0" or "120".
- Property Names should be kept short while still being meaningful.
- Property Names are case sensitive.
- All Numbers are in Decimal (Base 10). The JSON standard does not support hexadecimal number values such as 0x7F, as commonly used in MIDI documentation. These shall be converted to numbers (integers). e.g. Hex number 0x42 (or 42H) shall be converted to 66.

6.1.1 7-Bit Encoding of JSON String Values

Due to the restrictions of MIDI 1.0 SysEx, JSON string values have the following rules:

7-bit ASCII characters are supported, except that the "\" character has a special function as defined in the JSON standard, ECMA404 [ECMA01].

For clarity in Property Exchange, as ECMA404 may be unclear about this, the forward slash character "/" can be optionally escaped so it can be encoded in Property Exchange JSON as either "/" or "\/".

All characters which are not 7-bit ASCII characters shall be converted to UTF-16, and then escaped using "u" as defined in the JSON standard, ECMA404 [ECMA01].

Unicode Characters Examples:

- Beats♪ is converted to Beats\u266a.
- ノーミュージック、ノーライフ is converted to \u30ce\u30fc\u30df\u30e5\u30fc\u30b8\u30c3\u30af\u3001\u30ce\u30fc\u30e9\u30a4\u30d5
- 音楽がなければ人生じゃない。 is converted to \u97f3\u697d\u304c\u306a\u3051\u308c\u3070\u4eba\u751f\u3058\u3083\u306a\u3044\u3002

6.1.2 Example of Unicode Escape Sequence in a String Value

This example cannot be sent over SysEx:

```
{
  "title": "ピアノと弦",
  "channel": 1,
  "bankPC": [0,0,76],
  "description": "This text contains double quote \" and new line \n characters."
}
```

The non-ASCII characters, ピアノと弦, shall be escaped before sending.

An example after encoding the non-7-bit characters, allowing for transmission over SysEx:

```
{
  "title": "\u30d4\u30a2\u30ce\u3068\u5f26",
  "channel": 1,
}
```



```

    "bankPC": [0,0,76],
    "description": "This text contains double quote \" and new line \n characters."
  }

```

6.1.3 CommonMark

CommonMark *[COMM01]*, a strongly defined, highly compatible specification of Markdown, is a method for using plain text to display structured formatted text. It allows for simple screen devices to display the raw text while more complex devices can render the text in a formatted way. Some string fields, such as "description" in the **ModelList** Resource, will refer to supporting CommonMark.

6.1.4 Manufacturer-specific Properties

Manufacturers may wish to include their own specific information inside MMA/AMEI defined Resources (see Section 8).

This shall be accomplished by including a manufacturer-specific Property with a name prefixed with "x-". Manufacturer-specific properties shall conform to all other format rules defined by Property Exchange specifications.

Note: Manufacturer-specific properties that are contained within a Manufacturer/Device-Specific Resource do not need to be prefixed with "x-" (See Section 8.4).

Example of Property Data for a **DeviceInfo** Resource Inquiry including a Manufacturer Specific Property:

```

{
  "manufacturerId": [125,0,0],
  "manufacturer": "Educational Use",
  "familyId": [0,0],
  "family": "Example Range",
  "modelId": [48,0],
  "model": "Example Pedal",
  "versionId": [0,0,1,0],
  "version": "1.0",
  "x-uniqueKey": "myuniquevalue"
}

```

6.1.5 Non-JSON Data

Property Exchange allows devices to exchange Property Data that is not JSON compliant. If the Property Data is not JSON as defined in Section 4.1, the associated Header Field shall include a "mediaType" Property to declare the format of the Property Data (see Section 7.5).

Non-JSON Property Data shall be encoded to fit in the 7-bit data format of System Exclusive messages. Non-JSON Property Data may also be compressed. See Section 6.2.3 for details of Compression and Encoding.

6.1.6 Compression and Encoding Property Data

Property Data may be JSON data (by default) or may be another data type (with a "mediaType" declared, see Section 7.5). Regardless of data type, the Property Data field may optionally use compression and/or encoding if both devices support that compression and/or encoding. The compression and/or encoding types supported by a device shall be discovered using the **ResourceList** inquiry (See Section 9.1).

When the Property Data field contains a compressed and/or encoded payload, the Header Data field shall declare the compression and/or encoding type.

The following table lists the supported compression and encoding formats in this version of Property Exchange. The Property Value is the enum value used in Header fields (See Section 7) and in **ResourceList** (See Section 9.1) to declare encoding types.

Table 12 Encoding and Compression Types List

Property Value	Description
ASCII	Uncompressed, unencoded (must be 7-bit data)
Mcoded7	Encoded using Mcoded7 (See Section 4.3.1)
zlib+Mcoded7	Compressed using zlib (RFC 1950 [EXT07]) with Mcoded7 encoding.

6.1.7 Mcoded7: 8-Bit to 7-Bit encoding

The default format of the Property Data field in Property Exchange message is JSON data in ASCII text as defined in Section 6.1.1. Property Data which is not ASCII text as defined in Section 6.1.1 shall use the Mcoded7 format to encode 8-bit data to 7-bit.

Mcoded7 is also used in the File Dump format (without the name “Mcoded7”).

Description from the File Dump format from the MIDI 1.0 Specification *[MA01]*:

Each group of seven stored bytes is transmitted as eight bytes. First, the sign bits of the seven bytes are sent, followed by the low-order 7 bits of each byte. (The reasoning is that this would make the auxiliary bytes appear in every 8th byte without exception, which would therefore be slightly easier for the receiver to decode.)

The seven bytes:

```
AAAAaaaa BBBBbbbb CCCccccc DDDddddd EEEEEeee FFFFffff GGGggggg
```

are sent as:

```
0ABCDEFG
```

```
0AAAAaaaa 0BBBBbbbb 0CCCccccc 0DDDddddd 0EEEEeee 0FFFffff 0GGGggggg
```

From a buffer to be encoded, complete groups of seven bytes are encoded into groups of eight bytes. If the buffer size is not a multiple of seven, there will be some number of bytes leftover after the groups of seven are encoded. This short group is transmitted similarly, with the sign bits occupying the most significant bits of the first transmitted byte. For example:

```
AAAAaaaa BBBBbbbb CCCccccc
```

are transmitted as:

```
0ABC0000 0AAAAaaaa 0BBBBbbbb 0CCCccccc
```

6.2 Order of Processing for Compression and/or Encoding

6.2.1 Uncompressed and Unencoded JSON Property Data

When the Property Data is uncompressed and not encoded in Mcoded7, it shall be formatted using the following steps:

1. Create the JSON object (as a string)
2. Escape all multi-byte and non-ASCII characters as defined in Section 6.1.1.
3. Break the output into individual chunks based on Receivable Maximum SysEx Message Size (See MIDI-CI specification on how to retrieve this value from the Discovery Transaction.)
4. Send each chunk via MIDI-CI Property Exchange message(s).

6.2.2 Compressed and Encoded JSON Property Data

When the Property Data is zlib compressed and encoded in Mcoded7 (Mcoded7 is mandatory for zlib compressed data), it shall be formatted using the following steps:

1. Create the JSON object (as a string)
2. Escape all multi-byte and non-ASCII characters as defined in Section 6.1.1.
3. Compress using zlib
4. Encode use Mcoded7
5. Break the output into individual chunks based on Receivable Maximum SysEx Message Size (See MIDI-CI specification on how to retrieve this value from the Discovery Transaction.)
6. Send each chunk via MIDI-CI Property Exchange message(s).

6.2.3 Encoded Non-JSON Property Data

When Property Data is encoded it shall be formatted using the following steps:

1. Create the Property Data
2. Encode using Mcoded7
3. Break the output into individual chunks based on Receivable Maximum SysEx Message Size (See MIDI-CI specification on how to retrieve this value from the Discovery Transaction.)
4. Send each chunk via a MIDI-CI Property Exchange message.

6.2.4 Compressed and Encoded Non-JSON Property Data

When the Property Data is zlib compressed and encoded in Mcoded7 (Mcoded7 is mandatory for zlib compressed data), it shall be formatted using the following steps:

1. Create the Property Data
2. Compress using zlib
3. Encode use Mcoded7
4. Break the output into individual chunks based on Receivable Maximum SysEx Message Size (See MIDI-CI specification on how to retrieve this value from the Discovery Transaction.)
5. Send each chunk via MIDI-CI Property Exchange message(s).

7 Header Data

7.1 Header Format Restrictions

The Header Data field shall contain properties in JSON format. Header Data shall conform to the rules defined for Property Data in Section 6.1 and shall also conform to the additional set of rules and restrictions in Section 7.1.1.

Note: Header Data shall be JSON data even if the associated Property Data is not JSON data.

7.1.1 JSON Header Data Property Additional Rules

Restrictions, in addition to the rules in Section 6.1.1, on the Header Data make it more easily readable by devices with limited memory and processing power.

- Properties shall be of type number, boolean, or string.
- Described string Properties shall define a max length.
- Property Key names shall use camelCase ASCII strings no longer than 20 chars.
- For requests, the first Property shall be the Resource. For replies, the first Property shall be status. For subscription messages, the first Property shall be the command. In most cases no other data is needed.
- The Header Data shall not include any whitespace characters such as space, tab, line feed, or newline. Header Data shall be all on one line.

Note: these limitations only apply to JSON used in the Header Data (and not to JSON data in the Property Data field).

7.2 Common Header Properties in a Request

Table 13 Properties used in a Request Header

Property Key	Property Value Type	Description
resource	string (required, max 36 chars ASCII Alphanumeric characters only)	This is the targeted Resource.
resId	string (max 36 chars, ASCII Alphanumeric or "_" characters only)	Resource ID, the identifier used to select the desired Payload Data entry. See Section 8.5.
mutualEncoding	enum	This is used to indicate the format of the Property Data in this MIDI-CI Transaction. In the case of an Inquiry: Get Property Data message, this is the requested encoding of the Property Data in the expected response, a Reply to Get Property Data message from the Responder. In the case of an Inquiry: Set Property Data message, this is the encoding of the Property Data in that Inquiry: Set Property Data message. The value shall be one of the standard types defined in Section 6.1.6). The encodings supported by the Responder for each Resource are discovered by the use of ResourceList (See Section 14).
flowControl	Boolean default: false	Indicates that an Inquiry: Get Property Data transaction uses Flow Control. This shall only be set if

		the Responder declares support for Flow Control for this Resource in the ResourceList. See Section 15
--	--	---

7.3 Common Header Properties in a Reply

Table 14 Properties used in a Reply Header

Property Key	Property Value Type	Description
status	number (required, integer)	This is the Status of the response. This is similar to HTTP Status codes. This Property shall use the status list as described in Section 7.4.1
message	string (max 512 bytes after escaping)	This is an optional explanatory text for the user to provide a hint to why an error status was received.
mutualEncoding	enum	This is used in a Reply to Get Property Data message to indicate the format of the Payload Data. The value shall be one of the standard types defined in Section 6.1.5). This shall match the mutualEncoding Header Property in the Inquiry: Get Property Data message.
cacheTime	number (integer) (>=0)	This is the number of seconds that this document should be cached. See Section 7.4.3

7.4 Reporting Status and Errors: Reply Header

Each MIDI-CI Property Exchange reply message shall include a status Property in the Header Data field. If the status is any value other than 200, an associated message should be included.

