Common System Exclusive Commands TNG

1 Overview

This document details the MIDI system exclusive messages used in some iConnectivity products. A list of products that support these messages is provided in the Product IDs section near the end of this document. An overview of the message format and a general guide for using these messages is provided in this introductory chapter.

1.1 Message Classes, Data Classes, Data Blocks and Commands

Messages classes define specific operations that can be performed on a device. For example, getting values from a device, changing values in a device, or changing a device's operating mode. Data classes are used to define groups of related values. For example, hardware configuration values, MIDI configuration values, or the current audio mixer values. Combining a message class with a data class creates a specific message type. For example, getting hardware configuration values, changing MIDI configuration values, or getting the current audio mixer values.

Data blocks are used as building material to construct a message. Different types of data blocks are used to denote a functional block type. For example, one kind of data block contains a list of parameters to retrieve from a device, another kind of data block contains the parameter values.

All messages begin with a common header and end with a common footer. Between the header and the footer is the message content comprising the message type (message class and data class) and a number of data blocks. For example, to retrieve a list of hardware values for a specific hardware port the following message would be used:

Header
Message Type = retrieve hardware values
Data Block #1 = hardware port ID
Data Block #2 = a list of hardware parameter IDs
Footer

After receiving the above message, the device would respond with the following message to return the hardware values:

Header
Message Type = return hardware values
Data Block #1 = hardware port ID
Data Block #2 = a list of hardware parameter IDs and their associated values
Footer

Commands are a type of message class that instructs a device to perform a function. Commands do not use data classes but they use data blocks. Examples include rebooting a device, saving presets in RAM to non-volatile storage, or starting a bulk data transfer operation.

1.2 Parameters

The bulk of this document comprises data class parameter definitions. All data that can be read and written is accessed via a parameter ID unique to each data class. Many parameters are atomized down to a single value but in some cases bitmaps and structures are used to group several values together when it is convenient to do so. In some cases, structures use a sub-ID field to denote each member of the structure.

Parameters have the following attributes:

Read-Only or Read-Write

Read-Only parameters are those that cannot be modified by a user. Some read-only parameters have a constant value meaning that they never change (denoted as RC parameters). For example, the number of Ethernet ports on a device, or the MAC address associated with each Ethernet port. The remaining read-only parameters are dynamic meaning that the value can change as the device is operating (denoted as RD parameters). For example the dynamic IP address assigned to an Ethernet port, or the session name that initiated an RTP-MIDI session with the device. A host application need only read RC parameters once during a communication session but RD parameters should be refreshed as necessary (depending on the needs of the host application).

Read-Write parameters are those that can be modified by a user. Some read-write parameters require that the device be restarted before the new value is used (denoted as WB parameters). For example, the number of audio channels to use on a USB device port, or the audio sample rate and audio synchronization method used. However, most read-write parameters support real-time modification and these can be modified at any time and take effect instantly (denoted as WN parameters). For example, MIDI filtering, audio routing, or audio mixer settings. A host application should refresh all read-write parameters regularly during a communication session (depending on the needs of the host application).

Global or Preset

Some products support storing multiple configurations, or presets, “in the box” and allow changing between these presets at runtime without having to reboot the device. For these products, the parameters that can be specified on a per-preset level are denoted as preset (P) parameters. Parameters that apply to all presets (or for products that don’t support presets) are denoted as global (G) parameters. For example, most parameters related to MIDI port configuration (e.g. filtering, routing) are preset parameters whereas the configuration of an Ethernet port (e.g. IP address) is a global parameter.

Scene Applicable

Some products support multiple scenes within a preset. Typically, scenes are used to manage audio and MIDI routing for failover systems and employ special rules to deal with scene management and selection. Parameters that can be specified on a per-scene level are denoted as scene (S) parameters. Parameters that cannot be specified on a per-scene level are denoted as not-scene (T) parameters.

Parameter Attribute Notation

The following notation is used to summarize parameter attributes:

RD = read-only dynamic parameter
RC = read-only constant parameter
WN = writeable normal parameter
WB = writeable reboot parameter
P = preset parameter
G = global parameter
S = scene parameter
T = not a scene parameter

Examples:

RCGT = global read-only constant parameter (no scene support)
WBPT = preset writeable parameter that requires a reboot to take effect (no scene support)

RDPT = preset read-only dynamic parameter (no scene support)
WNPS = preset writeable normal scene parameter

1.3 Memory Model

1.3.1 Non-Volatile Storage (FLASH)

The memory model for global, preset, and scene parameters is shown below for non-volatile storage (for a device that supports two scenes per preset and 4 presets). All WB parameters are in the global section so that changing presets does not require rebooting a device. At runtime, the globals and one preset are copied to the work area in RAM where they can be manipulated by system exclusive commands and then saved to non-volatile storage.

Name	Description
Global Parameters	RCGT/RDGT/WNGT parameters. WBGT parameters.
Preset #1 Parameters	RCPT/RDPT/WNPT parameters for preset #1 including: 1. RCPS/RDPS/WNPS parameters for scene #1 2. RCPS/RDPS/WNPS parameters for scene #2
Preset #2 Parameters	RCPT/RDPT/WNPT parameters for preset #2 including: 1. RCPS/RDPS/WNPS parameters for scene #1 2. RCPS/RDPS/WNPS parameters for scene #2
Preset #3 Parameters	RCPT/RDPT/WNPT parameters for preset #3 including: 1. RCPS/RDPS/WNPS parameters for scene #1 2. RCPS/RDPS/WNPS parameters for scene #2
Preset #4 Parameters	RCPT/RDPT/WNPT parameters for preset #4 including: 1. RCPS/RDPS/WNPS parameters for scene #1 2. RCPS/RDPS/WNPS parameters for scene #2

1.3.2 Volatile Storage (RAM)

The memory model for global, preset, and scene parameters is shown below for volatile storage (for a device that supports two scenes per presets, N presets, and one shadow area). Devices support a work area that always contains the currently active global and preset parameters (I.e. all the parameters that the device is currently using). Device may also contain additional shadow areas that can be used by software to manipulate global and preset parameters without modifying those in the device's work area. Global and preset parameters can be moved between volatile (work and shadow areas) and non-volatile storage.

Area	Name	Description
Work	Active	RCGT/RDGT/WNGT parameters.

Area	Name	Description
	Global Parameters	WBGT parameters.
Work	Active Preset Parameters	RCPT/RDPT/WNPT parameters for active preset including: 1. RCPS/RDPS/WNPS parameters for scene #1 2. RCPS/RDPS/WNPS parameters for scene #2
Shadow	Global Parameters	RCGT/RDGT/WNGT parameters. WBGT parameters.
Shadow	Preset Parameters	RCPT/RDPT/WNPT parameters for any preset including: 1. RCPS/RDPS/WNPS parameters for scene #1 2. RCPS/RDPS/WNPS parameters for scene #2

1.4 Session Management

A host communicates with a device using the MIDI system exclusive commands described in this document. Communication sessions are stateless and a device can support multiple concurrent sessions. The host needs to manage the session to gather information from the device. A host should not send multiple messages to a device and then wait for multiple responses; the host should send one message then wait for a response from the device before sending the next message (i.e. use handshaking). Guidelines for session management are given below.

Device Discovery

A host application typically only has access to a list of available MIDI ports on a computer, not to a list of connected MIDI devices. A host should use device discovery to find the best MIDI port for communication and to set communication parameters. A suggested process is described below.

1. Host creates a random 28-bit value for the session ID (used in every message header).
2. Host sends HstSesnVal messages to every MIDI output port. Each message has a unique transaction ID. Host also sets the HstInSizeMax parameter for each message. SNUM portion of Device ID should be 0. PID can be 0 or a specific product ID.
3. Device returns a DevSesnVal message for each HstSesnVal message it receives along with Device ID and connection parameters. A device only returns a DevSesnVal message if its Device ID matches the Device ID in the HstSesnVal message (wildcards are supported).
4. Host receives DevSesnVal messages on MIDI input ports and uses the session ID and transaction ID to find the matching HstSesnVal message to pair up MIDI input ports with MIDI output ports.
5. Host will likely find numerous MIDI input/output ports that are connected to the same device (use Device ID in DevSesnVal message to identify the device). Host uses DevOpMode and DevMIDIPortInfo to determine if the device is in the correct operating mode for the application and to determine which set of MIDI input/output ports to use. Host selects one set of MIDI input/output ports to use for the remainder of the session.
6. Host saves the value for DevInSizeMax (maximum length of messages sent to this device) and verifies that DevOutSizeMax is less than or equal to HstInSizeMax. If DevOutSizeMax is larger than HstInSizeMax, host sends another HstSesnVal message to the device and the device returns another DevSesnVal message. Repeat this step as necessary until DevOutSizeMax is less than or equal to HstInSizeMax.
7. Host and device are now ready for host to gather information from the device.

Get Device Info

Once device discovery has completed, the host gathers information from the device beginning with the DeviceInfo data class. A suggested process is described below.

