Common Rules for MIDI-CI Property Exchange

Version 1.0
February 20, 2020

Published By:

Association of Musical Electronics Industry AMEI
and
MIDI Manufacturers Association MMA

M2-103-UM

http://www.midi.org

PREFACE

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 Property Exchange, compliments MIDI-CI by defining a set of design rules for all Property Exchange Resources and Properties.

http://www.midi.org/

Table of Contents

1. Introduction

1.1 Background: MIDI 2.0 and MIDI-CI

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.

1.2 Property Exchange

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 meta data, manufacturer, model number, and version.

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.

1.3 Related Documents

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

MIDI 2.0 Specification, Association of Musical Electronics Industry, http://www.amei.or.jp/, and MIDI Manufacturers Association, https://www.midi.org/

MIDI Capability Inquiry (MIDI-CI), Version 1.1, Association of Musical Electronics Industry, http://www.amei.or.jp/, and MIDI Manufacturers Association, https://www.midi.org/

Foundational and Basic Resources, Association of Musical Electronics Industry, http://www.amei.or.jp/, and MIDI Manufacturers Association, https://www.midi.org/

The JSON Data Interchange Syntax ECMA-404, https://www.ecma-international.org/publications/standards/Ecma-404.htm

JSON Path RFC 6901, https://tools.ietf.org/html/rfc6901

JSON Schema, Draft 4, 6, or 7 and subsequent revisions, https://json-schema.org/

CommonMark, Version 0.28, https://spec.commonmark.org/0.28/

Media Type Specifications and Registration Procedures RFC6838, https://tools.ietf.org/html/rfc6838

1.4 Terminology

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

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

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

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

MIDI 1.0 Specification – Complete MIDI 1.0 Detailed Specification, Document Version 96.1, Third Edition

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

MIDI 2.0 – The MIDI environment that encompasses all of MIDI 1.0, MIDI-CI, Universal MIDI Packet, MIDI 2.0 Protocol, MIDI 2.0 Messages, and other extensions to MIDI as described in AMEI and MMA specifications.

Mcoded7 - Data transferred over SysEx must be restricted to 7-bit bytes. Mcoded7 is a method of converting 8-bit data into 7-bit data. See Section 3.3.1.

MIDI-CI – MIDI Capability Inquiry, a specification published by MMA and AMEI.

MMA – MIDI Manufacturers Association. Authority for MIDI specifications worldwide except Japan.

Pascal Case –The practice of writing phrases such that each word or abbreviation in the phrase begins with a capital letter, with no intervening spaces or punctuation between words. Pascal Case is sometimes called “UpperCamelCase”.

PE – Property Exchange, the subject of this specification, in which one device may access Property Data from another device.

Property – A JSON key:value pair used by Property Exchange.

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.

Property Data – A set of one or more Properties in a device which are accessible by Property Exchange. Contained in the Property Data field of a MIDI-CI Property Exchange message.

Resource – A defined Property Data with an associated inquiry for accessing the Property Data.

Property Exchange Overview

1.5 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 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 11) 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.

1.6 Steps to Use Property Exchange:

1. Perform a MIDI-CI Discovery Transaction (See the MIDI Capability Inquiry specification 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).

1.7 PE Message Format

Property Exchange messages are in the following format:

Value	Parameter	Notes
F0	System Exclusive Start	MIDI-CI Universal System Exclusive fields. SubID#2: determines the function of the message. Property Exchange messages = 0x30-3F Source MUID: The MUID of the device sending this message.
7E	Universal System Exclusive	
7F	7F = to/from whole MIDI Port	
0D	Universal System Exclusive Sub ID #1: MIDI-CI	
1 byte	Universal System Exclusive Sub ID #2:	
01	MIDI-CI Message Version/Format	
4 bytes	Source MUID (LSB first)	

4 bytes	Destination MUID (LSB first)	Destination MUID: The MUID of the device intended to receive this message. *See MIDI-CI Specification V1.1*
2 bytes	Number of Bytes for Header Data in this Chunk (nh)	Header Data, JSON compliant with defined restrictions. *See Section 4.1*
nh bytes	Header Data	
2 bytes	Number of Chunks in Data Set 0x0000 = Number of Chunks in Data Set is Unknown	These fields track the separate but associated Inquiry and Reply Messages *See MIDI-CI Specification V1.1* *Also See Sections 2.2 and 2.3*
2 bytes	Number of This Chunk (count starts from 0x0001)	
1 byte	Request ID	
2 bytes	Number of Bytes for Property Data in this Chunk (nd)	Property Data: Resource determines the format of the data: 1. JSON Compliant Data with defined restrictions 2. Other Data Data compression is optional *See Section 3.1*
nd bytes	Property Data	
F7	End Universal System Exclusive	End

1.8 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 1.7. 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 11 "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.