The associated message is optional but recommended. If you include the message, you should use it to specify why the status code was triggered, not what the status code means.

The message Header Data may be used to declare a successful Transaction. If a Responder successfully receives and responds to an Inquiry: Set Property Data message, the Responder shall confirm with a value of 200 in the "status" Property.

Example:

```
{"status":200}
```

The message Header Data may be used to report an error code with an associated message.

Example:

```
{"status":400,"message":"Inquiry Header does not specify the Resource."}
```

7.4.1 Table of status codes used in a Reply Message

The following is a list of Property Exchange status codes. Reply messages shall only use the Status codes that are listed in this table and shall not use any other codes.

Note: The Notify message uses other status codes. See Section 12 for details.

Table 15 Reply Status Codes

Value	Parameter
200-299 Success Messages – Do Not Retry	

200	Success/Ok
201	Accepted – Inquiry: Set Property Data message is accepted but device doesn't have the time or processing power to guarantee the results.
300-399 Redirection Messages – These Should Retry	
341	Resource Currently Unavailable or an Error Occurred.
342	Bad Data/Unexpected End of Data Set – Example: might be caused by SysEx data corruption or some other data stream issue.
343	Too Many Requests – Device is unable to support this additional request at this time.
400-499 Client Error Responses – These are Fails – Do Not Retry	
400	Bad Request – Data was received but it isn't correct
403	Request received but reply not available based on Authorization. Example: contains protected or copyrighted data.
404	Resource Not Supported/Found
405	Resource Not Allowed – Resource is not applicable at this time. Example: Device may be in the wrong Mode.
406	Flow Control not supported
407	Flow Control is required
413	Payload Too Large
415	Unsupported Media Type or Encoding
445	Invalid Version of Data – Data is assumed valid, but the version of the data format is a version that the receiver cannot support. Example: the data might be intended for a newer version of software than the receiver, running a prior version of software, is able to use.
500-599 Server Error Responses – These are Major Errors	
500	Internal Device Error – A significant failure has occurred on the device.

7.4.2 Other Error Mechanisms

In addition to the errors defined above for use in a Reply message, Property Exchange provides mechanisms to Terminate an Inquiry, perform a Timeout Wait, or report a Timeout condition by way of a Notify message. For details see Section 12.

MIDI-CI also provides a NAK message that may be used with Property Exchange when it is not possible to return a more informative error report as defined above. See the MIDI-CI specification for details.

7.4.3 cacheTime Header Property

This is the number of seconds that this Property Data should be cached by the Initiator. During this **cacheTime**, the Initiator should use the cached Property Data. Only when the Initiator needs to access the Property Data after the **cacheTime** has elapsed, it needs to request it again from the Responder to get the most up-to-date Property Data. This optional feature reduces the load on the Responder by allowing Initiators to reuse the data without making a second request.

If the Resource Data has also been subscribed to and there is a notification that the data has changed, then Subscription mechanisms override the **cacheTime**.

cacheTime is independent of each Property Exchange request Header.

When using pagination, a subset of Property Data is retrieved in one or more Property Exchange requests. Each request has its own **cacheTime**.

For example, retrieving the first 10 entries of a **ProgramList** Resource could return a **cacheTime** of 30 seconds, and the request for entries 11 to 20 could return a **cacheTime** of 60 seconds.

Implementations should implement **cacheTime** per request.

In the example above, entries 1..10 will expire 30 seconds after receiving the response, and entries 11..20 will expire 60 seconds after the second response.

Some implementations may manage only one **cacheTime** per Resource, not per individual request. In this case, the oldest **cacheTime** should be retained, even if partial Property Data is received with a renewed **cacheTime**. Consequently, if by the time new partial Property Data is received, the oldest **cacheTime** is already expired, all previously cached data should be removed and only the new partial Property Data with its **cacheTime** is valid for this Resource.

7.5 Extra Header Property for Using Property Data which is Not JSON Data

If the associated Property Data is not JSON data which conforms to rules in Section 4.1, the Header Field shall include a "**mediaType**" Property to declare the format of the Property Data.

Table 16 Extra Header Properties for non-JSON Data

Property Key	Property Value Type	Description
mediaType	string (max 75 chars)	Media Type of the Property Data that is the payload of this Property Exchange message in the format(s) as defined by RFC 6838 [RFC6838] . This is sometimes referred to as MIME type.

7.6 Header Properties to be Defined in the Future

Some Resources in the future may require new or unique Header Properties. All such Header Properties shall comply with the rules in this Section 7.

8 Resources

A Resource is what determines how a request and response is structured. It tells the device how to act and what to do.

Resource Format Rules:

- Resource names shall be nouns like DeviceInfo, Tempo, Mode
- Resource names shall not contain spaces and shall use Pascal Case (sometimes called “UpperCamelCase”) e.g. Channel list is "ChannelList" and tempo is "Tempo".

8.1 Resources Data Formats

By default, Resources shall be JSON compliant with the format rules and restrictions defined in Sections 4 and 5.

Resources may also contain other data which is not compliant with JSON only if the Resource declares a non-JSON data type in the "mediaType" Property of the Header Field (see Sections 6.1 and 7.5).

8.2 Using Resources and Property Data

An Initiator sends an inquiry message with the Resource it wishes to retrieve declared in the Header Data field. The Responder returns a reply message with the corresponding Property Data in the Property Data field.

Table 17 Initiator Sends Inquiry: Get Property Data

Header Data	{"resource":"DeviceInfo"}
Property Data	<i>none</i>

Table 18 Responder Sends Reply to Get Property Data

Header Data	{"status":200}
Property Data	{ "manufacturerId":[125,0,0], "manufacturer":"Educational Use", "familyId":[0,0], "family":"Example Range", "modelId":[48,0], "model":"Example Pedal", "versionId":[0,0,1,0], "version":"1.0" }

8.3 MMA/AMEI Defined Resources

MMA/AMEI Resources should provide standardized mechanisms for a unified approach to common applications.

Resources defined by the MMA/AMEI shall describe the Properties that can be retrieved, the intent of the Properties, implementation rules or guidelines, and how they relate with other Resources.

Resource specifications shall define which Properties are required and which Properties are optional. Properties which are required, including strings marked as required, shall not be left empty.

See Section 9.2 and 9.3.

8.4 Custom, Manufacturer/Device-Specific Resources

Manufacturers may design devices that use custom Resources. The data format of custom Resources shall conform to the standard style guides defined by Property Exchange.

All manufacturer/device-specific Resource names shall be prefixed with "X-". This is to avoid any overlap with future defined Resources.

For Example:

X-MyCustomResource

Note: Inside a Manufacturer/Device-Specific Resource (such as X-MyCustomResource), individual properties do not need to be prefixed with "x-" (See Section 4.1.3).

8.5 Resource ID

Resources may be defined to require a Resource ID ("resId") Property.

A Resource may have a selection of various entries, any one of which could be returned as the associated Property Data for that specific Resource. Such Resources shall have a "resId" Property with a unique Property Value assigned to each entry. Any inquiry for that Resource shall include a "resId" Property to select the desired entry.

If a Resource is defined to use the "resId" Property, then the Resource shall have the **requireResId** property set to true in the **ResourceList**. See Section 12.

8.6 List Resources: Retrieving an Array of Objects

A List Resource is a specific type of Resource that provides a list of objects in a JSON array. Typical List Resources include **ProgramList** and **ChannelList**.

The names of List Resources shall have the suffix "List", e.g. "**ProgramList**"

8.6.1 Property Data in a Reply to a List Resource Inquiry

- A Reply to a List Resource shall be an array of objects
- Objects should only be one level deep. Objects, other than the "link" Property described below, may contain arrays or collections of integral values or strings and shall not contain other objects.
- List Resources can be complemented with additional Resources to access further, more detailed information that is not provided in the list. Objects in the reply to a List Resources may use a "link" Property to declare each Resource that is available for accessing more detailed information about a listed object. See Section 13.

8.6.2 Pagination

Pagination allows the Initiator to get a subset of the total list of objects available. Pagination uses the offset and limit mechanism commonly used in JSON applications. Pagination support for each Resource shall be declared using the "canPaginate":true Property in the **ResourceList**. See Section 14.2.1.

The "offset" and "limit" Properties may be used in the Header Data of an Inquiry: Get Property Data that requests a List Resource. If pagination is used, both Properties shall be in the Header data of the inquiry:

Table 19 Header Properties used with Pagination

Property Key	Property Value Type	Description
offset	number (integer, >=0 only)	This is the (0-based) start position in the array of objects in the List Resource.
limit	number (integer, >=1 only)	This specifies the maximum number of objects to be returned.

All Responder replies to an Inquiry: Get Property Data message with a List Resource, when the Resource has declared "canPaginate" in the ResourceList, shall contain a "totalCount" Property:

Table 20 Reply Header Properties used with Pagination

Property Key	Property Value Type	Description
totalCount	number (integer, >=0 only)	This is the number of total objects available, regardless of pagination.

The first Inquiry: Get Property Data message should request a list of entries starting from an offset of 0. The Initiator may determine how to access the list of entries using pagination based on the "totalCount" Property from the Responder and the "limit" Property chosen by the Initiator.

Table 21 Initiator Sends Inquiry: Get Property Data

Header Data	{"resource":"ExampleList","offset":10,"limit":5}
Property Data	none

Table 22 Responder Sends Reply to Get Property Data

Header Data	{"status":200,"totalCount":128}
Property Data	[<pre> { "title":"Result 11", link:[{"resource":"X-Detail","resId":"detail11"}] }, { "title":"Result 12", link:[{"resource":"X-Detail","resId":"detail12"}] }, { "title":"Result 13", link:[{"resource":"X-Detail","resId":"detail13"}] }, { "title":"Result 14", link:[{"resource":"X-Detail","resId":"detail14"}] }, { "title":"Result 15", link:[{"resource":"X-Detail","resId":"detail15"}] }]</pre>

If "offset" and "limit" properties are not used, the Property Data shall contain the whole list of all objects.