1. Host sends a GetParmDef:DeviceInfo message to the device to get a list of supported DeviceInfo parameter IDs.
2. Device returns a RetParmDef:DeviceInfo message to the host containing a list of supported DeviceInfo parameter IDs.
3. Host sends a GetParmVal:DeviceInfo message to the device to retrieve parameter values that are of interest to the host. Host and device should use handshaking (to send several messages repeat steps 3-5 as needed).
4. Device returns a RetParmVal:DeviceInfo message to the host containing parameter values.
5. Host uses the DeviceInfo parameter values to determine the next set of DeviceInfo parameters to retrieve. Repeat steps 3-5 as needed until all DeviceInfo is retrieved.

The host can then repeat the above procedure using the DeviceFeature data class if there are any device features of interest to the host.

Get Hardware Info

Once DeviceInfo has been gathered, the host can gather hardware information. A suggested process is described below.

1. Host sends a GetParmDef:HardwareInfo message to the device to get a list of supported HardwareInfo parameter IDs.
2. Device returns a RetParmDef:HardwareInfo message to the host containing a list of supported HardwareInfo parameter IDs.
3. Host sends a GetParmVal:HardwareInfo message to the device to retrieve parameter values that are of interest to the host. Host and device should use handshaking (to send several messages repeat steps 3-5 as needed).
4. Device returns a RetParmVal:HardwareInfo message to the host containing parameter values.
5. Host uses the HardwareInfo parameter values to determine the next set of HardwareInfo parameters to retrieve. Repeat steps 3-5 as needed until all HardwareInfo is retrieved.

Note that steps 3-5 need to be repeated for each hardware port (i.e. each USB device port, each control port, each Ethernet port, etc.)

Get MIDI Info

The same process described for getting hardware info can be used for MIDI info once hardware info has been retrieved. Begin with the MIDIInfo data class to get common MIDI parameters then use the MIDIPortInfo data class to get port specific MIDI parameters and finally the MIDIFeature data class.

Change Parameter Values

Once all relevant parameters have been retrieved from the device, the host can then change parameter values using the process described below.

1. Host sends a SetParmVal message to the device containing new values for one or more parameters.
2. Device returns an Ack message to the host indicating if the parameter values were accepted or not.

Get Commands

Supported commands can be gathered from a device at any time using the process described below.

1. Host sends GetCmdDef message (no data class) to the device.
2. Device returns a RetCmdDef message (no data class) to the host containing a list of supported command IDs and command arguments.

Execute Commands

To execute a command, a host uses the process described below.

1. Host sends a SetCmdVal message (no data class) to the device.
2. Device returns an Ack message to the host indicating if the command was successful or not.

Register For Notifications

Parameter values that are not read-only constant (RC) can change at any time. A host can continuously poll a device to check for changes but for some host applications it may be more efficient for the host to ask the device to automatically inform the host whenever parameter values change. A suggested process is described below.

1. Host sends a SetCmdVal:Notification message to the device to register for parameter notifications.
2. Device returns an Ack message to the host indicating if the registration was successful or not.
3. Device sends NotParmVal messages to the host whenever parameter values change that the host did not instigate (i.e. from another session, from user interaction, from some automatic process, etc.) Host does not respond to the NotParmVal messages.
4. Host must send a system exclusive message (any message) to the device before a timeout interval otherwise the device will assume that the host is not listening and will stop sending NotParmVal messages.
5. Host sends a SetCmdVal:Notification message to the device to unregister for parameter notifications when the host no longer wants to receive NotParmVal messages.
6. Device returns an Ack message to the host indicating if the unregistration was successful or not.

See the section on Notification commands for more detailed information.

2 Message Format

Header:

0xF0	- 1 byte, start of system exclusive
0x00, 0x01, 0x73	- 3 bytes, iConnectivity's manufacturer ID code
0x7D	- 1 byte, system exclusive class code

Body:

Device ID	- 7 bytes (2 bytes for product ID, 5 bytes for serial number)
Session ID	- 4 bytes (28 bit value)
Transaction ID	- 4 bytes (28 bit value)
Message Length	- 2 bytes, number of bytes in message content (14 bit value)
Message Content	- length varies
Checksum	- 1 byte

Footer:

0xF7	- 1 byte, end of system exclusive

Data Format

All fields are big-endian (most significant byte occurs first, least significant byte occurs last). All fields which accept values greater than 0x7F must have the value split across multiple bytes in order to be compatible with MIDI system exclusive format. The least significant 7 bits of the value are contained in the 7 least significant bits of the last byte. The next 7 bits are contained in the 7 least significant bits of the last-1 byte, etc. Here are some examples:

Value	First Byte	Second Byte
0x0003	0x00	0x03
0x007F	0x00	0x7F
0x0080	0x01	0x00
0x0081	0x01	0x01
0x1234	0x24	0x34
0x2CA5	0x59	0x25

See the section *Frequently Used Data Conversions* for more information on commonly used data encodings used throughout this document.

Device Identifier

The first 2 bytes of Device ID are product ID (PID) a 14-bit value split across 2 bytes (14x2 format). The remaining 5 bytes are serial number (SNUM) a 32-bit value split across 5 bytes (32x5 format). PID and SNUM are formatted so that each byte is 0x7F or less as explained in the Data Format section. Here are some examples:

PID	SNUM	Device Identifier
0x0001	0x00000001	0x00 0x01 0x00 0x00 0x00 0x00 0x01
0x0001	0x00000080	0x00 0x01 0x00 0x00 0x00 0x01 0x00
0x00A0	0x00000080	0x01 0x02 0x00 0x00 0x00 0x01 0x00
0x0ABC	0x12345678	0x15 0x3C 0x01 0x11 0x51 0x2C 0x78

Some messages allow PID and SNUM to be zero which indicates a wildcard as follows:

PID	SNUM	Meaning
= 0	= 0	All devices.

PID	SNUM	Meaning
= 0	> 0	All devices with the specified serial number.
> 0	= 0	All devices with the specified product ID.
> 0	> 0	A device that matches the specified product ID and serial number.

Session ID

Session ID can be used to uniquely identify messages for a particular host or application. Typically, the host sends a query to one or more devices, each device then sends a response back to the host. A device will return the same session ID in the response as was sent in the query. Session IDs should be a random number generated by the host. Session ID is a 28-bit value split across 4 bytes (28x4 format). Each byte is 0x7F or less as explained in the Data Format section.

Transaction ID

Transaction ID can be used to uniquely identify messages. Typically, the host sends a query message to one or more devices, each device then sends a response message back to the host. A device will return the same transaction ID in the response as was sent in the query. It is not required that transaction IDs increment for every message sent by a host, they can always be 0 or some other value, it is up to the host whether or not it wants to use transaction ID for managing messages. Transaction ID is a 28-bit value split across 4 bytes (28x4 format). Each byte is 0x7F or less as explained in the Data Format section.

Message Length

The number of bytes that follow in the message content section. Message length is a 14-bit value split across 2 bytes (16x2 format). Each byte is 0x7F or less as explained in the Data Format section.

Message Content

The message content field begins with a two-byte header. The first byte of the header indicates the Message Class. The second byte of the header indicates the Data Class. These two bytes combined identify the message type. The remainder of the message content field depends on the message type.

Minimum length of the message content field is two bytes. The host can send a Ping message to a device by using 0 for the Message Length field and leaving the Message Content field empty. The device will respond with a similar message if it supports this protocol. This should not be used as means to discover devices; the HstSesnVal message class should be used for device discovery.

Checksum

The checksum field contains the 2s complement of the sum of all bytes in the body, excluding the checksum byte. In other words, summing all the bytes in the body should result in 0x00. Note that the checksum byte must be 0x7F or less to be compatible with MIDI system exclusive format (i.e. if the calculated checksum value is 0x83 then the checksum value used in the system exclusive message should be 0x03).

Example: minimal length message with no message content

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x02                     // message length (2)
0x02, 0x01                     // message class, data class
0xxx                           // checksum
0xF7                           // footer
```

Example: ping message with zero message length (all products, all serial numbers)

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x00                     // product ID
0x00, 0x00, 0x00, 0x00, 0x00   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x00                     // message length (0)
0x00                           // checksum
0xF7                           // footer
```

Example: message with content (1 data block)

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x00                     // product ID (any)
0x00, 0x00, 0x00, 0x00, 0x00   // serial number (any)
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x0A                     // message length (10)
0x01, 0x01                     // message class, data class
0x01                           // NumDataBlock: (1)
0x07, 0x03                     // data block #1: size (7), type (ParmVal)
0x01                           // NumParmValBlock: (1)
0x04                           // ParmVal #1: ParmSize: (4)
0x01                           // ParmVal #1: ParmID: HstInSizeMax (0x01)
0x04, 0x00                     // ParmVal #1: ParmVal: (512)
0xxx                           // checksum
0xF7                           // footer
```

3 Data Blocks

Data blocks are used in the message content field and have the following format:

Byte	Description
1	DataBlockSize: size of this data block (including this byte).
2	DataBlockType: data block type.
3 – N	DataBlockContent: Depends on the data block type.

Data Block Types:

Value	Name	Description
0x01	ParmList	Parameter List: Used to hold a list of parameter IDs to be retrieved from a device for a specific data class.
0x02	ParmDef	Parameter Definition: Used to define which parameters are supported by a device for a specific data class along with their attributes.
0x03	ParmVal	Parameter Value: Used to define parameter values when reading from a device or writing to a device for a specific data class.
0x04	ArgVal	Argument Value: Used to define argument values when reading parameters from a device or writing parameters to a device.
0x05	CmdDef	Command Definition: Used to define which commands and command values a device supports.
0x06	CmdVal	Command Value: Used to execute commands in a device.
0x70	BulkHdr	Bulk Header: Used for backup and restore operations.

ArgVal blocks (if they are used) should always precede ParmList and ParmVal blocks so that the arguments are defined before any parameters are returned or modified.

3.1 ParmList (Data Block Type = 0x01)

This data block is used to hold a list of parameters to be retrieved from a device for a specific data class.

Data Block Content:

Data block content is a single parameter list block.

Parameter List Block:

Byte	Description
3	NumParmID: number of parameter IDs that follow.
4 – N	ParmID: single byte parameter IDs.

Example: ParmList data block