2. Property Exchange: MIDI-CI SysEx Messages

2.1 Property Exchange Inquiries and Replies

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

2.1.1 Inquiry: Get Property Data

The Inquiry: Get Property Data message is used to retrieve properties from the device.

2.1.2 Inquiry: Set Property Data

The Inquiry: Set Property Data message is used for the setting of information in a Resource.

2.1.3 All MIDI-CI Messages used for Property Exchange

Sub ID #2	Message Type
MIDI-CI Property Exchange Messages	
0x30	Inquiry: Property Exchange Capabilities
0x31	Reply to Property Exchange Capabilities
0x32	Inquiry: Has Property Data (Reserved)
0x33	Reply to Has Property Data (Reserved)
0x34	Inquiry: Get Property Data
0x35	Reply to Get Property Data
0x36	Inquiry: Set Property Data
0x37	Reply to Set Property Data
0x38	Subscription
0x39	Reply to Subscription
0x3A	Reserved
0x3B	Reserved
0x3C	Reserved
0x3D	Reserved
0x3E	Reserved
0x3F	Notify Message
MIDI-CI Management Messages	
0x70	Discovery
0x71	Reply to Discovery
0x72	Invalidate MUID
0x7F	NAK

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

2.2 Messages 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”.

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”.

Any message that contains Header Data only and does not contain any Property Data may not use the Chunking mechanism.
See the MIDI-CI specification version 1.1 for more details including values for special cases.

2.3 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 4.4.1) and the request will need to be retried.

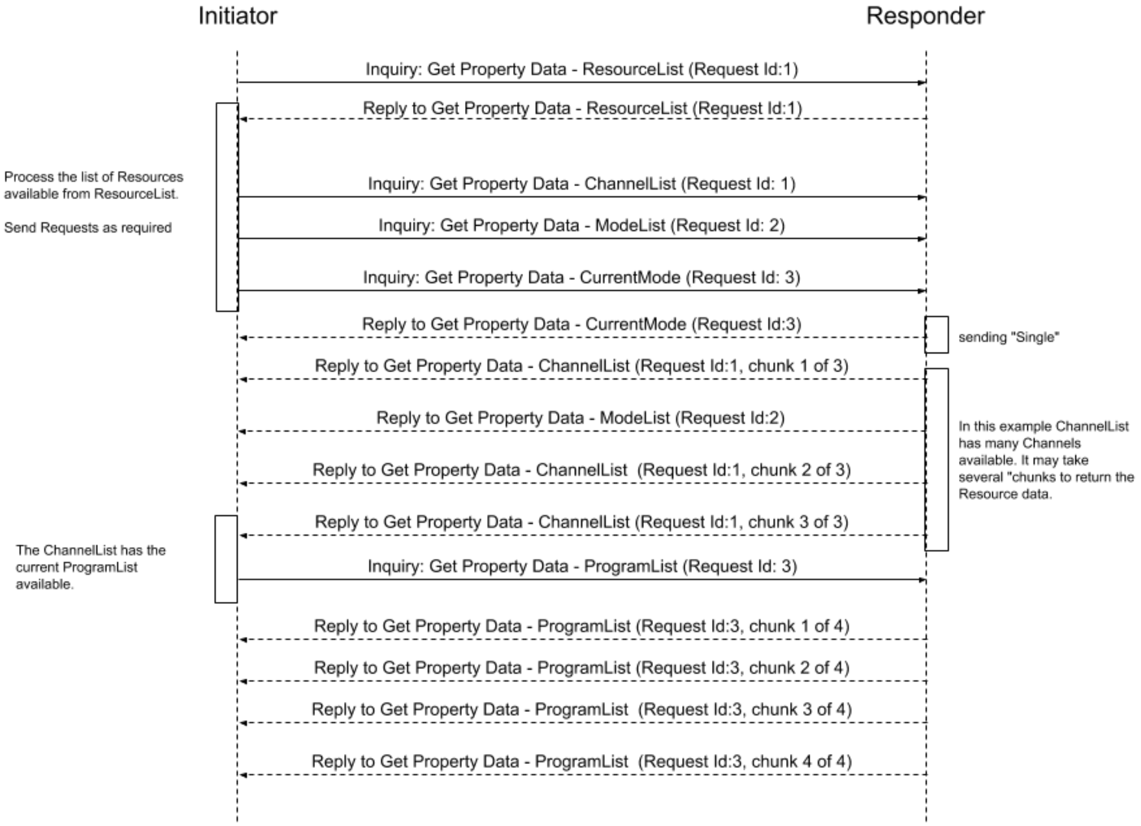
If an Initiator has exhausted the Number of Simultaneous Property Exchange Requests Supported, it should not issue any new request until a previous Transaction has completed.