8.7 Simple Property Resources

Simple Property Resources are defined to have Property Data which contains only a single string, number, or boolean.

Table 23 Initiator Sends Inquiry: Get Property Data

Header Data	{"resource":"LocalOn"}
Property Data	<i>none</i>

Table 24 Responder Sends Reply to Get Property Data

Header Data	{"status":200}
Property Data	true

Table 25 Initiator Sends Inquiry: Get Property Data

Header Data	{"resource":"CurrentMode"}
Property Data	<i>none</i>

Table 26 Responder Sends Reply to Get Property Data

Header Data	{"status":200}
Property Data	"performance"

9 Resources and Resource Data in a Device

Property Exchange specifications that are written to comply with this Common Rules for Property Exchange specification define common Resource Data for MIDI Devices. They also define the relationship between some of the Resource Data (or the relationship between Resources that access Resource Data). These Resource Data are flexible enough to describe everything from keyboards, drums, samplers through to robotics and lighting.

9.1 ResourceList – Used Before Other PE Inquiries

An Initiator requests the **ResourceList** Resource in an Inquiry: Get Property Data message to retrieve a list of all Resources that the Responder supports. This inquiry should be made before initiating any other Property Exchange inquiries.

All Devices that support Property Exchange shall respond correctly to a **ResourceList** inquiry to report Resources the Device supports.

ResourceList shall not be included in the response to a **ResourceList** inquiry.

For the full definition of **ResourceList**, see Section 14.

9.2 Foundational Resources Defined in other Specifications:

It is strongly recommended that all Devices should support the following Foundational Resources, defined in the "MIDI-CI Property Exchange Foundational Resources: DeviceInfo, ChannelList, JSONSchema" specification *[MA04]*. These Resources provide details about a device that will aid in auto configuration and provide useful background data to help optimize many other Property Exchange inquiries. Some PE inquiries have dependencies that require the prior use of these Resources.

- Device Information (**DeviceInfo** Resource) - This Resource Data provides fundamental information about a device. This contains data similar to the Device Inquiry Universal SysEx message. However, it also includes human-readable values for Manufacturer, Family, Model and Version information.
- Channels (**ChannelList** Resource) – The **ChannelList** Resource describes the current channels the device is transmitting and receiving across the whole Device. It describes for each Channel the Port of each Channel, and MPE status. It also describes for each receive Channel the current program and the **ProgramList** Resources available. In Devices that have more than one Mode, the current available channels can be significantly different in each Mode.

9.3 Other Resources Defined in other Specifications:

Other Resources are defined in other specifications published by AMEI/MMA. This Common Rules for Property Exchange specification defines the rules for all other Resource specifications.

10 Full and Partial SET Inquiries

Property Exchange offers two methods for updating Resource Data or subsets of Resource Data:

- Method 1, Full SET: To update a whole set of Resource Data, an Initiator shall send an Inquiry: Set Property Data message. The Property Data field shall contain the whole Resource Data which would be sent by a Responder in a Reply to Get Property Data message. This may be used only if the target Resource declares a Property Value of "partial" or "full" for the "canSet" Property in the **ResourceList** (see Section 14).
- Method 2, Partial SET: For incremental changes or to change a subset of the Resource Data, an Initiator shall send an Inquiry: Set Property Data message. The Header Data field shall include "setPartial":true. In the Property Data field, each key of the update Resource Data shall represent a JSON Pointer (as per RFC 6901) to the Resource Data and provides a value. This value shall be a string, number, or boolean. This value shall not be an object or array. This may be used only if the target Resource declares a Property Value of "partial" for the "canSet" Property in the **ResourceList** (see Section 14).

10.1 Full SET Example:

A hypothetical Manufacturer defined Resource called "X-ProgramEdit" retrieves the following Resource Data using an Inquiry: Get Property Data message:

Table 27 Initiator Sends Inquiry: Get Property Data

Header Data	{"resource":"X-ProgramEdit","resId":"abcd"}
Property Data	<i>none</i>

Table 28 Responder Sends Reply to Get Property Data

Header Data	{"status":200}
Property Data	{ "name": "PIANO 4", "lfoSpeed": 30, "lfoWaveform": "triangle", "pitchEnvelope": { "rates": [94,67,95,60], "levels": [50,50,50,50] } }

The Initiator decides to upload a whole new program. The Initiator uses an Inquiry: Set Property Data message:

Table 29 Initiator Sends Inquiry: Set Property Data Message

Header Data	{"resource":"X-ProgramEdit","resId":"abcd"}
Property Data	{ "name": "PIANO 4", "lfoSpeed": 30, "lfoWaveform": "triangle", "pitchEnvelope": { "rates": [94,67,95,60], "levels": [50,50,50,50] } }

	<pre> "levels": [50,50,50,50] } } </pre>
--	--

Table 30 Responder Sends Reply to Set Property Data Message

Header Data	{"status":200}
Property Data	<i>none</i>

10.2 Partial SET Examples:

A hypothetical Manufacturer defined Resource called "X-ProgramEdit" retrieves the following Resource Data using an Inquiry: Get Property Data message:

Table 31 Initiator Sends Inquiry: Get Property Data

Header Data	{"resource":"X-ProgramEdit","resId":"abcd"}
Property Data	<i>none</i>

Table 32 Responder Sends Reply to Get Property Data

Header Data	{"status":200}
Property Data	<pre> { "name": "PIANO 4", "lfoSpeed": 30, "lfoWaveform": "triangle", "pitchEnvelope": { "rates": [94,67,95,60], "levels": [50,50,50,50] } } </pre>

The Initiator makes a minor change to the LFO speed and only wants to send this change.

The Initiator uses an Inquiry: Set Property Data message:

Table 33 Initiator Sends Inquiry: Set Property Data Message

Header Data	{"resource":"X-ProgramEdit","resId":"abcd","setPartial":true}
Property Data	<pre> { "/lfoSpeed":10 } </pre>

Table 34 Responder Sends Reply to Set Property Data Message

Header Data	{"status":200}
-------------	----------------

Property Data	<i>none</i>
---------------	-------------

A Partial SET may send more than one change. This is an example that updates 2 array values:

Table 35 Initiator Sends Inquiry: Set Property Data Message

Header Data	<code>{"resource": "X-ProgramEdit", "resId": "abcd", "setPartial": true}</code>
Property Data	<code>{ "/pitchEnvelope/rates/0": 80, "/pitchEnvelope/levels/0": 60 }</code>

Table 36 Responder Sends Reply to Set Property Data Message

Header Data	<code>{"status": 200}</code>
Property Data	<i>none</i>

When these 2 updates are applied, getting the Property Data will now look like:

Table 37 Initiator Sends Inquiry: Get Property Data

Header Data	<code>{"resource": "X-ProgramEdit", "resId": "abcd"}</code>
Property Data	<i>none</i>

Table 38 Responder Sends Reply to Get Property Data

Header Data	<code>{"status": 200}</code>
Property Data	<code>{ "name": "PIANO 4", "lfoSpeed": 10, "lfoWaveform": "triangle", "pitchEnvelope": { "rates": [80, 67, 95, 60], "levels": [60, 50, 50, 50] } }</code>

11 Subscribing to Property Data

A Subscription mechanism in PE may be used when an Initiator wants to keep Property Data of a Responder's Resource synchronized between the Initiator and Responder. A subscription is only possible if it is supported by both Initiator and Responder.

If a Responder declares that a subscription is available, the Initiator may subscribe to the Resource while it needs ongoing, active access to the Property Data.

1. The Responder shall report in its reply to **ResourceList** that a Resource is subscribable. See Section 14.
2. The Initiator may subscribe to a subscribable Resource on the Responder using the Subscription message with a "start" command.

Subscriptions enable two-way communication of one set of Property Data:

Responder: If any Property in the subscribed Property Data changes in the Responder, then the Responder shall inform the Initiator using a Subscription message with a "partial", "full", or "notify" command (See Section 11).

Examples:

- If a MIDI Control Change message changes a Property Value in the Responder, the Responder sends the updated Property to the Initiator.
- If a user sets a device parameter that changes a Property Value in the Responder, the Responder sends the updated Property to the Initiator.
- If an Inquiry: Set Property Data message changes a Property Value in the Responder, the Responder sends the updated Property to the Initiator.

Initiator: If any Property in the subscribed Resource Data changes in the Initiator, the Initiator shall take one of two optional actions to update the Property in the Responder:

- A. If the Initiator is able to send a MIDI message (such as a MIDI Control Change) to set the Property on the Responder, then it shall send the MIDI message.
- B. If the Initiator is not able to send a MIDI message to set the Property on the Responder, it shall send an Inquiry: Set Property Data.

Example:

- A visual program editor is open in a DAW. Editing data on the device will update the DAW to reflect those changes without the user having to manually refresh the data. This also works in reverse where the DAW can send very specific changes back to the device.

Each subscription update from the Responder shall be sent using a Subscription message.

Each reply to a Subscription message shall be sent using a Reply to Subscription message.

Note: If a subscribed Resource has linked Resources (see Section 13), in some cases the Initiator may subscribe to the linked Resources in order to get additional, related updates.

Note: There may be a delay between the Initiator sending the Inquiry: Set Property Data message and receiving a corresponding Subscription update reflecting the change from the Responder. When displaying the state of a Property to a user, a device should not display the duplication of data.

11.1 Extra Request Header Properties for Subscriptions

Table 39 Property Fields used for Subscription

Property Key	Property Value Type	Description
--------------	---------------------	-------------

<p>subscribeId</p>	<p>string (required, max 8 chars, "a-z", "0-9" or "_ " characters only)</p>	<p>An initiator may decide to subscribe to several different Resource Data at any one time. To identify which messages are related to each Subscription a Subscription Id shall be assigned by the Responder when starting a Subscription.</p> <p>This Subscription Id shall be used by the Responder for all update messages related to that Subscription and shall also be used by either the Initiator or the Responder to end the subscription. Once a Subscription is ended, the Subscription Id can be reused for a new Subscription.</p>
<p>command</p>	<p>enum (required)</p>	<p>The command determines the intent of the Subscription MIDI-CI message</p> <p>Start Subscription ("start") - Initiator only This shall create a new Subscription. The header is identical to that used by an Inquiry: Get Property Data message. The Response does not return any Property Data.</p> <p>Partial Property Data Update ("partial") - Responder only The Responder may use this to send an update whenever it makes changes to a subset of the subscribed Property Data. The Property Data for the update shall use the same format as data in a Partial Inquiry: Set Property Data message.</p> <p>Full Property Data Update ("full") - Responder only The Responder shall use this to send a complete set of the subscribed Property Data using the same format as in a Reply to Get Property.</p> <p>Notify Command: Get Property Data ("notify") - Responder only The Responder shall use this to ask the Initiator for a complete refresh of the subscribed Property Data. There is no body in this message.</p> <p>The Initiator shall not use this but instead shall perform an Inquiry: Get Property Data to get the refreshed Property Data from the Responder.</p> <p>End Subscription ("end") - Initiator or Responder Either device can end the Subscription. There is no Property Data in this message.</p>
<p>endedBy</p>	<p>enum</p>	<p>This Property disambiguates the End Command between the Initiator and Responder.</p>

		("initiator") - Initiator only The Initiator is ending the Subscription.
		("responder") - Responder only The Responder is ending the Subscription.