```
0x05        // DataBlockSize: (5)
0x01        // DataBlockType: (ParmList)
0x02        // NumParmID: (2)
0x04        // ParmID #1: (4)
0x41        // ParmID #2: (65)
```

3.2 ParmDef (Data Block Type = 0x02)

This data block is used to define which parameters are supported for a specific data class along with their attributes.

Data Block Content:

Byte 3: NumParmDefBlock: number of parameter definition blocks that follow.
Bytes 4-N: parameter definition blocks.

Parameter Definition Block:

Byte	Description
1	ParmID : Parameter ID, specific to each data class.
2	ParmFlag: Parameter flags: bits 7 - 4: reserved (always 0). bit 3: set if this parameter supports scenes, clear if this parameter does not support scenes. bit 2: set if this parameter is a preset parameter, clear if this is a global parameter. bit 1: for writeable parameters, this bit is set if changing this parameter requires the device to be rebooted for the change to take effect, for read-only parameters this bit is set if the parameter is a constant value that never changes. bit 0: set if this parameter is writeable, clear if this parameter is read-only. The following notation is used for the flags: RD = read-only dynamic parameter (bit 0 = 0, bit 1 = 0) RC = read-only constant parameter (bit 0 = 0, bit 1 = 1) WN = writeable normal parameter (bit 0 = 1, bit 1 = 0) WB = writable reboot parameter (bit 0 = 1, bit 1 = 1) P = preset parameter (bit 2 = 1) G = global parameter (bit 2 = 0) S = scene parameter (bit 3 = 1) T = not a scene parameter (bit 3 = 0) Examples: RCGT = global read-only constant parameter (no scene support) WBPT = preset writeable parameter that requires a reboot to take effect (no scene support) RDPT = preset read-only dynamic parameter (no scene support) WNPS = preset writeable normal scene parameter

Example: ParmDef data block

```
0x0B                    // DataBlockSize: (11)
0x02                    // DataBlockType: (ParmDef)
0x04                    // NumParmDefBlock: (4)
0x04, 0x00              // block #1: ID = 4, flags = RDGT
0x07, 0x02              // block #2: ID = 7, flags = RCGT
0x09, 0x0D              // block #3: ID = 9, flags = WNPS
0x05, 0x03              // block #4: ID = 5, flags = WBGT
```


3.3 ParmVal (Data Block Type = 0x03)

This data block is used to define parameter values when reading from a device or writing to a device for a specific data class.

Data Block Content:

Byte 3: NumParmValBlock: number of parameter value blocks that follow.
Bytes 4-N: parameter value blocks.

Parameter Value Block:

Byte	Description
1	ParmSize: Size of this block (including this byte).
2	ParmID: Parameter ID, specific to each data class.
3 – N	ParmVal: Parameter value, specific to the parameter ID. Some parameters consist of a single value, some parameters use bitmaps and contain several values, some parameters use a fixed structure, and some parameters use a variable length structure comprising a number of sub-ID and value pairs.

Example: ParmVal data block

```
0x12                    // DataBlockSize: (18)
0x03                    // DataBlockType: (ParmVal)
0x02                    // NumParmValBlock: (3)
0x03                    // block #1: ParmSize (3)
0x04                    // block #1: ParmID (4)
0x09                    // block #1: ParmVal (9)
0x04                    // block #2: ParmSize (4)
0x41                    // block #2: ParmID (65)
0x01, 0x02              // block #2: ParmVal (0x0102)
0x08                    // block #3: ParmSize (8)
0x05                    // block #3: ParmID (5)
0x01, 0x03              // block #3: ParmVal: sub-ID (1), value (3)
0x02, 0x08              // block #3: ParmVal: sub-ID (2), value (8)
0x04, 0x09              // block #3: ParmVal: sub-ID (4), value (9)
```

3.4 ArgVal (Data Block Type = 0x04)

This data block is used to define argument values when reading parameters from a device or writing parameters to a device. If argument value blocks are used, they should always precede the ParmList and ParmVal blocks to which they apply. Not all arguments are applicable to every data class; see the details for each data class for a list of valid arguments.

Data Block Content:

Byte 3: NumArgValBlock: number of argument value blocks that follow.
Bytes 4-N: argument value blocks.

Argument Value Block:

Byte	Description
1	ArgID: Argument ID, see tables below.
2	ArgVal: Argument value, specific to the argument ID.

All Arguments:

ArgID	Name	Description
0x01	AreaID	Area selection (0 – N), optional. Only applies to devices that support areas. 0 is used for the work area, 1 – N is used for the shadow areas. Work area is used if this argument is not supplied.
0x02	SceneID	Scene selection (0 – N). Only applies to devices and parameters that support scenes. 0 is used for the currently active scene, 1 – N is used to specify a particular scene. Use 0 for globals and parameters that do not support scenes.
0x03	HWPortType	Hardware port type (1 – N).
0x04	HWPortID	Hardware port ID (1 – N).
0x05	MIDIPortID	MIDI port ID (1 – N).
0x06	MIDIChannel	MIDI channel number (1 – 16).
0x07	AMPID	AMP ID (1 – N). This specifies the AMP algorithm ID, the operatorID, the custom route map ID, or the lookup table ID (depending on the parameter ID).
0x08	USBHMIDIID	USB host controller MIDI ID (1 – N).
0x09	PresetID	Preset ID (1 – N). Can be used to read some preset parameters directly from non-volatile storage without having to load a preset into the working area of the shadow area.

The following table shows all the values that can be used for a device that supports one shadow area and two scenes:

AreaID	SceneID	Description
0	0	Work area, currently active scene.
0	1	Work area, scene 1.
0	2	Work area, scene 2.

AreaID	SceneID	Description
1	0	Shadow area, currently active scene.
1	1	Shadow area, scene 1.
1	2	Shadow area, scene 2.

Example: ArgVal data block

```
0x07        // DataBlockSize: (7)
0x04        // DataBlockType: (ArgVal)
0x02        // NumArgValBlock: (2)
0x01        // block #1: ArgID: (1)
0x00        // block #1: ArgVal: (0)
0x02        // block #2: ArgID: (2)
0x01        // block #2: ArgVal: (1)
```

3.5 CmdDef (Data Block Type = 0x05)

This data block is used to define which command IDs and command values a device supports.

Data Block Content:

Byte 3: NumCmdDefBlock: number of command definition blocks that follow.
Bytes 4-N: command definition blocks.

Command Definition Block:

Byte	Description
1	CmdSize: Size of this block (including this byte).
2	CmdID: Command ID (0x01 – 0x7F).
3 – N	CmdVal: Command values supported for this command ID (0x01 – 0x7F).

Example: CmdDef data block

```
0x0A        // DataBlockSize: (10)
0x05        // DataBlockType: (CmdDef)
0x02        // NumCmdDefBlock: (2)
0x03        // block #1: CmdSize: (3)
0x04        // block #1: CmdID: (4)
0x09        // block #1: CmdVal: (9)
0x04        // block #2: CmdSize: (4)
0x41        // block #2: CmdID: (65)
0x07        // block #2: CmdVal #1: (7)
0x09        // block #2: CmdVal #2: (9)
```


3.6 CmdVal (Data Block Type = 0x06)

This data block is sent from a host to a device to execute commands.

Data Block Content:

Byte 3: NumCmdValBlock: number of command value blocks that follow.
Bytes 4-N: command value blocks.

Command Value Block:

Byte	Description
1	CmdSize: Size of this block (including this byte).
2	CmdID: Command ID (0x01 – 0x7F).
3	CmdVal: Command value (0x01 – 0x7F).
4 – N	CmdArg: Command arguments (specific to the command ID and command value).

Example: CmdVal data block