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 9.3 for more details about timeouts.

Diagram: Example Using Request ID for Several Inquiries:


Initiator
Responder
Inquiry: Get Property Data - ResourceList (Request Id:1)
Reply to Get Property Data - ResourceList (Request Id:1)
Inquiry: Get Property Data - ChannelList (Request Id: 1)
Inquiry: Get Property Data - ModeList (Request Id: 2)
Inquiry: Get Property Data - CurrentMode (Request Id: 3)
Reply to Get Property Data - CurrentMode (Request Id:3)
sending "Single"
Reply to Get Property Data - ChannelList (Request Id:1, chunk 1 of 3)
Reply to Get Property Data - ModeList (Request Id:2)
Reply to Get Property Data - ChannelList (Request Id:1, chunk 2 of 3)
Reply to Get Property Data - ChannelList (Request Id:1, chunk 3 of 3)
Inquiry: Get Property Data - ProgramList (Request Id: 3)
Reply to Get Property Data - ProgramList (Request Id:3, chunk 1 of 4)
Reply to Get Property Data - ProgramList (Request Id:3, chunk 2 of 4)
Reply to Get Property Data - ProgramList (Request Id:3, chunk 3 of 4)
Reply to Get Property Data - ProgramList (Request Id:3, chunk 4 of 4)

Process the list of Resources available from ResourceList.

Send Requests as required

In this example ChannelList has many Channels available. It may take several "chunks to return the Resource data.

The ChannelList has the current ProgramList available.

See the MIDI Capability Inquiry (MIDI-CI), Version 1.1 for more details.

3. 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 3.2).
By default, the Property Data field is uncompressed. See Section 3.3 for details of optional compression and or encoding of the Property Data field.

3.1 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.
- 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.

3.1.1 7-Bit Encoding of JSON String Values

Due to the restrictions of MIDI 1.0 SysEx, JSON string values shall be restricted to 7-bit ASCII characters. All other characters shall be converted to UTF-16 and then escaped as defined in the JSON standard (ECMA404).
Unicode Characters Examples:

1. Beats♪ is converted to Beats\u266a.
2. 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
3. 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

3.1.2 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]
}
```

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]
}
```

3.1.3 CommonMark

CommonMark Version 0.28, 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 "ModeList" Resource, will refer to supporting CommonMark.

3.1.4 Manufacturer-specific Properties

Manufacturers may wish to include their own specific information inside MMA/AMEI defined Resources (see Section 5).
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 5.4).

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"
}
```

3.2 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 3.1, the associated Header Field shall include a `"mediaType"` Property to declare the format of the Property Data (see Section 4.5).

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 3.3 for details of Compression and Encoding.

3.3 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 4.5). 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 6.1).
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 4) and in ResourceList (See Section 6.1) to declare encoding types.

Property Value	**Description**
ASCII	Uncompressed, unencoded (must be 7-bit data)
Mcoded7	Encoded using Mcoded7 (See Section 3.3.1)
zlib+Mcoded7	Compressed using zlib (per RFC 1950) with Mcoded7 encoding.

3.3.1 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 3.1.1. Property Data which is not ASCII text as defined in Section 3.1.1 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:

> 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
>
> 0AAAaaaa 0BBBbbbb 0CCCcccc 0DDDdddd 0EEEeeee 0FFFffff 0GGGgggg
>
> 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 0AAAaaaa 0BBBbbbb 0CCCcccc

3.4 Order of Processing for Compression and/or Encoding

3.4.1 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 3.1.1.
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).

3.4.2 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 3.1.1.
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).

3.4.3 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.

3.4.4 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).

4. Header Data

4.1 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 3.1 and shall also conform to the additional set of rules and restrictions in Section 4.1.1.

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

4.1.1 JSON Header Data Property Additional Rules

Restrictions 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).

4.2 Common Header Properties in a Request

Property Key	Property Value Type	Description
resource	string (required, max 36 chars, "a-z" characters only)	This the targeted Resource.
resId	string (max 36 chars, "a-z", "0-9" or "_" characters only)	Resource ID, the identifier used to select the desired Payload Data entry. See Section 5.5.
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 3.3). The encodings supported by the Responder for each Resource are discovered by the use of ResourceList (See Section 11).

4.3 Common Header Properties in a Reply

Property Key	Property Value Type	Description
status	number (required, integer)	This the Status of the response. This is similar to HTTP Status codes. • Shall use the status list as described in Section 4.4.1
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 3.3). 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. The Initiator should use this cached result for subsequent requests for the same Resource Data.

4.4 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."}
```

4.4.1 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 9 for details.