11.1.1 Selecting "partial", "full", or "notify"

When an individual Property or small number of Properties of the Property Data changes in the Responder, the Responder should use a "command": "partial" Property.

When the complete Property Data or a substantial subset of the Property Data changes in the Responder, the Responder should use a "command": "full" Property. The "command": "full" Property should be also used for all Simple Property Resources.

Use of "command": "partial" and "command": "full" should be limited to only when the Property Data fits into a single Property Exchange Message Chunk. If the Property Data spans multiple message Chunks, then the use of "command": "notify" is recommended.

11.2 Replying to Subscription Messages

A device that receives a Subscription message for any subscribed Resource Data shall reply with a Reply to Subscription message, so the original sender is aware of the success or failure of a command. If there is a failure, the sender may decide to retry or end the Subscription by sending a Subscription Message with the "command" Property set to "end".

If an Initiator receives but cannot handle a "partial" or "full", the Initiator shall:

1. Return a Reply to Subscription message to report success in receiving the message.
2. Send an Inquiry: Get Property Data to remain synchronized with subscribed data of the Responder.

11.3 Setting the Property Data when Sending the "command" Property with the Property Value "full"

When Property Data is an array or object the Property Data for the update shall use the same format as data in a Partial Inquiry: Set Property message (See Section 8).

11.3.1 Example: Subscription to a Simple Property Resource

Table 40 Step 1: Initiator Sends a Subscription Message

Header Data	{"command": "start", "resource": "CurrentMode"}
Property Data	<i>none</i>

Table 41 Responder Sends Reply to Subscription Message

Header Data	{"status": 200, "subscribeId": "sub328"}
Property Data	<i>none</i>

Table 42 Step 2: Responder Sends Subscription Message using a "full" command

Header Data	{"command": "full", "subscribeId": "sub328"}
Property Data	"multichannel"

Table 43 Initiator Sends Reply to Subscription Message

Header Data	{"status": 200, "subscribeId": "sub328"}
Property Data	<i>none</i>

11.4 Example Set of Transactions with Initiator Subscribing to Responder Resources

The following shows messages exchanged between the two devices.

This diagram shows the example life cycle of a single Subscription. There may be several Subscription life-cycles between two devices, each subscription having its own Subscription Id.

A hypothetical Manufacturer defined Resource called "**X-ProgramEdit**" retrieves the following Resource Data using an Inquiry: Get Property Data message:

Table 44 Initiator Sends Inquiry: Get Property Data

Header Data	{"resource": "X-ProgramEdit", "resId": "abcd"}
Property Data	<i>none</i>

Table 45 Responder Sends Reply to Get Property Data

Header Data	{"status": 200}
Property Data	{ <pre> "name": "PIANO 4", "lfoSpeed": 10, "lfoWaveform": "triangle", "pitchEnvelope": { "rates": [80,67,95,60], "levels": [60,50,50,50] } </pre>

The **X-ProgramEdit** Resource in the **ResourceList** has previously declared the Resource to be subscribable with the "**canSubscribe**" Property set to true.

The Initiator subscribes by sending a Subscription message:

Table 46 Initiator Sends a Subscription Message

Header Data	{"command": "start", "resource": "X-ProgramEdit", "resId": "abcd"}
Property Data	<i>none</i>

Table 47 Responder Sends Reply to Subscription Message

Header Data	{"status":200,"subscribeId":"sub138047"}
Property Data	<i>none</i>

The user changes the LFO waveform on the Responder Device to a square. The Responder Device sends a Subscription message to inform the Initiator of the update:

Table 48 Responder Sends Subscription Message using a "full" command

Header Data	{"command":"partial","subscribeId":"sub138047"}
Property Data	{ "/lfoWaveform":"square" }

Table 49 Initiator Sends Reply to Subscription Message

Header Data	{"status":200}
Property Data	<i>none</i>

The user hits a randomize button on the Responder Device. As there are numerous changes the Device decides to notify the Initiator to get a complete new set of Resource Data:

Table 50 Responder Sends Subscription Message using a "full" command

Header Data	{"command":"notify","subscribeId":"sub138047"}
Property Data	<i>none</i>

Table 51 Initiator Sends Reply to Subscription Message

Header Data	{"status":200}
Property Data	<i>none</i>

The Initiator shall refresh its Resource Data using the same Header data it used originally:

Table 52 Initiator Sends Inquiry: Get Property Data

Header Data	{"resource":"X-ProgramEdit","resId":"abcd"}
Property Data	<i>none</i>

Table 53 Responder Sends Reply to Get Property Data

Header Data	{"status":200}
Property Data	{ "name": "PIANO 4", "lfoSpeed": 50, "lfoWaveform": "saw", "pitchEnvelope": {

	<pre> "rates": [25,46,17,0], "levels": [80,10,36,94] } } </pre>
--	---

The user changes the "name" Property from the Initiator. The Initiator uses a Partial Set.

Table 54 Responder Sends Subscription Message using a "full" command

Header Data	{"resource":"X-ProgramEdit","resId":"abcd","setPartial":true}
Property Data	{ "/name":"Broken Piano" }

Table 55 Initiator Sends Reply to Subscription Message

Header Data	{"status":200}
Property Data	<i>none</i>

The Responder receives a Control Change MIDI message that modifies the LFO Speed and sets it to the value of 70. The Responder sends out a Subscription Message to the Initiator.

Table 56 Responder Sends Subscription Message using a "full" command

Header Data	{"command":"partial","subscribeId":"sub138047"}
Property Data	{ "/lfoSpeed":70 }

Table 57 Initiator Sends Reply to Subscription Message

Header Data	{"status":200}
Property Data	<i>none</i>

The Initiator decides that it no longer needs to keep the Resource Data in sync and decides to end the Subscription.

Table 58 Initiator Subscription Message

Header Data	{"command":"end","subscribeId":"sub138047"}
Property Data	<i>none</i>

Table 59 Responder Sends Reply to Get Property Data

Header Data	{"status":200}
Property Data	<i>none</i>

11.5 Subscription End

If an Initiator no longer needs ongoing, active access to the Property Data, it should end the subscription.

A subscription should be considered active until any of the following events occur:

- The Initiator unsubscribes by sending a Subscription message with the "**command**" Property set to "end" to the Responder.
- The Responder ends a subscription by sending a Subscription message with the "**command**" Property set to "end" to the Initiator.
- An Invalidate MUID message is received with a Target MUID that matches the MUID of either the Initiator or the Responder, at which point all subscriptions are ended (without the use of an End Subscription message).

12 Terminating Transactions and Transaction Timeouts

Property Exchange provides mechanisms to Terminate an Inquiry, to inform that a timeout wait is occurring, or to report a timeout condition by way of MIDI-CI ACK and NAK messages. These notifications may be sent by an Initiator or a Responder.

MIDI-CI ACK and NAK messages include the Request ID of the Transaction which is being addressed by these mechanisms.

See MIDI Capability Inquiry v1.2 [MA03] on how to send MIDI-CI ACK and NAK messages and for the use of the Request Id.

MIDI-CI ACK and NAK messages replace the Notify message which was defined in prior Property Exchange specifications. Some requirements of the Notify message remain for backward compatibility, as defined below.

12.1 Terminate Transaction

12.1.1 MIDI-CI NAK: Responder Terminates Transaction

A Responder may terminate a currently running Transaction by sending a MIDI-CI NAK message with a Status Code = 0x20 and Status Data = 0x00.

If the Original Transaction Sub-ID#2 Classification field is set to 0x34 (MIDI-CI Inquiry: Get Property Data), then the Initiator shall immediately stop expecting any further Chunks related to the declared Request Id.

If the Original Transaction Sub-ID#2 Classification field is set to 0x36 (MIDI-CI Inquiry: Set Property Data), then the Initiator shall stop sending Chunks related to the declared Request Id.

Initiator and Responder shall immediately consider the Transaction concluded, without a reply from the Initiator.

12.1.2 MIDI-CI NAK: Initiator Terminates Transaction

An Initiator may inform a Responder that it has terminated a currently running Transaction by sending a MIDI-CI NAK message with a Status Code is set to 0x20 and Status Data = 0x01.

If the Original Transaction Sub-ID#2 Classification field is set to 0x34 (MIDI-CI Inquiry: Get Property Data), then the Responder shall immediately stop sending chunks related to the declared Request Id.

If the Original Transaction Sub-ID#2 Classification field is set to 0x36 (MIDI-CI Inquiry: Set Property Data), then the Initiator shall stop expecting any further chunks related to the declared Request Id.

Initiator and Responder shall immediately consider the Transaction concluded without a reply from the Responder.

12.1.3 Notify: Terminate Inquiry

The Notify: Terminate Inquiry message should not be sent. MIDI-CI NAK messages as defined in Sections 12.2.1 and 12.2.2 provide a superior mechanism. However, for backward compatibility, Property Exchange Receivers (Initiators and Responders) shall terminate the currently running request in response to a Notify message with a "status" Property with a value of 144. All inquiries related to the Request Id used shall cease immediately.

Note: Previous versions of this specification had a different definition: Either device may terminate a currently running request by sending a Notify message with a "status" Property with a value of 144. All inquiries related to the Request Id used shall cease immediately.

Table 60 Initiator or Responder Sends Notify Message

Header Data	{"status":144}
-------------	----------------

Property Data	<i>none</i>
---------------	-------------

12.2 Timeout

A Responder shall send a Reply to any Inquiry within the 3-seconds timeout period, unless the Responder requests an extended time for processing.

MIDI-CI provides a mechanism for Property Exchange messages to be sent in multiple chunks. If a device sends a message in multiple chunks, the time between consecutive chunks shall be under 3 seconds.