```
0x0B    // DataBlockSize: (11)
0x06    // DataBlockType: (CmdVal)
0x02    // NumCmdValBlock: (2)
0x03    // block #1: CmdSize: (3)
0x04    // block #1: CmdID: (4)
0x09    // block #1: CmdVal: (9)
0x05    // block #2: CmdSize: (5)
0x41    // block #2: CmdID: (65)
0x07    // block #2: CmdVal: (7)
0x01    // block #2: CmdArg 1: (1)
0x08    // block #2: CmdArg 2: (8)
```

3.7 BulkHdr (Data Block Type = 0x70)

This data block is used for backup and restore operations. See the Bulk Data Transfer section for more information regarding bulk data transfer operations.

4 Message Classes

The following Message Classes are defined:

Message Classes:

Value	Name	Description
0x01	HstSesnVal	Sent from host to device to exchange session information.
0x02	GetParmDef	Sent from host to device to retrieve a list of parameters.
0x03	GetParmVal	Sent from host to device to retrieve parameter values.
0x04	GetCmdDef	Sent from host to device to retrieve a list of commands.
0x10	SetParmVal	Sent from host to device to set parameter values.
0x11	SetCmdVal	Sent from host to device to execute a command.
0x40	Ack	Sent from device to host in response to various messages (indicates success or failure).
0x41	DevSesnVal	Sent from device to host in response to a HstSesnVal message (exchange session information)
0x42	RetParmDef	Sent from device to host in response to a GetParmDef message (return a parameter list).
0x43	RetParmVal	Sent from device to host in response to a GetParmVal message (return parameter values).
0x44	RetCmdDef	Sent from device to host in response to a GetCmdDef message (return a command list).
0x50	NotParmVal	Sent from a device to a host when parameters are changed on a device that the host did not initiate.
0x70	BulkTransfer	Sent between a device and a host (or another device) to perform bulk data transfer operations (i.e. backup and restore).

4.1 HstSesnVal (Message Class = 0x01)

HstSesnVal messages are sent from a host to a device to exchange session information. The device will return either a DevSesnVal message or an Ack message with an error code. The following format is used for all HstSesnVal messages:

Byte	Description
1	Message Class (0x01)
2	Data Class (0x01 = SessionInfo)
3	NumDataBlock: number of data blocks that follow.

Byte	Description
4 – N	One or more ParmVal data blocks.

Example: HstSesnVal for data class SessionInfo

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x0A                     // message length (10)
0x01, 0x01                     // message class, data class
0x01                           // NumDataBlock: (1)

0x07, 0x03                     // data block #1: size (7), type (ParmVal)
0x01                           // NumParmValBlock: (1)
0x04                           // block #1: ParmSize: (4)
0x01                           // block #1: ParmID: (1)
0x02, 0x00                     // block #1: ParmVal: (256)

0xxx                           // checksum
0xF7                           // footer
```

4.2 GetParmDef (Message Class = 0x02)

GetParmDef messages are sent from a host to a device to retrieve a list of parameter definitions from a device for a specific data class. The device will return either a RetParmDef message or an Ack message with an error code. The following format is used for all GetParmDef messages for all data classes:

Byte	Description
1	Message Class (0x02)
2	Data Class (0x01 – 0x7F)
	No message content for this message class.

Example: GetParmDef for data class nn

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x02                     // message length (2)
0x02, 0xnn                     // message class, data class
0xxx                           // checksum
0xF7                           // footer
```

4.3 GetParmVal (Message Class = 0x03)

GetParmVal messages are sent from a host to a device to retrieve parameter values from a device. The device will return either a RetParmVal message or an Ack message with an error code. The following format is used for all GetParmVal messages:

Byte	Description
1	Message Class (0x03)
2	Data Class (0x01 – 0x7F)
3	NumDataBlock: number of data blocks that follow.
4 – N	One ArgVal data block (optional, depends on data class), followed by one or more ParmList data blocks.

Example: GetParmVal for data class nn, using ArgVal data block

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
0x00, 0x05                      // product ID
0x01, 0x02, 0x03, 0x04, 0x05    // serial number
0x00, 0x00, 0x00, 0x00          // session ID
0x00, 0x00, 0x00, 0x00          // transaction ID
0x00, 0x10                      // message length (13)
0x03, 0xnn                      // message class, data class
0x02                            // NumDataBlock: (2)

0x05, 0x04                      // data block #1: size (5), type (ArgVal)
0x01                            // NumArgValBlock: (1)
0x01                            // block #1: ArgID: (1)
0x01                            // block #1: ArgVal: (1)

0x05, 0x01                      // data block #2: size (5), type (ParmList)
0x02                            // NumParmID: (2)
0x04                            // ParmID #1: (4)
0x41                            // ParmID #2: (65)

0xxx                            // checksum
0xF7                            // footer
```

4.4 GetCmdDef (Message Class = 0x04)

GetCmdDef messages are sent from a host to a device to retrieve a list of executable commands that the device supports. The device will return either a RetCmdDef message or an Ack message with an error code. The following format is used for all GetCmdDef messages:

Byte	Description
1	Message Class (0x04)
2	Data Class (0x00)
	No message content for this message class.

Example: GetCmdDef

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x02                     // message length (2)
0x04, 0x00                     // message class, data class
0xxx                           // checksum
0xF7                           // footer
```

4.5 SetParmVal (Message Class = 0x10)

SetParmVal messages are sent from a host to a device to set parameter values in a device. The device will return an Ack message. The following format is used for all SetParmVal messages:

Byte	Description
1	Message Class (0x10)
2	Data Class (0x01 – 0x7F)
3	NumDataBlock: number of data blocks that follow.
4 – N	One ArgVal data block (optional, depends on data class), followed by one or more ParmVal data blocks.

Example: SetParmVal for data class nn, using ArgVal data block

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x12                     // message length (18)
0x10, 0xnn                     // message class, data class
0x02                           // NumDataBlock: (2)

0x05, 0x04                     // data block #1: size (5), type (ArgVal)
0x01                           // NumArgValBlock: (1)
```

```
0x01                        // block #1: ArgID: (1)
0x01                        // block #1: ArgVal: (1)

0x0A, 0x03                  // data block #2: size (10), type (ParmVal)
0x02                        // NumParmValBlock: (2)
0x03                        // block #1: ParmSize: (3)
0x04                        // block #1: ParmID: (4)
0x09                        // block #1: ParmVal: (9)
0x04                        // block #2: ParmSize: (4)
0x41                        // block #2: ParmID: (65)
0x01, 0x02                  // block #2: ParmVal: (0x0102)

0xxx                        // checksum
0xF7                        // footer
```

4.6 SetCmdVal (Message Class = 0x11)

SetCmdVal messages are sent from a host to a device to execute a command. The device will return an Ack message. The following format is used for all SetCmdVal messages:

Byte	Description
1	Message Class (0x11)
2	Data Class (0x00)
3	NumDataBlock: number of data blocks that follow.
4 – N	One or more CmdVal data blocks.

Example: SetCmdVal

```
0xF0, 0x00, 0x01, 0x73, 0x7D  // header
0x00, 0x05                    // product ID
0x01, 0x02, 0x03, 0x04, 0x05  // serial number
0x00, 0x00, 0x00, 0x00        // session ID
0x00, 0x00, 0x00, 0x00        // transaction ID
0x00, 0x0D                    // message length (13)
0x11, 0x00                    // message class, data class
0x01                          // NumDataBlock: (1)

0x0A, 0x06                    // data block #1: size (10), type (CmdVal)
0x02                          // NumCmdValBlock: (2)
0x03                          // block #1: CmdSize: (3)
0x04                          // block #1: CmdID: (4)
0x09                          // block #1: CmdVal: (9)
0x04                          // block #2: CmdSize: (4)
0x41                          // block #2: CmdID: (65)
0x07                          // block #2: CmdVal: (7)
0x01                          // block #2: CmdArg 1: (1)

0xxx                          // checksum
```

0xF7 // footer

4.7 Ack (Message Class = 0x40)

Ack messages are sent from a device to a host in response to various messages sent from the host (returns success or failure). The following format is used for all Ack messages:

Byte	Description
1	Message Class (0x40)
2	Data Class (0x00)
3	Message class of the host message that caused this Ack message to be sent.
4	Data class of the host message that caused this Ack message to be sent.
5	ErrorCode: an error code from the Ack Error Codes table.

Ack Error Codes:

Value	Description
0x00	No error.
0x01	Malformed message, message failed to parse correctly, bad length, bad checksum.
0x02	Message class not supported.
0x03	Data class not supported.
0x04	Message in too large, message cannot be received because it is too large.
0x05	Message out too large, message cannot be sent because it is too large.
0x06	Data block length is invalid.
0x07	Data block type is invalid.
0x08	Argument ID is invalid.
0x09	Argument value is invalid.
0x0A	Parameter ID is invalid.
0x0B	Parameter value is invalid.
0x0C	Invalid characters used for name.

Value	Description
0x0D	Command ID is invalid.
0x0E	Command value is invalid.
0x0F	Command argument is invalid.
0x10	Required ArgVal block was not found or does not occur before other data blocks.
0x11	Sub ID is invalid.
0x12	Sub ID value is invalid.
0x13	Command failed.