200-299 Success Messages – Do Not Retry	
200	Success/Ok
202	Accepted – This means the data was 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 – This might be caused by SysEx data corruption or some other data stream issue.
343	Too Many Requests – Used when a 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 – It maybe that a particular Resource is not applicable at this time e.g. it may be in the wrong Mode.
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. For 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.

4.4.2 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 9.

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.

4.5 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 3.1, the Header Field shall include a "mediaType" Property to declare the format of the Property 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. This is often referred to as MIME type.

5. 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".

5.1 Resources Data Formats

By default, Resources shall be JSON compliant with the format rules and restrictions defined in Sections 3 and 4.

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 3.2 and 4.5).

5.2 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.

Example using the DeviceInfo Resource:

Initiator Sends Inquiry: Get Property Data Message

Header Data	`{"resource":"DeviceInfo"}`
Property Data	*none*

Responder Sends Reply to Get Property Data Message

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"
}
```

5.3 MMA/AMEI Defined Resources

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.

MMA/AMEI Resources should provide a unified approach to common applications. Following are some of the currently defined Resources:

- Device Information
- Channels in use
- Modes Available
- Programs available
- Controllers available

For descriptions of these, see Section 6.

Other standard Resources will be defined in future specifications by AMEI/MMA.

5.4 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 3.1.3).

5.5 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 11.

5.6 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"

5.6.1 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 10.

5.6.2 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 11.2.1.

Offset and limit Properties may be used in the Header Data of an inquiry that requests a List Resource. If pagination is used, both Properties shall be provided:

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.

Offset is 0 based. Record Count is 1 based.
Therefore, the first object in a reply is (offset value + 1).

All replies to a List Resource inquiry, when the Resource has declared "canPaginate" in the ResourceList, shall contain a "totalCount" Property so the Initiator can determine whether it wants/needs to use pagination. The inclusion of the "totalCount" property allows the Initiator to calculate the total number of pages available when using pagination.

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

Example of a List Resource Request and Response with Pagination

Initiator Sends Inquiry: Get Property Data Message

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

Responder Sends Reply to Get Property Data Message

Header Data	`{"status":200,"totalCount":128}`

Property Data

```
[
   {
      "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"}]
   }
]
```

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

5.7 Simple Property Resources

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

Example using the LocalOn Resource:

Initiator Sends Inquiry: Get Property Data Message

Header Data	`{"resource":"LocalOn"}`
Property Data	*none*

Responder Sends Reply to Get Property Data Message

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

Example using the CurrentMode Resource:

Initiator Sends Inquiry: Get Property Data Message

Header Data	`{"resource":"CurrentMode"}`
Property Data	*none*

Responder Sends Reply to Get Property Data Message

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

6. 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.

6.1 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 11.

6.2 Foundational Resources Defined in other Specifications:

It is strongly recommended that all Devices should support the following foundational Resources, defined in the Foundational and Basic Resources specification. These Resources provide core 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.

6.3 Other Resources Defined in other Specifications:

Other Resources are defined in other specifications published by AMEI/MMA.

Some examples include:

- **Modes** – A device may have a Performance mode, a Single Patch Mode, a GM mode or some other Mode. Each mode is generally structured differently in terms of the MIDI channels and Programs available.

- **Programs** – Program names and categories, with Bank MSB/LSB and Program Change messages for recalling each Program.
- **Active Controller Messages** – Discover the CC/NRPN/RPN and other Controller messages that the Device sends or receives on a Channel.
- **Get and Set Device State** – A snapshot of the device to restore at a later time.

7. 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 11).
>
> 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 11).

7.1 Full SET Example:

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

Initiator Sends Inquiry: Get Property Data Message

Header Data	`{"resource":"X-ProgramEdit","resId":"abcd"}`
Property Data	*none*

Responder Sends Reply to Get Property Data Message

Header Data	`{"status":200}`
Property Data	see below

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:

Initiator Sends Inquiry: Set Property Data Message

Header Data	`{"resource":"X-ProgramEdit","resId":"abcd"}`

Property Data