12.2.1 MIDI-CI ACK: Timeout Wait

If a Responder cannot generate and send either a suitable reply or a next Chunk of a Transaction within the default 3-second timeout, then it should send a MIDI-CI ACK message with the associated Request Id, a Status Code 0x10 (Timeout Wait), and a Status Data value indicating the timeout wait period to keep the current Transaction active. The Status Data value specifies the additional timeout period in multiples of 100 milliseconds.

On reception, the Initiator shall extend its timeout period by at least the requested period (timeout wait period is added to the current timeout period).

Example 1:

A Responder receives an Inquiry, is processing that request, and is unable to begin the reply before the default timeout period (3 seconds). This Responder needs at most a further 6 seconds (9 seconds total). The Responder sends a MIDI-CI ACK message with a Status Code of 0x10 and Status Data of 0x3C (6 seconds).

Example 2:

A Responder has already sent some data and is building the next Chunk in a reply, and is unable to send the next Chunk before the timeout period. The Responder sends a MIDI-CI ACK message with a Status Code = 0x10 and Status Data = 0x1E to extend the timeout period by a further 3 seconds.

Table 61 MIDI-CI ACK Message used for Flow Control

Value	Parameter
F0	System Exclusive Start
7E	Universal System Exclusive
7F	Device ID: Source or Destination (depending on type of message): 7F: To/from Function Block
0D	Universal System Exclusive Sub-ID#1: MIDI-CI
7D	Universal System Exclusive Sub-ID#2: MIDI-CI ACK Message
02	MIDI-CI Message Version/Format v 1.2
4 bytes	Source MUID (LSB first)
4 bytes	Destination MUID (LSB first)
1 byte	Original Sub Id Classification
0x10	ACK Status Code – Timeout Wait
1 byte	Status Data is the time to wait in multiples of 100 milliseconds (0-12.7 seconds). For Example: 0x01 = 100 Milliseconds

	0x1E = 3 Seconds 0x3C = 6 Seconds 0x7F = 12.7 Seconds
1 byte	PE Request Id
2 bytes	PE Chunk Number 1 (LSB First)
0x00 0x00	Reserved
0x00 0x00	No message - Message length = 0
F7	End Universal System Exclusive

12.2.2 Notify: Timeout Wait (deprecated)

Use of the Notify: Timeout Wait message has been deprecated, and future implementations should instead use the MIDI-CI ACK as defined in 12.2.1. However, for backward compatibility, a Receiver should handle a received Notify message with status 100 the same as the newer ACK Timeout Wait message as defined in section 12.2.1

Note: Previous versions of this specification had a different definition: If a Responder has already sent some data and is building the next chunk in a reply, and if it cannot send the next chunk before the timeout period, the Responder may send a Notify message with a "status" Property with a value of 100.

Table 62 Responder Sends Notify Message

Header Data	{"status":100}
Property Data	None

Note: This message has been deprecated, and future implementations should use the MIDI-CI ACK methods in 11.3.1

12.2.3 MIDI-CI NAK: Timeout Has Occurred

If an Initiator does not receive an expected reply, a next Chunk, or a Timeout Wait message within the timeout period, the Initiator may terminate the Transaction by sending a MIDI-CI NAK with Status Code = 0x42.

Initiator and Responder shall immediately consider the Transaction concluded, without a reply from the Responder.

12.2.4 Notify: Timeout Has Occurred

Use of the Notify: Timeout Has Occurred message has been deprecated, and future implementations should instead use the MIDI-CI ACK method as defined in 12.2.3. However, for backward compatibility, a device should handle a received Notify message with status 408 the same as the newer NAK Timeout Has Occurred message as defined in section 12.2.3.

Note: Previous versions of this specification had a different definition: If a Responder cannot generate a suitable reply or next chunk in a message within the timeout period, it may send a Notify message with a "status" Property with a value of 408 to cancel the inquiry.

If an Initiator does not receive an expected Reply, next message chunk, or a Notify: Timeout Wait message within the timeout period, it may terminate the inquiry by sending a Notify message with a "status" Property with a value of 408.

Table 63 Initiator Sends Notify Message

Header Data	{"status":408}
Property Data	<i>none</i>

13 Resource Link Properties

Resources that return an object or an array of objects may provide links to other related Resources. Links can refer to MMA/AMEI defined Resources or manufacturer-specific Resources.

Table 64 Link Properties

Property Key	Property Value Type	Description
resource	string (max 36 characters)(required)	This is the Resource to be linked.
resId	string (max 36 characters)	Resource ID, the identifier used to select the desired Payload Data entry. See Section 8.5.
title	string	This is the human readable title for this link. Fallback on using the "title" Property in ResourceList if not supplied.
role	string (max 32 characters)	Role declares to the application on how to use the link. Future specifications shall define any roles for any linked Resources and whether such roles are optional or required. Such specifications shall include but are not limited to: <ol style="list-style-type: none"> 1. MMA/AMEI specifications for defining common roles. 2. Resource specifications which define a specific role for a specific Resource. 3. Manufacturer specific definitions, where such role shall be prefixed with "x-".

The role Property indicates how a Link will be used:

Examples:

- To use additional, related Properties.
- To use an additional component.
- To select an additional component from a list.
- To open a graphic editor.

Table 65 Example of Using Links With a "resId" Property

Property Data	<pre>[{ "title":"Ch.1", "channel":1, "bankPC":[0,0,0], "programTitle":"Piano", "links":[{"resource":"CMList", "resId":"singch1"}, {"resource":"X-ProgramEdit", "resId":"singch1"}] }]</pre>
---------------	---

In order to use the first link in the data above, the Initiator sends an Inquiry: Get Property Data message using the associated "resId" Property.

Table 66 Initiator Sends an Inquiry: Get Property Data Message

Header Data	{"resource": "CMList", "resId": "singch1"}
Property Data	<i>none</i>

Table 67 Example of Using Links Without a "resId" Property

Property Data	<pre>{ "version": "1.0", "manufacturerId": [125,0,0], "manufacturer": "Educational Use", "familyId": [0,0], "family": "Test Range", "modelId": [48,0], "model": "MIDI-CI Test Workbench", "versionId": [0,0,1,0] "links": [{"resource": "Tempo", "title": "BPM"}, {"resource": "LocalOn", "title": "Local On/Off"}] }</pre>
---------------	---

In this example two links are available. The Initiator may choose to present this information to the user.

Educational Use: MIDI-CI Test Workbench	
BPM	120↑
Local On/Off	<input checked="" type="checkbox"/>

The **Tempo** and **LocalOn** Resources in this example are Simple Property Resources (See Section 8.7) and are shown as editable in line.

13.1 Using a "role" Property of a Link

A Resource may have a close connection to or a dependency on another Resource. The two Resources might be used together to achieve a larger set of goals. Or one Resource might naturally lead to the use of another Resource in a user's typical workflow.

A Resource may use optional "links" Property to declare these connections to other related Resources. Within each Link entry, an optional "role" Property declares the purpose of the Link. In this specification, the role declares that a Link is used as an Action. An Action presents the user with the ability to trigger a function on a Device. Triggered Actions may change a Device's state, settings, or configuration. Those changes might cause a change in the Property Data of the Device's Resources.

A Responder should only declare Roles which the Responder has the ability to process. An Initiator shall only use Roles that are declared by the Responder.

13.1.1 Role: simpleAction

A Property Exchange Link which contains a **role** Property with the value of `simpleAction` is used to indicate to the Initiator that there is an option for a user to trigger a Resource defined action. An Action is often displayed as a button for a user to press.

Example Simple Actions which might be used include:

- Internally Copy (or paste) the current entry from a Resource List
- Initialize or delete data
- Save changes
- Launch an internal function that cannot be triggered by Channel Voice Messages
- Other function(s) defined by any Resource

The `simpleAction` may directly trigger the Action, or may present a confirmation dialog box. This optional dialog box presents the user with the choice to proceed with or cancel the Action.

An activated Action shall use an Inquiry: Set Property Data message. The Property Data used is the data contained in the "**propData**" Property contained in the Link entry.

The Header Data in the Reply to Set Property Data message may contain an optional "**msg**" Property which is a text string that should be displayed to the user.

Table 68 Link object Properties for use with a "simpleAction" Role

Property Key	Property Value Type	Description
role	string. set to "simpleAction"	Declares that the role of this Link entry is a simple Action
resource	string (max 36 characters)(required)	Action Resource to use with an Inquiry: Set Property Data. This Action Resource shall be included in ResourceList.
resId	string (max 36 chars, "a-z", "0-9" or "_" characters only)	If the Action Resource declared in the ResourceList requires a resId Property, then this shall be filled.
title	string	An optional title to display.
propData	defined by the Resource schema* required**	JSON Property data which is sent using an Inquiry: Set Property Data message when the Action occurs.
confirmMsg	string	An optional confirmation dialog message presented to the user if confirmation is required before proceeding. This is used when the user is given an opportunity to not proceed. This is recommended for Actions that delete or make significant changes to data
confirmType	enum "okcancel", "yesno"	This is the type of confirmation box. okcancel - The message box contains two push buttons: OK and Cancel. yesno - The message box contains two push buttons: Yes and No.
confirmIcon	enum "exclamation", "warning", "information", "question"	Displays an icon in the confirmation box that provides a hint to the user on the type of confirmation provided.

* When the simple Action occurs, the "propData" Property Value must match the schema of the Resource declared in the Link entry.

**only required when role Property value is "simpleAction"

13.1.1.1 Example Copy and Paste in ProgramList

A device might wish to allow its users to copy and paste Programs contained in a **ProgramList** Resource.

Table 69 Example of ProgramList Resource using “simpleAction” Property

Property Data	<pre>[{ "bankPC": [0,1,1], "title": "Piano", "links": [{ "resource": "X-Copy", "propData": "0_1_1", "role": "simpleAction", "title": "copy" }, { "resource": "X-Paste", "propData": "0_1_1", "role": "simpleAction", "title": "paste" }] }, { "bankPC": [0,1,3], "title": "Synth", "links": [{ "resource": "X-Copy", "propData": "0_1_3", "role": "simpleAction", "title": "copy" }, { "resource": "X-Paste", "propData": "0_1_3", "role": "simpleAction", "title": "paste" }] }, ... { "bankPC": [0,1,120], "title": "<Empty>", "links": [{ "resource": "X-Paste", "propData": "0_1_120", "role": "simpleAction", "title": "paste" }] }]</pre>
---------------	---

The Initiator would likely display the **ProgramList** Property Data similar to the following:

Title	Copy	Paste
Piano	Copy	Paste
Synth	Copy	Paste
...		
<Empty>		Paste

Figure 3 Example of Initiator displaying simpleAction Roles

An initiator understands that the copy and paste are buttons because the "role" Property has the value of "simpleAction"

When the button is pressed the "resource" and "resId" (if provided) are used in an Inquiry: Set Property Data Message:

Table 70 Initiator Sends an Inquiry: Set Property Data Message

Header Data	{"resource": "X-Copy"}
Property Data	"0_1_1"

The Responder performs the Initialize Program function and returns a reply.