Example: Ack for data class 0x02 with no error

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
0x00, 0x05                      // product ID
0x01, 0x02, 0x03, 0x04, 0x05    // serial number
0x00, 0x00, 0x00, 0x00          // session ID
0x00, 0x00, 0x00, 0x00          // transaction ID
0x00, 0x05                      // message length (5)
0x40, 0x00                      // message class, data class of this message
0x01, 0x02                      // message class, data class of host message
0x00                            // ErrorCode: no error
0xxx                            // checksum
0xF7                            // footer
```

4.8 DevSesnVal (Message Class = 0x41)

DevSesnVal messages are sent from a device to a host in response to an HstSesnVal message (exchange session information). The following format is used for all DevSesnVal messages:

Byte	Description
1	Message Class (0x41)
2	Data Class (0x01 = SessionInfo)
3	NumDataBlock: number of data blocks that follow.
4 – N	One or more ParmVal data blocks.

Example: DevSesnVal for data class SessionInfo

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
```

```
0x00, 0x05                  // product ID
0x01, 0x02, 0x03, 0x04, 0x05  // serial number
0x00, 0x00, 0x00, 0x00      // session ID
0x00, 0x00, 0x00, 0x00      // transaction ID
0x00, 0x17                  // message length (23)
0x41, 0x01                  // message class, data class
0x01                        // NumDataBlock: (1)

0x14, 0x03                  // data block #1: size (20), type (ParmVal)
0x04                        // NumParmValBlock: (4)
0x04                        // block #1: ParmSize: (4)
0x10                        // block #1: ParmID: (0x10)
0x02, 0x00                  // block #1: ParmVal: (256)
0x04                        // block #2: ParmSize: (4)
0x11                        // block #2: ParmID: (0x11)
0x02, 0x00                  // block #2: ParmVal: (256)
0x03                        // block #3: ParmSize: (3)
0x12                        // block #3: ParmID: (0x12)
0x01                        // block #3: ParmVal: (1)
0x06                        // block #4: ParmSize: (6)
0x13                        // block #4: ParmID: (0x13)
0x01, 0x02, 0x03, 0x04      // block #4: ParmVal: (0x01020304)

0xxx                        // checksum
0xF7                        // footer
```

4.9 RetParmDef (Message Class = 0x42)

RetParmDef messages are sent from a device to a host in response to a GetParmDef message (returns a list of parameters from a device). The following format is used for all RetParmDef messages for all data classes:

Byte	Description
1	Message Class (0x42)
2	Data Class (0x01 – 0x7F)
3	NumDataBlock: number of data blocks that follow.
4 – N	One or more ParmDef data blocks.

Example: RetParmDef for data class nn

```
0xF0, 0x00, 0x01, 0x73, 0x7D  // header
0x00, 0x05                  // product ID
0x01, 0x02, 0x03, 0x04, 0x05  // serial number
0x00, 0x00, 0x00, 0x00      // session ID
0x00, 0x00, 0x00, 0x00      // transaction ID
0x00, 0x0E                  // message length (14)
0x42, 0xnn                  // message class, data class
0x01                        // NumDataBlock (1)
```

```
0x0B, 0x02          // data block #1: size (11), type (ParmDef)
0x04                // NumParmDefBlock: (4)
0x04, 0x00          // block #1: ID = 4, flags = RDGT
0x07, 0x02          // block #2: ID = 7, flags = RCGT
0x09, 0x0D          // block #3: ID = 9, flags = WNPS
0x05, 0x03          // block #4: ID = 5, flags = WBGT

0xxx                // checksum
0xF7                // footer
```

4.10 RetParmVal (Message Class = 0x43)

RetParmVal messages are sent from a device to a host in response to a GetParmVal message (return parameter values from a device). The following format is used for all RetParmVal messages:

Byte	Description
1	Message Class (0x43)
2	Data Class (0x01 – 0x7F)
3	NumDataBlock: number of data blocks that follow.
4 – N	One ArgVal data block (optional, depends on data class), followed by one or more ParmVal data blocks.

Example: RetParmVal for data class nn, using ArgVal data block

```
0xF0, 0x00, 0x01, 0x73, 0x7D  // header
0x00, 0x05                    // product ID
0x01, 0x02, 0x03, 0x04, 0x05  // serial number
0x00, 0x00, 0x00, 0x00        // session ID
0x00, 0x00, 0x00, 0x00        // transaction ID
0x00, 0x12                    // message length (18)
0x43, 0xnn                    // message class, data class
0x02                          // NumDataBlock: (2)

0x05, 0x04                    // data block #1: size (5), type (ArgVal)
0x01                          // NumArgValBlock: (1)
0x01                          // block #1: ArgID: (1)
0x01                          // block #1: ArgVal: (1)

0x0A, 0x03                    // data block #2: size (10), type (ParmVal)
0x02                          // NumParmValBlock: (2)
0x03                          // block #1: ParmSize: (3)
0x04                          // block #1: ParmID: (4)
0x09                          // block #1: ParmVal: (9)
0x04                          // block #2: ParmSize: (4)
0x41                          // block #2: ParmID: (65)
0x01, 0x02                    // block #2: ParmVal: (0x0102)
```

```
0xxx                    // checksum
0xF7                    // footer
```

4.11 RetCmdDef (Message Class = 0x44)

RetCmdDef messages are sent from a device to a host in response to a GetCmdDef message (returns a list of command IDs and command values that the device supports). The following format is used for all RetCmdDef messages:

Byte	Description
1	Message Class (0x44)
2	Data Class (0x00)
3	NumCmdDefBlock: number of command definition blocks that follow.
4 – N	One or more command definition blocks.

Example: RetCmdDef, two command IDs supported (one and two command values respectivey)

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
0x00, 0x05                      // product ID
0x01, 0x02, 0x03, 0x04, 0x05    // serial number
0x00, 0x00, 0x00, 0x00          // session ID
0x00, 0x00, 0x00, 0x00          // transaction ID
0x00, 0x0D                      // message length (13)
0x44, 0x00                      // message class, data class
0x01                            // NumDataBlock: (1)

0x0A, 0x05                      // data block #1: size (10), type (CmdDef)
0x02                            // NumCmdDefBlock: (2)
0x03                            // block #1: CmdSize: (3)
0x04                            // block #1: CmdID: (4)
0x09                            // block #1: CmdVal: (9)
0x04                            // block #2: CmdSize: (4)
0x41                            // block #2: CmdID: (65)
0x07                            // block #2: CmdVal 1: (7)
0x09                            // block #2: CmdVal 2: (9)

0xxx                            // checksum
0xF7                            // footer
```

4.12 NotParmVal (Message Class = 0x50)

NotParmVal messages are sent from a device to a host when parameters are changed on a device that the host did not initiate, but only if the host has registered for automatic updates using the Notification command. The following format is used for all NotParmVal messages (which is identical to the RetParmVal message format):

Byte	Description
1	Message Class (0x50)
2	Data Class (0x01 – 0x7F)
3	NumDataBlock: number of data blocks that follow.
4 – N	One ArgVal data block (optional, depends on data class), followed by one or more ParmVal data blocks.

4.13 BulkTransfer (Message Class = 0x70)

BulkTransfer messages are sent between a device and a host (or another device) to perform bulk data transfer operations (i.e. backup and restore). See the Bulk Data Transfer section for more information regarding bulk data transfer operations. The following format is used for all BulkTransfer messages:

Byte	Description
1	Message Class (0x70)
2	Data Class (0x70)
3	NumDataBlock: number of data blocks that follow.
4 – N	One bulk header data block possibly followed by additional data blocks.

5 Data Classes

The following Data Classes are defined:

Data Classes:

Value	Name	Description
0x00	null	Placeholder value for messages that don't require a data class.
0x01	SessionInfo	Used to discover devices and to establish communications parameters.
0x02	DeviceInfo	Contains many read-only global parameters for a device.
0x03	DeviceFeature	Contains parameters related to global device features.

Value	Name	Description
0x04	HardwareInfo	Contains parameters related to hardware ports.
0x05	MIDIInfo	Contains global parameters related to the MIDI configuration of a device.
0x06	MIDIPortInfo	Contains parameters related to specific MIDI ports.
0x07	MIDIFeature	Contains parameters for advanced MIDI features.
0x70	BulkData	Used to perform bulk data transfer operations.

5.1 SessionInfo (Data Class = 0x01)

This data class is used to discover devices and to establish communication parameters for a session. Sessions are stateless but a device will maintain any session values established on a MIDI port until the device is either reset or a new set of session parameters are established on that MIDI port.

The host sends a HstSesnVal message to discover which MIDI ports have devices attached to them and to determine the best MIDI port to establish a communication session. The host can specify a specific product ID (PID), a specific serial number (SNUM), both PID & SNUM, or neither (using wildcards). This message should always be the first query sent to a device because the DevSesnVal response contains useful information about which MIDI port the host should use to establish a communication session with the device.

Different parameters are used for the HstSesnVal and DevSesnVal messages. The host should send all host related parameters in the HstSesnVal message. The device will return all device related parameters in the DevSesnVal message.

Message Class Support:

Message Class	Notes
HstSesnVal DevSesnVal	

HstSesnVal Parameters:

ID	Name	Description
0x01	HstInSizeMax	Maximum length allowed for sysex messages sent to the host (2 bytes, 14x2 format). If the device needs to send a sysex message to the host that is longer than this value the device will return an Ack message with an error code. The host must then either: allocate a larger buffer then send another HstSesnVal message with the larger buffer size and then resend the message that failed, or split the message into several messages so that the

ID	Name	Description
		device can respond with several smaller messages. A device cannot automatically split a message that is too large into several smaller messages.

DevSesnVal Parameters:

Note that these parameters are a small subset of the DeviceInfo parameters.

ID	Name	Description
0x10	DevInSizeMax	Maximum length allowed for sysex messages sent to the device (2 bytes, 14x2 format). If the host needs to send a sysex message to the device that is longer than this value the host will need to split the message into multiple smaller messages.
0x11	DevOutSizeMax	Maximum length for sysex messages that the device can send back to the host (2 bytes, 14x2 format). The host can use this value to allocate a buffer for incoming sysex messages from the device. This value will always be less than or equal to the HstInSizeMax value that the host sent in the HstSesnVal message.
0x12	DevOpMode	Device's current operating mode (1 byte): 0 = bootloader mode 1 = application mode
0x13	DevMIDIPortInfo	Info regarding the MIDI port that this message is sent on (4 bytes): Byte 1: MIDI port ID (1 – N) Byte 2: MIDI port type: 0x01 = DIN 0x02 = USB device 0x03 = USB host 0x04 = Ethernet Bytes 3-4: MIDI port detail, depends on MIDI port type: DIN: Byte 3: DIN IN port (1 – N) Byte 4: DIN OUT port (1 – N) USB device: Byte 3: USB device port (1 - N) Byte 4: MIDI port (1 - 16) USB host: Byte 3: USB host controller (1 - N) Byte 4: USB host controller port (1 - N) Ethernet: Byte 3: Ethernet port (1 - N) Byte 4: RTP-MIDI session (1 - N)

Example: HstSesnVal (discover devices, any product, any serial number)

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
0x00, 0x00                      // product ID
```

```
0x00, 0x00, 0x00, 0x00, 0x00    // serial number
0x00, 0x00, 0x00, 0x00          // session ID
0x00, 0x00, 0x00, 0x00          // transaction ID
0x00, 0x0A                      // message length (10)
0x01, 0x01                      // message class, data class,
0x01                            // NumDataBlock: (1)

0x07, 0x03                      // data block #1: size (7), type (ParmVal)
0x01                            // NumParmValBlock: (1)
0x04                            // block #1: ParmSize: (4)
0x01                            // block #1: ParmID: HstInSizeMax (0x01)
0x04, 0x00                      // block #1: ParmVal: (512)

0xxx                            // checksum
0xF7                            // footer
```