```
{
   "name": "Violin 2",
   "lfoSpeed": 10,
   "lfoWaveform": "sine",
   "pitchEnvelope": {
      "rates": [30,20,90,47],
      "levels": [100,90,80,70]
   }
}
```

Responder Sends Reply to Set Property Data Message

Header Data	`{"status":200}`
Property Data	*none*

7.2 Partial SET Examples:

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

Initiator Sends Inquiry: Get Property Data Message

Header Data	`{"resource":"X-ProgramEdit","resId":"abcd"}`
Property Data	*none*

Responder Sends Reply to Get Property Data Message

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 makes a minor change to the LFO speed and only wants to send this change.
The Initiator uses an Inquiry: Set Property Data message:

Initiator Sends Inquiry: Set Property Data Message

Header Data	`{"resource":"X-ProgramEdit","resId":"abcd","setPartial":true}`

Property Data	`{`
	`"/lfoSpeed":10`
	`}`

Note the addition of the "setPartial":true Property.

Responder Sends Reply to Set Property Data Message

Header Data	`{"status":200}`
Property Data	*none*

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

Initiator Sends Inquiry: Set Property Data Message

Header Data	`{"resource":"X-ProgramEdit","resId":"abcd","setPartial":true}`

Property Data	`{`
	`"/pitchEnvelope/rates/0":80,`
	`"/pitchEnvelope/levels/0":60`
	`}`

Responder Sends Reply to Set Property Data Message

Header Data	`{"status":200}`
Property Data	*none*

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

Initiator Sends Inquiry: Get Property Data Message

Header Data	`{"resource":"X-ProgramEdit","resId":"abcd"}`
Property Data	*none*

Responder Sends Reply to Get Property Data Message

Header Data	`{"status":200}`

Property Data	

```
{
   "name": "PIANO 4",
   "lfoSpeed": 10,
   "lfoWaveform": "triangle",
   "pitchEnvelope": {
      "rates": [80,67,95,60],
      "levels": [60,50,50,50]
   }
}
```

8. 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 11.
2. The Initiator may subscribe to 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 8.1).
>
> > 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:
> 1. 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.
> 2. 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 10), in some case 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.

8.1 Extra Request Header Properties for Subscriptions

Property Key	Property Value Type	Description
subscribeId	string (required, max 8 chars, "a-z", "0-9" or "_" characters only)	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. 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.
command	enum (required)	The command determines the intent of the Subscription MIDI-CI message **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. **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. **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. **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.

		The Initiator shall not use this but instead shall perform an Inquiry: Get Property Data to get the refreshed Property Data from the Responder. **End Subscription ("end") - Initiator or Responder** Either device can end the Subscription. There is no Property Data in this message.

8.1.1 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.

8.2 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.

8.3 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 7).

8.3.1 Example: Subscription to a Simple Property Resource

Step 1:

Initiator Sends Subscription Message

Header Data	`{"command":"start","resource":"CurrentMode"}`
Property Data	*none*

Responder Sends Reply to Get Subscription Message

Header Data	`{"status":200,"subscribeId":"sub32847623"}`
Property Data	*none*

Step 2:

Responder Sends Subscription Message using a "full" command

Header Data	`{"command":"`**`full`**`","subscribeId":"sub32847623"}`
Property Data	`"multichannel"`

Initiator Sends Reply to Subscription Message

Header Data	`{"status":200}`
Property Data	*none*

8.4 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:

Initiator Sends Inquiry: Get Property Data Message

Header Data	`{"resource":"X-ProgramEdit","resId":"abcd"}`
Property Data	*none*

Responder Sends Reply to Get Property Data Message

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

```
{
   "name": "PIANO 4",
   "lfoSpeed": 10,
   "lfoWaveform": "triangle",
   "pitchEnvelope": {
      "rates": [80,67,95,60],
      "levels": [60,50,50,50]
   }
}
```

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:

Initiator Sends Subscription Message

Header Data	`{"command":"start","resource":"X-ProgramEdit","resId":"abcd"}`
Property Data	*none*

Responder Sends Reply to Subscription Message

Header Data	`{"status":200,"subscribeId":"sub138047"}`
Property Data	*none*

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:

Responder Sends Subscription Message

Header Data	`{"command":"partial","subscribeId":"sub138047"}`
Property Data	