Table 71 Initiator Sends Reply to Inquiry: Set Property Data Message

Header Data	{"status":200,"message":"Copy of Piano Program successful"}
Property Data	<i>none</i>

The Property Data string in the Response is displayed to the user. If the Property Data is empty, then the user is not displayed any message.

13.1.2 Role: selectorAction

A Property Exchange Link may provide options for user selection. The Link entry shall include three parts:

- A "role" Property with the value of "selectorAction".
- A List Resource ("resource" and "resId") that contains two or more user selectable options.
- An Action Resource ("actionResource" and "actionResId") that is the target of the options. This Resource shall be a Simple Property Resource.

The Initiator might display the selectorAction as a button which, when enacted, displays the available options to the user. The user may select a single option from this list. This selection is then used with the Action Resource.

Examples of Selector Actions which might be presented:

- Audio files available to be used in a loop
- An arpeggiator style to apply

- A song or sequence to be played
- An audio output assignment for a synthesizer part
- Other function(s) defined by any Resource

Table 72 Link object Properties for use with a "selectorAction" Role

Property Key	Property Value Type	Description
role	string. set to "selectorAction"	Declares that the role of this Link entry is a select Action
resource	string (max 36 characters) (required)	The List Resource to use with an Inquiry: Get Property Data message. This List Resource shall be included in ResourceList
resId	string (max 36 chars, "a-z", "0-9" or "_" characters only)	If the List Resource declared in the ResourceList requires a resId Property, then this shall be filled.
title	string	An optional title to display.
actionResource	string, required*	This is the Action Resource to use with an Inquiry: Set Property Data message to declare to the Responder which List Resource entry the user has selected.
actionResId	string (max 36 chars, "a-z", "0-9" or "_" characters only)	If the Action Resource declared in the ResourceList requires a resId Property, then this shall be filled.
selectedProperty	string, required*	The Property Key to use from the selected List Entry as the value of the Trigger Resource Property Data.

*only required when role Property value is "selectorAction"

When a user selects a List Resource entry, the Initiator shall send an Inquiry: Set Property Data message to set the associated Resource which is declared in the “**actionResource**” property. The Property Data used is discovered by looking at the "**selectedProperty**" Property Value, and finding the matching Property Key in the selected List Resource entry. The Property Data must match the schema of the associated Resource.

The Header Data in the Reply to Set Property Data message may contain an optional "**msg**" Property which is a text string that should be displayed to the user.

13.1.2.1 Example Selection of an Audio output on the ChannelList Resource from a Manufacturer defined List Resource

A Device might allow users to select an audio output assignment for each MIDI Channel. The example device uses 2 MIDI Channels, Piano and Bass, so there are two "**selectorAction**" Roles available. The example follows the transactions when the user decides to set the audio output for the Piano part.

Step 1: Initiator Gets a ChannelList**Table 73 Example of ChannelList Resource using “selectAction” Property**

Property Data	<pre>[{ "Channel": 1, "bankPC": [0,1,120], "title": "Piano", "links": [{ "resource": "X-AudioOutList", "resId": "ch1", "role": "selectorAction", "title": "Stereo Main", "selectedProperty": "output", "actionResource": "X-Output", "actionResId": "ch1" }], }, { "Channel": 2, "bankPC": [0,1,60], "title": "Bass", "links": [{ "resource": "X-AudioOutList", "resId": "ch2", "role": "selectorAction", "title": "Stereo Main", "selectedProperty": "output", "actionResource": "X-Output", "actionResId": "ch2" }] }]</pre>
---------------	--

The Initiator discovers the existence of a Piano channel and a Bass channel in the Responder, each with selectable audio output assignments.

Step 2: Initiator Displays to the User

The Initiator would likely display the **ChannelList** Property Data, with a button to display audio output assignment selection for each of the two channels:

Channel	Title	Channel Output
1	Piano	Stereo Main
2	Bass	Stereo Main

Figure 4 Example of Initiator displaying selectAction Roles

Step 3: User Selects a selectorAction Button

When the "Stereo Main" button is pressed on Channel 1, the Link "resource" and "resId" Properties, previously declared in the **ChannelList** Property Data above, are used in an Inquiry: Get Property Data message to request the **X-AudioOutList**.

Table 74 Initiator Sends an Inquiry: Get Property Data Message

Header Data	{"resource":"X-AudioOutList","resId":"ch1"}
Property Data	none

Table 75 Responder Sends Reply to Get Property Data

Header Data	{"status":200}
Property Data	[{"output": "LM", "name": "Left Main", "selected": false}, {"output": "RM", "name": "Right Main", "selected": false}, {"output": "SM", "name": "Stereo Main", "selected": true}, {"output": "LA", "name": "Left Alt", "selected": false}, {"output": "RA", "name": "Right Alt", "selected": false}, {"output": "SA", "name": "Stereo Alt", "selected": false}]

Step 4: Initiator Displays to the User

The Initiator would likely display the **X-AudioOutList** Property Data as list of available audio outputs selections:

Output	Selected
Left Main	<input type="checkbox"/>
Right Main	<input type="checkbox"/>
Stereo Main	<input checked="" type="checkbox"/>
Left Alt	<input type="checkbox"/>
Right Alt	<input type="checkbox"/>
Stereo Alt	<input type="checkbox"/>

Figure 5 Example of Initiator displaying selectAction Roles

Step 5: User Selects an Audio Output

If the user decides to select the "Left Alt" Resource List item by clicking on the table row, then the Initiator sends an Inquiry: Set Property Data message with the "resource" and "resId" Properties from the "actionResource" Property in the Link entry.

In this example the **X-AudioOutList** List Resource entry selected is:

```
{"output": "LA", "name": "Left Alt", "selected": false}
```

The "selectProperty" Property Value is "output". The "output" Property Value from the entry is "LA". This is used as the Property Data.

Table 76 Initiator Sends an Inquiry: Set Property Data Message

Header Data	{"resource":"X-Output","resId":"ch1"}
Property Data	"LA"

The Responder performs the Initialize Program function and returns a reply.

Table 77 Initiator Sends Reply to Subscription Message

Header Data	{"status":200,"message":"The Output of Channel 1 is set to Left Alt."}
Property Data	<i>none</i>

The process for selecting the audio output for Channel 1 is finished.

13.2 Subscriptions and Linked Resources

When a subscription to a Resource is established, linked Resources are not automatically included in the subscription. If the Initiator also needs to subscribe to a linked Resource, an additional subscription is required.

In general, an Initiator does not need to subscribe to a linked Resource until it accesses the link and begins ongoing, active use of the linked Resource.

14 Resource: ResourceList

The "**ResourceList**" Resource provides a comprehensive method for an Initiator to understand how to control the Responder using Property Exchange. Response to **ResourceList** is mandatory for all devices that support Property Exchange.

When a Responder receives an Inquiry: Get Property Data message with "**ResourceList**" the Responder shall send a Reply to Get Property Data message with a list of all Resources it supports so the Initiator understands its availability and settings. (**ResourceList** is itself a List Resource.)

Each Resource listed provides various qualifiers on how that Resource can be used, such as being able to use Inquiry: SET Property Data or Subscription messages. Resources defined by AMEI/MMA specifications have defined default settings so that devices can have a shorter **ResourceList**. Devices may override individual settings as needed.

ResourceList is usually the first inquired Resource before requesting any other Property Exchange Resource. The reply to **ResourceList** shall include all the MMA/AMEI and Manufacturer defined Resources supported by a device. The **ResourceList** Property Data returned shall not include an entry for **ResourceList** itself in the list of objects.

14.1 Request ResourceList using Inquiry: Get Property Data

An Initiator requests the **ResourceList** Resource using the MIDI-CI Inquiry: Get Property Data message.

Table 78 Initiator Sends an Inquiry: Get Property Data Message

Header Data	{"resource":"ResourceList"}
Property Data	<i>none</i>

14.2 Property Data for ResourceList Returned via a Reply to Get Property Data Message

When a Responder receives an Inquiry: Get Property Data message with a **ResourceList** request, the Responder sends a Reply to Inquiry: Get Property Data message with an array of objects in the Property Data field. Each object in the array declares a Resource which is supported by the Responder.

The structure of objects in a reply to a **ResourceList** request are as follows:

Table 79 Link Properties

Property Key	Property Value Type	Description
resource	string (max 36 characters)(required)	This is the Resource name. Manufacturer custom Resource names always start with "X-". See Section 8.4.
canGet	boolean (default: true)	Declares whether this Resource is retrievable by an Inquiry: Get Property Data message. For example, a firmware upload Resource may set this to false.
fcOnGet	boolean default: false	This Resource supports retrieving data using Flow Control when requesting an Inquiry: Get Property Data Transaction. See Section 15.
canSet	enum ("none", "full", "partial") (default:none)	Declares the supported use of Inquiry: Set Property Data message on this Resource. When set to "partial", "full" is also supported. See Section 10.

fcOnSet	boolean default: false	This Resource requires the Initiator to use Flow Control on an Inquiry: Set Property Data Message Transaction. See Section 15.
canSubscribe	boolean (default: false)	Declares whether subscription can be used on this Resource. See Section 11.
requireResId	boolean (default: false)	Inquiry shall contain resId Property in the Header Data. See Section 8.
mediaTypes	array of mediaType strings (default: [application/json])	Media Types as defined by RFC-6838 [RFC6838]. Common types are listed in Media Types [EXT06]. This is used to declare the type of data. For example, one Resource might allow MIDI files, but another allows images or audio data only. See Section 6.1.5.
encodings	array of strings (default: ["ASCII"])	This is the list of encodings the Resource supports. See Section 6.1.6 for the list of acceptable values.
schema	JSON Schema Object	Manufacturer Specific Resources should define their Properties here. For MMA/AMEI defined resources, do not specify a different schema here*. See Section 14.4 for more information

*AMEI/MMA Resources have JSON Schemas. However, they do need to be declared in **ResourceList**.

14.2.1 Additional Properties for List Resources

Any Resource which is a List Resource declared as an entry in the **ResourceList** Property Data shall include the following additional Properties if they are applicable to that List Resource.