Example: HstSesnVal (discover all devices that are a specific product ID (5), any serial number)

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
0x00, 0x05                      // product ID (5)
0x00, 0x00, 0x00, 0x00, 0x00    // serial number
0x00, 0x00, 0x00, 0x00          // session ID
0x00, 0x00, 0x00, 0x00          // transaction ID
0x00, 0x0A                      // message length (10)
0x01, 0x01                      // message class, data class
0x01                            // NumDataBlock: (1)

0x07, 0x03                      // data block #1: size (7), type (ParmVal)
0x01                            // NumParmValBlock: (1)
0x04                            // block #1: ParmSize: (4)
0x01                            // block #1: ParmID: HstInSizeMax (0x01)
0x02, 0x00                      // block #1: ParmVal: (256)

0xxx                            // checksum
0xF7                            // footer
```

Example: DevSesnVal (response to HstSesnVal example above)

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
0x00, 0x05                      // product ID (5)
0x01, 0x02, 0x03, 0x04, 0x05    // serial number
0x00, 0x00, 0x00, 0x00          // session ID
0x00, 0x00, 0x00, 0x00          // transaction ID
0x00, 0x17                      // message length (23)
0x41, 0x01                      // message class, data class
0x01                            // NumDataBlock: (1)

0x14, 0x03                      // data block #1: size (20), type (ParmVal)
0x04                            // NumParmValBlock: (4)
0x04                            // block #1: ParmSize: (4)
0x10                            // block #1: ParmID: DevInSizeMax (0x10)
0x02, 0x00                      // block #1: ParmVal: (256)
0x04                            // block #2: ParmSize: (4)
0x11                            // block #2: ParmID: DevOutSizeMax (0x11)
0x02, 0x00                      // block #2: ParmVal: (256)
0x03                            // block #3: ParmSize: (3)
```

```
0x12                    // block #3: ParmID: DevOpMode (0x12)
0x01                    // block #3: ParmVal: application mode (1)
0x06                    // block #4: ParmSize: (6)
0x13                    // block #4: ParmID: DevMIDIPortInfo (0x13)
0x05                    // block #4: ParmVal 1: MIDI port ID (5)
0x02                    // block #4: ParmVal 2: MIDI port type: USB device (2)
0x01                    // block #4: ParmVal 3: USB device port number (1)
0x01                    // block #4: ParmVal 4: MIDI port number (1)

0xxx                    // checksum
0xF7                    // footer
```

5.2 DeviceInfo (Data Class = 0x02)

This data class contains many read-only global parameters for a device and is also used to set the device name. This data class should always be one of the first messages sent to a device.

Message Class Support:

Message Class	Notes
GetParmDef RetParmDef	
GetParmVal SetParmVal RetParmVal NotParmVal	Argument value block is optional. All parameters are global and do not support scenes.

Applicable Arguments:

ArgID	Name	Description
0x01	Area	Area selection (0 – N), optional. Only applies to devices that support areas. 0 is used for the work area, 1 – N is used for the shadow areas. Work area is used if this argument is not supplied.

Parameters:

ID	Name	Description
0x01	ProductName	Product name (7-bit ASCII string).
0x02	MfgName	Manufacturer (7-bit ASCII string).
0x03	ModelNumber	Model number (7-bit ASCII string).

ID	Name	Description
0x04	SerialNumber	Serial number (7-bit ASCII string).
0x05	FirmwareVersion	Firmware version (4 bytes) using the format: Major.Minor.Revision.Beta (where 0 is used in the Beta field for final builds). Examples: 0x01 0x02 0x03 0x00 = 1.2.3 0x02 0x00 0x0B 0x04 = 2.0.11b4
0x06	HardwareVersion	Hardware version (2 bytes) using the format: Major.Minor. Examples: 0x01 0x02 = 1.2 0x02 0x22 = 2.34
0x07	DevNameMax	Maximum length allowed for device name (DevIinfo:DevName) (1 byte).
0x08	DevUserDataMax	Max length allowed for device user data (DevInfo:DevUserData) (1 byte).
0x09	DINInPortCount	Number of DIN IN ports (0 – N) (1 byte).
0x0A	DINOutPortCount	Number of DIN OUT ports (0 – N) (1 byte).
0x0B	USBDPortCount	Number of USB device ports (0 – N) (1 byte).
0x0C	USBHPortCount	Number of USB host controller ports (0 – N) (1 byte).
0x0D	EthPortCount	Number of Ethernet ports (0 – N) (1 byte).
0x0E	CtrlPortCount	Number of control ports (0 – N) (1 byte).
0x0F	HWPortNameMax	Maximum length allowed for hardware port name (HardwareInfo:HWPortName) (1 byte).
0x10	DevInSizeMax	Maximum length allowed for sysex messages sent to the device (2 bytes, 14x2 format). If the host needs to send a sysex message to the device that is longer than this value the host will need to split the message into multiple smaller messages.
0x11	DevOutSizeMax	Maximum length for sysex messages that the device can send back to the host (2 bytes, 14x2 format). The host can use this value to allocate a buffer for incoming sysex messages from the device. This value will always be less than or equal to the HstInSizeMax value that the host sent in the most recent HstSesnVal message.
0x12	DevOpMode	Device's current operating mode (1 byte): 0 = bootloader mode 1 = application mode
0x13	DevMIDIPortInfo	Info regarding the MIDI port that this message is sent on (4 bytes): Byte 1: MIDI port ID (1 – N) Byte 2: MIDI port type: 0x01 = DIN

ID	Name	Description
		0x02 = USB device 0x03 = USB host 0x04 = Ethernet Bytes 3-4: MIDI port detail, depends on MIDI port type: DIN: Byte 3: DIN IN port (1 – N) Byte 4: DIN OUT port (1 – N) USB device: Byte 3: USB device port (1 - N) Byte 4: MIDI port (1 - 16) USB host: Byte 3: USB host controller (1 - N) Byte 4: USB host controller port (1 - N) Ethernet: Byte 3: Ethernet port (1 - N) Byte 4: RTP-MIDI session (1 - N)
0x14	PresetMax	Maximum number of presets supported (1 – N) (1 byte).
0x15	PresetNameMax	Maximum length allowed for preset names (DeviceFeature:PresetName) (1 byte).
0x16	PresetUserDataMax	Maximum length allowed for preset user data (DeviceFeature:PresetUserData) (1 byte).
0x17	SceneMax	Maximum number of scenes supported per preset (1 – N) (1 byte). A value of 1 indicates a device that does not support scene changes.
0x18	ShadowAreaMax	Maximum number of shadow areas supported (0 – N) (1 byte).
0x19	NotificationTimeout	Notification timeout, in seconds (1 byte). If a host registers for automatic updates using the Notification command, the host must ensure that it sends a sysex message to the device at regular intervals otherwise the device will assume the host is not there and will stop sending notification messages. Example: Timeout = 5 // host should send a sysex message every 5 seconds (or sooner) to continue receiving automatic updates
0x40	DevName	Device name (7-bit ASCII string). This name will be used for the Bonjour name of the device on a network. Note that the actual name used on the network may be different if there is a name conflict following Bonjour name resolution (see HardwareInfo:EthDevName). Maximum length is given by DevInfo:DevNameMax.
0x41	DevUserData	Device user data. Free field for storing user data (all bytes must be in the range 0x00 – 0x7F). Bytes 1: index (i.e. starting address) Bytes 2-N: user data

ID	Name	Description
		The first byte is an index into the user data field (i.e. the starting address). The remaining bytes are the user data. RetParmVal will always return the entire user data field (index is always 0, user data is always DevUserDataMax bytes in length). SetParmVal can use a non-zero value for the offset byte and can write just a portion of the user data field. Maximum length is given by DevInfo:DevUserDataMax.