```
{
   "/lfoWaveform":"square"
}
```

Initiator Sends Reply to Subscription Message

Header Data	`{"status":200}`
Property Data	*none*

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:

Responder Sends Subscription Message

Header Data	`{"command":"notify","subscribeId":"sub138047"}`
Property Data	*none*

Initiator Sends Reply to Subscription Message

Header Data	`{"status":200}`
Property Data	*none*

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

Initiator Sends Inquiry: Get Property Data Message

Header Data	`{"resource":"X-ProgramEdit","resId":"abcd"}`
Property Data	*none*

Responder Sends Reply to Get Property Data Message

Header Data	`{"status":200}`

Property Data

```
{
   "name": "PIANO 4",
   "lfoSpeed": 50,
   "lfoWaveform": "saw",
   "pitchEnvelope": {
      "rates": [25,46,17,0],
      "levels": [80,10,36,94]
   }
}
```

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

Initiator Sends Inquiry: Set Property Data Message

Header Data	`{"resource":"X-ProgramEdit","resId":"abcd","setPartial":true}`

Property Data

```
{
   "/name":"Broken Piano"
}
```

Responder Sends Reply to Set Property Data Message

Header Data	`{"status":200}`
Property Data	*none*

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.

Responder Sends Subscription Message

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

Initiator Sends Reply to Subscription Message

Header Data	`{"status":200}`
Property Data	*none*

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

Initiator Subscription Message

Header Data	`{"command":"end","subscribeId":"sub138047"}`
Property Data	*none*

Responder Sends Reply to Subscription Message

Header Data	`{"status":200}`
Property Data	*none*

8.5 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).

9. Notify Message

Property Exchange provides mechanisms to Terminate an Inquiry, perform a Timeout Wait, or report a Timeout condition by way of a Notify message.

A Notify message may be sent by the Initiator or the Receiver. There is no reply message associated with any Notify message.

A Notify message shall use the Request Id of the message in progress which it is currently referring too.

9.1.1 Table of status codes used in a Notify message

Notify messages shall only use the Status codes that are listed in this table and shall not use any other codes.

100-199 Notification Messages	
100	Timeout Wait
144	Terminate Inquiry
400-499 Client Error Responses	
408	Timeout Has Occurred

9.2 Notify: Terminate Inquiry

Either device can 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.

Initiator or Responder Sends Notify Message

Header Data	`{"status":144}`
Property Data	*none*

9.3 Notify: Timeout Messages

A Responder shall send a Reply to any Inquiry within a 3-seconds timeout period if it expects the Initiator to process the reply.

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.

9.3.1 Notify: Timeout Wait

If a device 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 100 to keep the current inquiry active.

Example 1:

If a Responder receives an Inquiry and is processing that request, and if the Responder cannot begin the reply before the timeout period, the Responder may send a Notify message with a "status" Property with a value of 100.

Example 2:

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.

Responder Sends Notify Message

Header Data	`{"status":100}`
Property Data	*none*

9.3.2 Notify: Timeout Has Occurred

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.

Initiator Sends Notify Message

Header Data	`{"status":408}`
Property Data	*none*

Note: MIDI-CI also provides a NAK that may be used with Property Exchange when it is not possible to use the Notify message and not possible to return a more informative error report as defined in Section 4.4. See the MIDI-CI specification for details of the NAK.

10. 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.

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 5.5.
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: 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-".

If the Resource has a resId then it must be used. If the Resource is not defined to have a resId, then the Resource may be linked using the Resource name only.

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.

10.1.1 Example of Using Links With a "resId" Property

Property Data

```
[
   {
      "title":"Ch.1",
      "channel":1,
      "bankPC":[0,0,0],
      "programTitle":"Piano",
      "links":[
         {"resource":"ExampleCMList", "resId":"singch1"},
         {"resource":"X-ProgramEdit", "resId":"singch1"}
      ]
   }
]
```

Note: "ExampleCMList" is a hypothetical Resource, representing an example of a potential list of controller messages. It is used in a number of examples in this document to show how Resources function. The format in this document should not be considered to be definitive for such a Resource.

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.

Initiator Sends an Inquiry: Get Property Data Message

Header Data	`{"resource":"ExampleCMList","resId":"singch1"}`
Property Data	*none*

10.1.2 Example of Using Links Without a "resId" Property

Property Data

```
{
   "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"}
   ]
}
```

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	☑

The Tempo and LocalOn Resources in this example are Simple Property Resources (See Section 5.7) and are shown as editable in line.

10.1.3 Example of Using a "role" Property of a Link

This example uses a hypothetical "role" called "actionButton", which might be defined to bring up a button for user selection. The title displayed on the button would be "Initialize Program".

Property Data