Table 80 ResourceList Properties for List Resource

Property Key	Property Value Type	Description
canPaginate	boolean (default:false)	If the Resource returns an array of objects, then the Resource may declare whether pagination is supported. See Section 8.6.2.
columns	array of Column objects	This is an optional array of data to know which Properties, and in what order, to display in a table. See Section 14.5 for more information.

*Note: By default, List Resources are defined to have **canPaginate** set to false. However, a List Resource specification may declare that pagination is mandatory (true) when implementing the Resource.*

Table 81 Example Transaction for ResourceList: Initiator Sends an Inquiry: Get Property Data Message

Header Data	{"resource":"ResourceList"}
Property Data	none

Table 82 Responder Sends Reply to Get Property Data

Header Data	{"status":200}
-------------	----------------

Property Data	<pre>[{"resource": "DeviceInfo"}, {"resource": "ChannelList", "canSubscribe": true}, {"resource": "ProgramList"}, {"resource": "CMList"}, { "resource": "BufferState", "canSet": "full", "mediaTypes": ["application/octet-stream"], "encodings": ["MCoded7"] }]</pre>
---------------	--

14.3 Minimized Listing of AMEI/MMA Defined Resources in ResourceList using Default Values

Each **ResourceList** object includes a set of Properties that explain the details about how that particular Resource may be used on the Device. Many of those Properties define a default value within the range of allowed Property Values.

If a Device’s implementation uses the default value for a Property, then that Property is not required to be included in the **ResourceList** object.

If a Device’s implementation does not use the default value for a Property, then the **ResourceList** object shall include the Property and Property Value to declare a value which overrides the default value.

The default values for "**schema**", "**requireResId**", "**canPaginate**", "**canSet**", "**canGet**" Properties shall not be overridden on AMEI/MMA defined Resources unless the specification for that Resource specifically defines an allowance for overriding.

14.3.1 Example: ResourceList Object for the ChannelList Resource

Table 83 declaring all the default values

Property Data	<pre>[{ "resource": "ChannelList", "canGet": true, "canSet": "none", "canSubscribe": false, "canPaginate": false, "schema": { "type": "array", "title": "Channel List", "\$ref": "http://schema.midi.org/property-exchange/M2-105-S_v1-0_ChannelList.json" }, "columns": [{"property": "title", "title": "Title"}, {"property": "channel", "title": "MIDI Channel"}, {"property": "programTitle", "title": "Program Title"}, {"link": "ProgramList", "title": "Program List"}] }]</pre>
---------------	---

	<pre> }] </pre>
--	------------------------------------

Table 84 Minimized object using all the default values

Property Data	<pre> [{ "resource": "ChannelList" }] </pre>
---------------	--

Table 85 Minimized object using the default values for some Properties and declaring non-default values for other Properties

Property Data	<pre> [{ "resource": "ChannelList", "canSubscribe": true, "columns": [{ "property": "channel", "title": "MIDI Channel" }, { "property": "title", "title": "Title" }, { "link": "ProgramList", "title": "Program List" }, { "link": "CMList", "title": "Channel Controllers" }] }] </pre>
---------------	--

14.4 Schema Property for Manufacturer Specific JSON Resources

AMEI/MMA standardized Resources do not require exchange of a JSON Schema and shall not be overridden.

Manufacturer-specific JSON Resources shall supply at least a simple JSON Schema. The JSON Schema shall supply at least **“title”** and **“type”**. Including these 2 objects is enough to meet the minimum requirement. Supplying a fully descriptive JSON Schema is optional.

For devices that want to exchange a more complete JSON Schema, conformance to JSON Schema draft 4 is recommended. Other revisions of JSON Schema may be declared using the **“\$schema”** property.

14.4.1 Including a Schema Property in the ResourceList

Any Manufacturer-specific JSON Resources that has a simple structure or is a Simple Property Resource may include the complete JSON Schema in a **“schema”** Property.

Table 86 Example Resource List Object with a “schema” Property

Property Data	<pre> [{ "resource": "X-WIFIOn", "canSet": "full", "schema": { "title": "WIFI Enabled", "description": "Turn WIFI on or Off", "type": "boolean" } }] </pre>
---------------	---

	<pre> }] </pre>
--	------------------------------------

14.4.2 Providing a Reference to an Expanded JSON Schema in the ResourceList

It is recommended that Manufacturer Specified Resources use a reference to supply a fully descriptive JSON Schema for complex data. JSON Schema provides "\$ref" keyword in the "schema" Property to support this concept.

From JSON Schema: "The value of \$ref is a URI, and the part after # sign (the “fragment” or “named anchor”) is in a format called JSON Pointer."

To indicate that the JSON Schema comes from the device, the following URI scheme is used:

"midi+jsonschema://<JSON Schema Id>"

Table 87 Example ResourceList Object using a JSON Schema reference

Property Data	<pre> [{ "resource": "X-Globset", "canSet": "full", "schema": { "title": "System Settings", "description": "Set the global settings here", "type": "object", "\$ref": "midi+jsonschema://globalSchema" } }] </pre>
---------------	--

To retrieve a JSON Schema an Initiator may send an Inquiry: Get Property Data message with the Resource set to "JSONSchema".

Note: The JSONSchema Resource is defined in the Foundational and Basic Resources specification.

Table 88 Example of a JSON Schema reference: Initiator Sends an Inquiry: Get Property Data Message

Header Data	{"resource": "JSONSchema", "resId": "globalSchema"}
Property Data	<i>none</i>

Table 89 Responder Sends Reply to Get Property Data

Header Data	{"status": 200}
Property Data	<pre> { "\$schema": "http://json-schema.org/draft-07/schema#", "type": "object", "properties": { "deviceName": { "type": "string" }, "audioOut": { </pre>


```

        "type": "string",
        "enum": ["USB","line"]
    },
    "cvPitch": {
        "type": "string",
        "enum": ["v/oct","hz/oct"]
    },
    "gate": {
        "type": "string",
        "enum": ["s-trig","v-trig"]
    }
}
    }
}
    
```

14.5 Using Columns for List Resources

The "columns" Property may be used to describe a table for presenting selected data from a List Resource.

The "columns" Property contains an array of objects. Each object becomes a column in the table. Each object in the array contains a "property" or "link", and a "title".

Table 90 Example ResourceList Object Record Containing a "columns" Property

Property Data	<pre> [{ "resource": "ChannelList", "canSubscribe": true, "columns":[{"property": "channel", "title": "MIDI Channel"}, {"property": "programTitle", "title": "Program Name"}, {"link":"CMList", "title": "Controllers"}] }] </pre>
---------------	--

Property Value of "title" in each object becomes the title for each column in the table.

In this example, we get the title for three columns: channel, programTitle, and CMList.

MIDI Channel	Program Name	Controllers

Property Data from a List Resource provides the data to fill the table.

Table 91 ChannelList Result for List Object Above



Property Data	<pre>[{ "title":"Ch.1", "channel":1, "bankPC":[0,0,0], "programTitle":"Piano", "links":[{"resource":"CMList", "resId":"singch1"}] }, { "title":"Ch.13", "channel":13, "bankPC":[0,0,1], "programTitle":"Synth Lead 2", "links":[{"resource":"CMList", "resId":"singch2"}] }]</pre>
---------------	--

Each entry in the List Resource provides the data for a row in the table. The data chosen to populate each field of a row is determined by the columns array.

In this example, the columns array above lists three items to fill the three fields of a row, two "property" items and a "link" item to be used from each entry in the List Resource:

- "channel"
- "programTitle"
- "CMList"

The data from the columns array and the List Resource in this example combine to generate the following table.

MIDI Channel	Program Name	Controllers
1	Piano	
13	Synth Lead 2	

14.6 ResourceList properties for non-JSON Resources

Property Exchange does not use the schema "type" property which is usually found in a JSON Schema because the "type" property does not support the necessary media types. That schema "type" property shall be left empty in any JSON Schema used for PE. Instead, the "mediaType" Property of the PE Header Data shall declare the data type.

The "schema" shall still provide the "title" Property of the Resource.

Table 92 Example Non-JSON Resource

Property Data	<pre>[{</pre>
---------------	------------------

	<pre> "resource": "X-Sample", "canSet": "full", "mediaTypes": ["audio/wav"], "encodings": ["MCoded7"], "schema": { "title": "Sample Upload", "description": "Upload the sample here" } }] </pre>
--	---

14.7 Extended Example of Property Data for ResourceList

This Example shows a mix of MMA/AMEI defined Resources, some with manufacturer overrides from the default Property Values, and Manufacturer defined Resources.

Table 93 Example Non-JSON Resource

Property Data	<pre> [{"resource": "JSONSchema", "encodings": ["ASCII", "zlib+Mcoded7"]}, {"resource": "DeviceInfo"}, { "resource": "ChannelList", "canSubscribe": true, "columns": [{"property": "perfChannel"}, {"property": "channel"}, {"property": "name"}, {"link": "ProgramList"}, {"link": "CMList"}, {"link": "X-Patch"}] }, {"resource": "ProgramList"}, {"resource": "CMList"}, {"resource": "ModelList"}, { "resource": "CurrentMode", "canSet": "full", "canSubscribe": true }, { "resource": "RawBuffer", "canSet": "full", "mediaTypes": ["application/octet-stream"], "encodings": ["MCoded7"] }, { "resource": "X-Tempo", </pre>
---------------	---

```

"canSet": "full",
"schema": {
  "title": "BPM",
  "description": "Set the Tempo, Global Setting",
  "type": "integer",
  "minimum": 40,
  "maximum": 240
}
},
{
  "resource": "X-ProgramEdit",
  "canSet": "patch",
  "requireResId": true,
  "schema": {
    "title": "Edit Patch.",
    "description": "Edit the Patch",
    "type": "object",
    "$ref": "midi+jsonschema://programDetails"
  }
}
]

```

14.8 JSON Schema for ResourceList

The JSON Schema for **ResourceList** can be found at:

http://schema.midi.org/property-exchange/M2-103-S_v1-1_ResourceList.json

15 Flow Control

Property Exchange contains a chunking mechanism as a way for Property Data to split over many Inquiry: Get Property Data or Inquiry: Set Property Data SysEx packets. A Device that is receiving information might experience a loss of data when buffers are full and the Device is not able to receive additional data. In Property Exchange this might occur on the receiving Device when it is unable to keep up with chunks sent due to internal processing.

Missing chunks in a small Property Data is easily managed, as the Transaction can be restarted. In larger payloads this becomes quite disruptive to the user. For example, some binary Property Data may require 1000 chunks or more. If chunk 882 fails, then the user will have to start the Transaction again.