Example: GetParmDef

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x02                     // message length (2)
0x02, 0x02                     // message class, data class
0xxx                           // checksum
0xF7                           // footer
```

Example: RetParmDef

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x0E                     // message length (14)
0x42, 0x02                     // message class, data class
0x01                           // NumDataBlock: (1)

0x0B, 0x02                     // data block #1: size (11), type (ParmDef)
0x04                           // NumParmDefBlock: (4)
0x01, 0x02                     // block #1: ID = 0x01, flags = RCGT
0x02, 0x02                     // block #2: ID = 0x02, flags = RCGT
0x07, 0x02                     // block #3: ID = 0x07, flags = RCGT
0x40, 0x01                     // block #4: ID = 0x40, flags = WNGT

0xxx                           // checksum
0xF7                           // footer
```

Example: GetParmVal

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x0D                     // message length (13)
0x03, 0x02                     // message class, data class
0x02                           // NumDataBlock: (2)
```

```
0x05, 0x04                  // data block #1: size (5), type (ArgVal)
0x01                        // NumArgValBlock: (1)
0x01                        // block #1: ArgID: (1)
0x01                        // block #1: ArgVal: (1)

0x05, 0x01                  // data block #2: size (5), type (ParmList)
0x02                        // NumParmID: (2)
0x07                        // ParmID #1: (7)
0x40                        // ParmID #2: (64)

0xxx                        // checksum
0xF7                        // footer
```

Example: RetParmVal

```
0xF0, 0x00, 0x01, 0x73, 0x7D  // header
0x00, 0x05                    // product ID
0x01, 0x02, 0x03, 0x04, 0x05  // serial number
0x00, 0x00, 0x00, 0x00        // session ID
0x00, 0x00, 0x00, 0x00        // transaction ID
0x00, 0x14                    // message length (20)
0x43, 0x02                    // message class, data class
0x02                          // NumDataBlock: (2)

0x05, 0x04                    // data block #1: size (5), type (ArgVal)
0x01                          // NumArgValBlock: (1)
0x01                          // block #1: ArgID: (1)
0x01                          // block #1: ArgVal: (1)

0x0C, 0x03                    // data block #2: size (12), type (ParmVal)
0x02                          // NumParmValBlock: (2)
0x03                          // block #1: size (3)
0x07                          // block #1: parm ID (7)
0x0F                          // block #1: parm value (15)
0x06                          // block #2: size (6)
0x40                          // block #2: parm ID (64)
0x41, 0x42, 0x43, 0x44        // block #2: parm value (ASCII string “ABCD”)

0xxx                          // checksum
0xF7                          // footer
```

Example: SetParmVal

```
0xF0, 0x00, 0x01, 0x73, 0x7D  // header
0x00, 0x05                    // product ID
0x01, 0x02, 0x03, 0x04, 0x05  // serial number
0x00, 0x00, 0x00, 0x00        // session ID
0x00, 0x00, 0x00, 0x00        // transaction ID
0x00, 0x14                    // message length (20)
0x10, 0x02                    // message class, data class
0x02                          // NumDataBlock: (2)

0x05, 0x04                    // data block #1: size (5), type (ArgVal)
0x01                          // NumArgValBlock: (1)
0x01                          // block #1: ArgID: (1)
```

```
0x01                        // block #1: ArgVal: (1)

0x0C, 0x03                  // data block #2: size (12), type (ParmVal)
0x01                        // NumParmValBlock: (2)
0x06                        // block #1: size (6)
0x40                        // block #1: parm ID (64)
0x61, 0x62, 0x63, 0x64      // block #1: parm value (ASCII string “abcd”)
0x03                        // block #2: size (3)
0x07                        // block #2: parm ID (7)
0x0C                        // block #2: parm value (12)

0xxx                        // checksum
0xF7                        // footer
```

5.3 DeviceFeature (Data Class = 0x03)

This data class contains parameters related to global device features.

Message Class Support:

Message Class	Notes
GetParmDef RetParmDef	
GetParmVal SetParmVal RetParmVal NotParmVal	Argument value block is optional. All parameters are global and do not support scenes.

5.3.1 Presets & Scenes

Applicable Arguments:

ArgID	Name	Description
0x01	Area	Area selection (0 – N), optional. Only applies to devices that support areas. 0 is used for the work area, 1 – N is used for the shadow areas. Work area is used if this argument is not supplied.
0x09	PresetID	Preset ID (1 – N). Maximum value is given by DeviceInfo:PresetMax. Required when using the PresetName parameter to read preset names from non-volatile storage.

Parameters for Presets & Scenes:

ID	Name	Description
0x01	PresetNumber	Currently loaded preset number (1 – N) (1 byte). This is a read-only parameter that cannot be used to change presets; use the SaveLoad commands to change presets. Maximum value is given by DevInfo:PresetMax.
0x02	PresetName	Preset name (7-bit ASCII string). Maximum length is given by DevInfo:PresetNameMax. Can be used to read/write the currently loaded preset name in the working area or the shadow area. Can also be used to read (but not write) preset names from non-volatile storage. When reading/writing the preset name in the working area ArgID:Area can be used but is not required. When reading/writing the preset name in the shadow area ArgID:Area is required. In both cases, ArgID:PresetID must not be used. When reading the preset name from non-volatile storage ArgID:Area can be used but will be ignored. However, ArgID:PresetID is required.
0x03	PresetUserData	Currently loaded preset user data. Free field for storing user data (all bytes must be in the range 0x00 – 0x7F). Bytes 1: index (i.e. starting address) Bytes 2-N: user data The first byte is an index into the user data field (i.e. the starting address). The remaining bytes are the user data. RetParmVal will always return the entire user data field (index is always 0, user data is always PresetUserDataMax bytes in length). SetParmVal can use a non-zero value for the offset byte and can write just a portion of the user data field. Maximum length is given by DevInfo:PresetUserDataMax.
0x04	SceneNumber	Currently active scene number (1 – N) (1 byte). This parameter can be used to change the active scene in the currently loaded preset. Maximum value is given by DevInfo:SceneMax.

Example: Get PresetName from shadow area

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
0x00, 0x05                      // product ID
0x01, 0x02, 0x03, 0x04, 0x05    // serial number
0x00, 0x00, 0x00, 0x00          // session ID
0x00, 0x00, 0x00, 0x00          // transaction ID
0x00, 0x0C                      // message length (12)
0x03, 0x03                      // message class, data class
0x02                            // NumDataBlock: (2)

0x05, 0x04                      // data block #1: size (5), type (ArgVal)
0x01                            // NumArgValBlock: (1)
0x01                            // block #1: ArgID: (1)
0x01                            // block #1: ArgVal: (1)

0x04, 0x01                      // data block #2: size (4), type (ParmList)
```

```
0x01                          // NumParmID: (1)
0x02                          // ParmID #1: (2)

0xxx                          // checksum
0xF7                          // footer
```

Example: Get PresetName for preset #4 from non-volatile storage

```
0xF0, 0x00, 0x01, 0x73, 0x7D  // header
0x00, 0x05                    // product ID
0x01, 0x02, 0x03, 0x04, 0x05  // serial number
0x00, 0x00, 0x00, 0x00        // session ID
0x00, 0x00, 0x00, 0x00        // transaction ID
0x00, 0x0C                    // message length (12)
0x03, 0x03                    // message class, data class
0x02                          // NumDataBlock: (2)

0x05, 0x04                    // data block #1: size (5), type (ArgVal)
0x01                          // NumArgValBlock: (1)
0x09                          // block #1: ArgID: (9)
0x04                          // block #1: ArgVal: (4)

0x04, 0x01                    // data block #2: size (4), type (ParmList)
0x01                          // NumParmID: (1)
0x02                          // ParmID #1: (2)

0xxx                          // checksum
0xF7                          // footer
```

5.3.2 Automatic Failover

This section will be completed later.

5.4 HardwareInfo (Data Class = 0x04)

This data class contains parameters related to hardware ports. The DeviceInfo data class returns information regarding the number and types of hardware ports that the device supports. Hardware port type and port ID are required arguments for all parameters in this data class. MIDIPortID is required for USBH-MIDI parameters.