```
[
   {
      "channel": 1,
      "bankPC": [0,64,1],
      "title": "My Edited Program",
      "links": [
         {
            "resource": "X-Init",
            "resId": "singch1",
            "role": "actionButton",
            "title": "Initialize Program"
         }
      ]
   }
]
```

When the user selects the button, the Initiator performs an Inquiry Get Property Data selecting the linked Resource.

Initiator Sends an Inquiry: Get Property Data Message

Header Data	`{"resource":"X-Init","resId":"singch1"}`
Property Data	*none*

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

Responder Sends Reply to Get Property Data Message

Header Data	`{"status":200}`
Property Data	`"Success"`

The Initiator displays "Success" to the user.

10.1.4 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.

11. 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.

11.1 Request ResourceList using Inquiry: Get Property Data

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

Initiator Sends Inquiry: Get Property Data Message

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

11.2 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:

Property Key	Property Value Type	Description
resource	string (required, max 36 chars)	This is the Resource name. Manufacturer custom Resource names always start with "X-". See Section 5.4.
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.
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 7.
canSubscribe	boolean (default: false)	Declares whether subscription can be used on this Resource. See Section 8.
requireResId	boolean (default:false)	Inquiry shall contain resId Property in the Header Data. See Section 5.5.
mediaTypes	array of mediaType strings (default: [application/json])	Media Types as defined by RFC-6838. 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 3.2.
encodings	array of strings (default: ["ASCII"])	This is the list of encodings the Resource supports. See Section 3.3 for the list of acceptable values.
schema	JSON Schema Object	Usually only used with Manufacturer Specific Resources*. Please see Section 11.4 for more information

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

11.2.1 Additional Properties for List Resources

Property Key	Property Value Type	Description
canPaginate	boolean (default:false)	If the Resource returns an array of objects, then the Resource shall declare whether pagination is supported. See Section 5.6.2.
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 11.5 for more information.

Example Transaction for ResourceList:

Initiator Sends Inquiry: Get Property Data Message

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

Responder Sends Reply to Get Property Data Message

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

```
[
   {"resource": "DeviceInfo"},
   {"resource": "ChannelList","canSubscribe":true},
   {"resource": "ProgramList"},
   {"resource": "ExampleCMList"},
   {
      "resource": "BufferState",
      "canSet": "full",
      "mediaTypes": ["application/octet-stream"],
      "encodings": ["MCoded7"]
   }
]
```

11.3 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.

11.3.1 Example: ResourceList Object for the ChannelList Resource

This example shows declaring all the default values:

Property Data

```
[
   {
   "resource":"ChannelList",
   "canGet":true,
   "canSet":"no",
   "canSubscribe":false,
   "requireResId":false,
   "canPaginate":false,
   "schema":{
      "type":"array",
      "title":"Channel List",
      "$ref":"http://midi.org/midi-ci/pe/schema/channellist.json"
      },
   "columns":[
      {"property":"title","title":"Title"},
      {"property":"deviceBasicChannel","title":"Device Basic Channel"},
      {"property":"umpGroup","title":"UMP Group"},
      {"property":"channel","title":"MIDI Channel"},
      {"property":"programTitle","title":"Program Title"},
      {"link":"ProgramList","title":"Program List"}
      ]
   }
]
```

This example shows a minimized object using all the default values:

Property Data

```
[
 {"resource":"ChannelList"}
]
```

This example shows a minimized object using the default values for some Properties and declaring non-default values for other Properties:

Property Data

```
[
   {
      "resource":"ChannelList",
      "canSubscribe":true,
      "columns":[
         {"property":"channel","title":"MIDI Channel"},
         {"property":"title","title":"Title"},
         {"link":"ProgramList","title":"Program List"},
         {"link":"ExampleCMList","title":"Channel Controllers"}
      ]
   }
]
```

11.4 Schema Property for Manufacturer Specific JSON Resources

AMEI/MMA standardized Resources do not require exchange of a JSON Schema.

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.

11.4.1 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.

11.4.1.1 Example Resource List Object with a "schema" Property:

Property Data

```
[
   {
      "resource":"X-WIFIOn",
      "canSet":"full",
      "schema":{
         "title":"WIFI Enabled",
         "description":"Turn WIFI on or Off",
         "type":"boolean"
      }
   }
]
```

11.4.2 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>"

11.4.2.1 Example ResourceList Object using a JSON Schema reference:

Property Data

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

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.*

11.4.2.2 Example of a JSON Schema reference:

Initiator Sends an Inquiry: Get Property Data Message

Header Data	`{"resource":"JSONSchema","resId":"globalSchema"}`
Property Data	*none*

Responder Sends Reply to Get Property Data Message

Header Data	`{"status":200}`

Property Data

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "properties": {
    "deviceName": {
      "type": "string"
    },
    "audioOut": {
      "type": "string",
      "enum": ["USB","line"]
    },
    "cvPitch": {
      "type": "string",
      "enum": ["v/oct","hz/oct"]
    },
    "gate": {
      "type": "string",
      "enum": ["s-trig","v-trig"]
    }
  }
}
```

11.5 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".