A Receiver may mitigate these problems by sending Property Exchange Flow Control MIDI-CI ACK between each Chunk from the Device receiving the Chunks. When the Sender Device receives Property Exchange Flow Control MIDI-CI ACK, the Sender sends the next Chunk.

To use Flow Control, the following steps are used:

1. Initiator declares that Flow Control shall be used by including a "flowControl":true in the Property Exchange Header Data. This applies whether the Initiator will be the Sender or Receiver.
2. Sender sends Chunk 1
3. Receiver sends back MIDI-CI ACK for Chunk 1 using a Status Code = 0x11 (Chunk received and processed. Send next Chunk).
4. Sender sends the next Chunk.

Steps 3 and 4 are repeated until all Chunks are received.

5. To determine the final step, see Section 15.2 and 15.3.

This method also allows a receiver to request a resend of all parts of the last set of Chunks sent, if it has an internal failure or the Chunk Property Data is corrupted. (See Section 15.4 below)

15.1 Flow Control MIDI-CI ACK Message

A MIDI-CI ACK Message used for Flow Control is structured as follows:

Table 94 MIDI-CI ACK Message used for Flow Control

Value	Parameter
F0	System Exclusive Start
7E	Universal System Exclusive
7F	Device ID: Source or Destination (depending on type of message): 7F: To/from Function Block
0D	Universal System Exclusive Sub-ID#1: MIDI-CI
7D	Universal System Exclusive Sub-ID#2: MIDI-CI ACK Message
02	MIDI-CI Message Version/Format v 1.2
4 bytes	Source MUID (LSB first)
4 bytes	Destination MUID (LSB first)
1 byte	Original Sub Id Classification
0x11	ACK Status Code – Send next chunks

0x00	Reserved
1 byte	PE Request Id
2 bytes	PE Chunk Number 1 (LSB First)
0x00 0x00	Reserved
0x00 0x00	No message - Message length = 0
F7	End Universal System Exclusive

15.2 Using Flow Control with Inquiry: Get Property Data

To use Flow Control with an Inquiry: Get Property Data, the MIDI-CI PE Header Data "flowControl" Property indicates the Transaction should use Property Exchange Flow Control MIDI-CI ACK messages.

Table 95 Example of using Flow Control an Inquiry: Get Property Data Message

Header Data	<code>{"resource":"State","resId":"buffer","mutualEncoding":"Mcoded7","flowControl":true}</code>
Property Data	<i>none</i>

If the "flowControl" Property is not set to true, then the Responder shall return the Property Data without waiting for the Initiator to send a MIDI-CI ACK message between Chunks.

If the "flowControl" Property is used on a Resource that does not support Flow Control, then an error should be returned using a "status":406 in the Reply to Inquiry Get message.

A MIDI-CI ACK is required on the Last Chunk in an Inquiry: Get Property Data Message and is used to acknowledge that all chunks have been received.

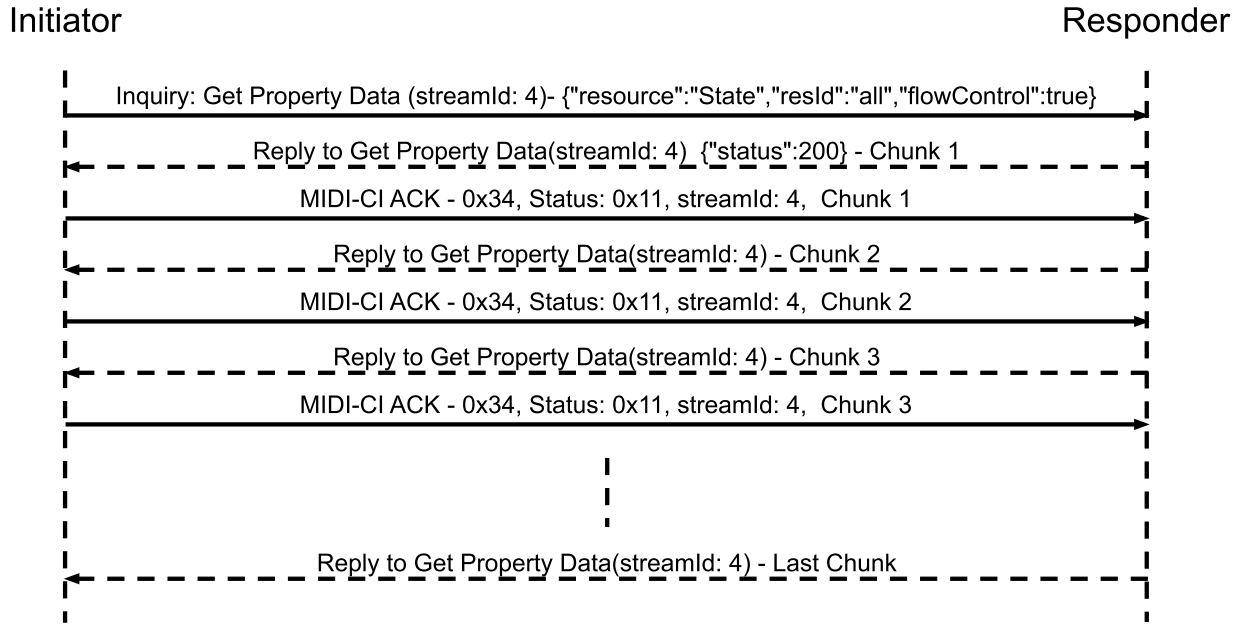


Figure 6 Flow Diagram of using Flow Control with an Inquiry: Get Property Data

15.3 Using Flow Control with Inquiry: Set Property Data

To use Flow Control with an Inquiry: Set Property Data the MIDI-CI PE Header Data "flowControl" Property indicates the Transaction should use Property Exchange Flow Control MIDI-CI ACK messages.

Table 96 Example of using Flow Control an Inquiry: Set Property Data Message

Header Data	{"resource":"State","resId":"buffer","mutualEncoding":"Mcoded7","flowControl":true}
Property Data	<data sent out over many chunks>

If the "flowControl" Property is not set to true on a Resource that has declared "fcOnSet":true, then an error should be returned using a "status":407 in the Reply to Inquiry Get message.

If the "flowControl" Property is used on a Resource that does not support Flow Control, then an error should be returned using a "status":406 in the Reply to Inquiry Get message.

A MIDI-CI ACK is not required on the Last Chunk in an Inquiry: Set Property Data Message as the Reply to Set Data Message is used to acknowledge that all chunks have been received.

Note: A Sender should not discard the transmitted data until you receive the ACK so it can be retransmitted if required.

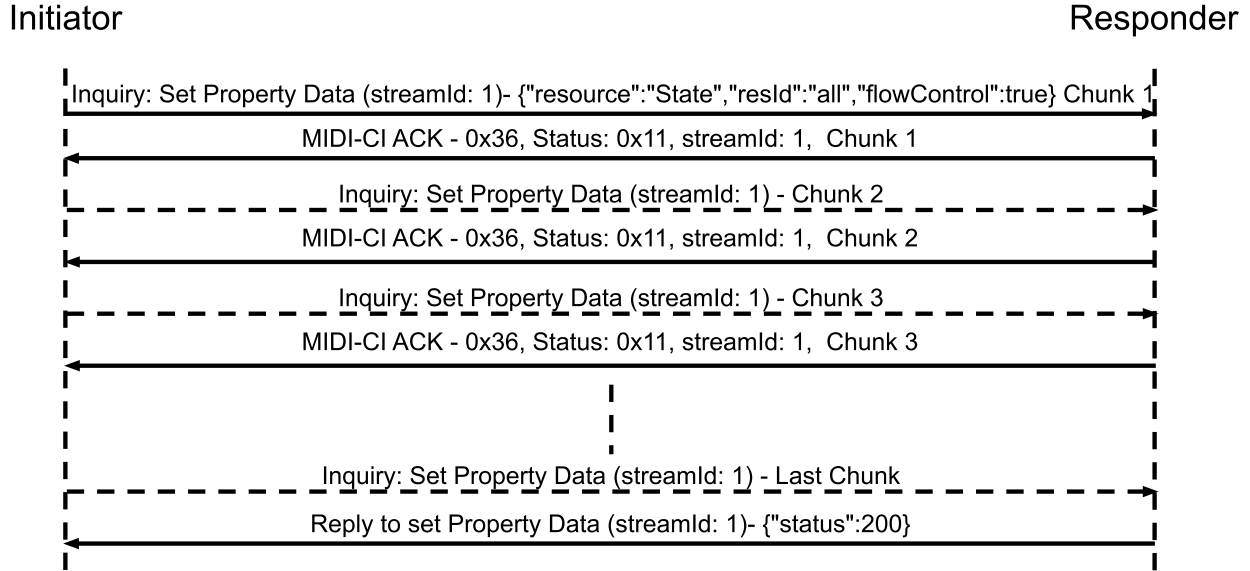


Figure 7 Flow Diagram of using Flow Control with an Inquiry: Set Property Data

15.4 Sending a Retransmit MIDI-CI NAK Message

If a Chunk is lost or corrupted during transmission, then the receiving Device should request a retransmit of the last Chunk. It does this by sending the following MIDI-CI NAK message:

Table 97 Retransmit MIDI-CI NAK Message

Value	Parameter
F0	System Exclusive Start
7E	Universal System Exclusive
7F	Device ID: Source or Destination (depending on type of message): 7F: To/from Function Block
0D	Universal System Exclusive Sub-ID#1: MIDI-CI
7F	Universal System Exclusive Sub-ID#2: MIDI-CI NAK Message
02	MIDI-CI Message Version/Format v 1.2
4 bytes	Source MUID (LSB first)
4 bytes	Destination MUID (LSB first)
1 byte	Original Sub Id Classification
0x12	NAK Status Code Resend the most recent Chunk

0x00	Reserved
1 byte	PE Stream Id
2 bytes	Last successful PE Chunk Number 1 (LSB First)
0x00 0x00	Reserved
0x00 0x00	No message - Message length = 0
F7	End Universal System Exclusive

15.5 Timeouts when using Flow Control

A Sender of Property Data may experience timeouts when waiting for a MIDI-CI ACK. When using Flow Control, if a MIDI-CI ACK is not received within 3 seconds after sending a Chunk, then the Sender should attempt to resend the Chunk. If that fails 3 times, then the Sender should fail and send a MIDI-CI NAK Terminate Message (see Section 12.1).

A Receiver of Property Data may experience timeouts. Both the Sender and Receiver shall use the methods described in Section 12.2 to manage timeouts and terminate appropriately if the timeout is not resolved.



<http://www.amei.or.jp>



<https://www.midi.org>