11.5.1 Example ResourceList Object Record Containing a "columns" Property

Property Data

```
[
   {
      "resource": "ChannelList",
      "canSubscribe": true,
      "columns":[
         {"property": "channel", "title": "MIDI Channel"},
         {"property": "programTitle", "title": "Program Name"},
         {"link":"ExampleCMList", "title": "Controllers"}
      ]
   }
]
```

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 ExampleCMList.

MIDI Channel	Program Name	Controllers

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

ChannelList Result for List Object Above

Property Data

```
[
   {
      "title":"Ch.1",
      "channel":1,
      "bankPC":[0,0,0],
      "programTitle":"Piano",
      "links":[
         {"resource":"ExampleCMList", "resId":"singch1"}
      ]
   },
   {
      "title":"Ch.13",
      "channel":13,
      "bankPC":[0,0,1],
      "programTitle":"Synth Lead 2",
      "links":[
         {"resource":"ExampleCMList", "resId":"singch2"}
      ]
   }
]
```

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"
"ExampleCMList"

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	Controllers
13	Synth Lead 2	Controllers

11.6 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.

11.6.1 Example Non-JSON Resource:

Property Data	

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

11.7 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.

Property Data	

```
[
   {"resource":"JSONSchema","encodings":["ASCII","zlib+Mcoded7"]},
   {"resource": "DeviceInfo"},
   {
      "resource": "ChannelList",
      "canSubscribe": true,
      "columns": [
        {"property": "perfChannel"},
        {"property": "channel"},
        {"property": "name"},
        {"link": "ProgramList"},
        {"link": "ExampleCMList"},
        {"link": "X-Patch"}
     ]
   },
   {"resource": "ProgramList"},
   {"resource": "ExampleCMList"},
   {"resource": "ModeList"},
   {
      "resource": "CurrentMode",
      "canSet": "full",
      "canSubscribe": true
   },
   {
      "resource": "RawBuffer",
      "canSet": "full",
      "mediaTypes": ["application/octet-stream"],
      "encodings": ["MCoded7"]
   },
   {
      "resource": "X-Tempo",
      "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"
      }
   }
]
```

Appendix A – Example Minimal Implementation

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.

The effect pedal supports 4 inquiries:

- ResourceList
- DeviceInfo
- ChannelList
- ExampleCMList

The effect pedal has a simple parser to recognize only these 4 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:

Action 1:

Initiator Sends Inquiry: Get Property Data Message

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

Responder Sends Reply to Get: Property Data Message

Header Data	`{"status":200}`

Property Data	`[`
	`{"resource":"DeviceInfo"},`
	`{"resource":"ChannelList"},`
	`{"resource":"ExampleCMList"}`
	`]`

Initiator now knows which Resources the Responder supports and can make those inquiries.

Action 2:

Initiator Sends Inquiry: Get Property Data Message

Header Data	`{"resource":"DeviceInfo"}`
Property Data	*none*

Responder Sends Reply to Get Property Data Message

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"
}
```

Action 3:

Initiator Sends Inquiry: Get Property Data Message

Header Data	`{"resource":"ChannelList"}`
Property Data	*none*

Responder Sends Reply to Get Property Data Message

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

```
[
   {
      "title":"Simple Pedal",
      "channel":1,
      "links":[
         {"resource":"ExampleCMList","resId":"all"}
      ]
   }
]
```

Action 4:

Initiator Sends Inquiry: Get Property Data Message

Header Data	`{"resource":"ExampleCMList","resId":"all"}`
Property Data	*none*

Responder Sends Reply to Get Property Data Message

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

Property Data

```
[
   {
      "name":"Level",
      "priority":1,
      "control":"CC",
      "controlNo":[75],
      "default":100,
      "value":100
   },
   {
      "name":"Tone",
      "priority":1,
      "control":"CC",
      "controlNo":[76],
      "default":64,
      "value":80
   },
   {
      "name":"Distortion",
      "priority":1,
      "control":"CC",
      "controlNo":[77],
      "default":64,
      "value":127
   }
]
```