Message Class Support:

Message Class	Notes
GetParmDef RetParmDef	
GetParmVal SetParmVal RetParmVal NotParmVal	Argument value block is required. All parameters are global and do not support scenes.

Applicable Arguments:

ArgID	Name	Description
0x01	Area	Area selection (0 – N), optional. Only applies to devices that support areas. 0 is used for the work area, 1 – N is used for the shadow areas. Work area is used if this argument is not supplied.
0x03	HWPortType	Hardware port type (1 – N), see table below. Required for all parameters.
0x04	HWPortID	Hardware port ID (1 – N). Required for all parameters.
0x08	USBHMIDIID	USB host controller MIDI ID (1 – N). Required for the USB host controller MIDI port parameters.

Hardware Port Types:

Value	Description
0x01	DIN In
0x02	DIN Out
0x03	USB Device
0x04	USB Host Controller
0x05	Ethernet
0x06	Control

Parameters for All Hardware Port Types:

ID	Name	Description
0x01	HWPortName	Hardware port name (7-bit ASCII string). Maximum length is given by DevInfo:HWPortNameMax.

Parameters for DIN In Ports:

ID	Name	Description
	none	

Parameters for DIN Out Ports:

ID	Name	Description
	none	

Parameters for USB Device Ports:

ID	Name	Description
0x10	USBDFlags	USB device port flags (1 byte): bits 7 - 2: reserved (always 0). bit 1: set if this port supports high-speed (480 Mbit), clear if this port only supports full-speed (12 Mbit) bit 0: set if this port supports firmware updates when the device is operating in bootloader mode
0x11	USBDConnect	Indicates if a host is connected to this USB device port and has enumerated the device (1 byte): 0 = no host connected or device not enumerated 1 = host connected and device is enumerated
0x12	USBDMIDI PortMax	Maximum number of MIDI ports supported on this USB device port (1 – 16) (1 byte).
0x13	USBDMIDI PortCount	Number of MIDI ports to use on this USB device port (1 – USBDMIDIPortMax) (1 byte).

Parameters for USB Host Controller Ports:

ID	Name	Description
0x20	USBHJackCount	The number of USB jacks on the interface for this host controller (1 – N) (1 byte).
0x21	USBHMIDI PortMax	Maximum number of MIDI ports supported on this host controller (1 – N) (1 byte).
0x22	USBHMIDI PortCount	Number of MIDI ports to use on this host controller (1 – USBHMIDIPortMax) (1 byte).
0x23	USBHMIDI MultiMax	Maximum number of ports to host on multi-port MIDI devices connected to this host controller (1 – USBHMIDIPortCount) (1 byte).
0x24	USBHMIDI MultiRoute	Flag to enable routing MIDI events between ports on multi-port MIDI devices connected to this host controller (1 byte). 0 = routing is disabled (MIDI routing matrix is not used for these ports)

ID	Name	Description
		1 = routing is enabled (MIDI routing matrix is used for these ports)

Parameters for USB Host Controller MIDI Ports:

These parameters provide information about all USB-MIDI devices on a USB host controller port even if they are not connected to a MIDI port. USBHMIDIID is a required parameter (1 – USBHMIDIPortCount) but this is not the same ID as used with the MIDIPortInfo data class. Note that similar parameters are present in the MIDIPortInfo section for USB host controller MIDI ports.

ID	Name	Description
0x28	USBHMIDI VID	Hosted device's USB vendor ID (16-bit value encoded in 3 bytes, 16x3 format). 0 if no device is hosted.
0x29	USBHMIDI PID	Hosted device's USB product ID (16-bit value encoded in 3 bytes, 16x3 format). 0 if no device is hosted.
0x2A	USBHMIDI VName	Hosted device's vendor name from USB descriptor (7-bit ASCII string).
0x2B	USBHMIDI PName	Hosted device's product name from USB descriptor (7-bit ASCII string).
0x2C	USBHMIDI SerialNum	Hosted device's serial number from USB descriptor (7-bit ASCII string). Most USB-MIDI devices do not have a serial number in the USB descriptor.
0x2D	USBHMIDI PortCountIn	Number of MIDI IN ports supported by the hosted device (1 – 16) (1 byte). 0 if no device is hosted.
0x2E	USBHMIDI PortCountOut	Number of MIDI OUT ports supported by the hosted device (1 – 16) (1 byte). 0 if no device is hosted.
0x2F	USBHMIDI Identifier	A unique ID assigned to each hosted device (1 – N) (1 byte). This can be used to distinguish between multiple devices that have the same USBHMIDIVID and USBHMIDIPID with a blank USBHMIDISerialNum. 0 if no device is hosted.

Parameters for Ethernet Ports:

ID	Name	Description
0x30	EthMACAddress	Ethernet MAC address, 48-bit value encoded in 12 bytes using BAx2 format. Example (MAC address = AC:7A:42:12:34:56): 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x02, 0x04, 0x0A, 0x07, 0x0C, 0x0A

ID	Name	Description
0x31	EthConnect	Indicates if Ethernet port is connected to a network (1 byte): 0 = not connected 1 = connected
0x32	EthCurrentIP	Current (active) IP address, subnet mask, and gateway address (15 bytes). All are four-byte big-endian values encoded in five-bytes (32x5 format). These values are only valid if the Ethernet port is connected to a network, is active, and (if EthIPMode = dynamic) has obtained values from a DHCP server or AutoIP has finished negotiation. Example: 0x0A, 0x4F, 0x78, 0x00, 0x08 // current IP address (169.254.0.8) 0x0F, 0x7F, 0x7C, 0x00, 0x00 // current subnet mask (255.255.0.0) 0x0A, 0x4F, 0x78, 0x00, 0x01 // current gateway address (169.254.0.1)
0x33	EthDevName	Device name on network (Bonjour name), 7-bit ASCII string, taken from DeviceInfo:DevName parameter. EthDevName is only valid if the Ethernet port is connected to a network and has finished Bonjour name resolution.
0x34	EthIPMode	IP mode (1 byte): 0 = use static IP (as defined in EthStaticIP) 1 = use dynamic IP (DHCP or AutoIP if DHCP server is not available)
0x35	EthStaticIP	Device IP address, subnet mask, and gateway address to use when EthIPMode = static (15 bytes). All are four-byte big-endian values encoded in five-bytes (32x5 format). Example: 0x0C, 0x05, 0x20, 0x02, 0x64 // static IP address (192.168.1.100) 0x0F, 0x7F, 0x7F, 0x7E, 0x00 // static subnet mask (255.255.255.0) 0x0C, 0x05, 0x20, 0x02, 0x01 // static gateway address (192.168.1.1)
0x36	EthMIDIPortMax	Maximum number of RTP-MIDI sessions supported on this Ethernet port (1 – N) (1 byte).
0x37	EthMIDIPortCount	Number of RTP-MIDI sessions to use on this Ethernet port (1 – EthMIDIPortMax) (1 byte).

Parameters for Control Ports:

ID	Name	Description
0x40	CtrlType	Control port type (1 byte): 0 = 1/4” phone jack, tip connection. 1 = 1/4” phone jack, ring connection.
0x41	CtrlFlags	Control port flags (1 byte): bits 7 - 2: reserved (always 0). bit 1: set if this port can be used as an output. bit 0: set if this port can be used as an input.

Example: GetParmDef

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x02                     // message length (2)
0x02, 0x04                     // message class, data class
0xxx                           // checksum
0xF7                           // footer
```

Example: RetParmDef

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x0E                     // message length (14)
0x42, 0x04                     // message class, data class
0x01                           // NumDataBlock: (1)

0x0B, 0x02                     // data block #1: size (11), type (ParmDef)
0x04                           // NumParmDefBlock: (4)
0x01, 0x02                     // block #1: ID = 0x01, flags = RCGT
0x02, 0x00                     // block #2: ID = 0x02, flags = RDGT
0x05, 0x01                     // block #3: ID = 0x05, flags = WNGT
0x06, 0x01                     // block #4: ID = 0x06, flags = WNGT

0xxx                           // checksum
0xF7                           // footer
```

Example: GetParmVal (Ethernet port 1)

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x0E                     // message length (15)
0x03, 0x04                     // message class, data class
0x02                           // NumDataBlock: (2)

0x07, 0x04                     // data block #1: size (7), type (ArgVal)
0x02                           // NumArgValBlock: (2)
0x03                           // block #1: ArgID: hardware port type (3)
0x05                           // block #1: ArgVal: Ethernet (5)
0x04                           // block #2: ArgID: hardware port ID (4)
0x01                           // block #2: ArgVal: (1)

0x05, 0x01                     // data block #3: size (5), type (ParmList)
0x02                           // NumParmID: (2)
0x30                           // ParmID #1:  EthMACAddress
0x32                           // ParmID #2:  EthCurrentIP
```

```
0xxx                          // checksum
0xF7                          // footer
```

Example: RetParmVal (Ethernet port 1)

```
0xF0, 0x00, 0x01, 0x73, 0x7D  // header
0x00, 0x05                    // product ID
0x01, 0x02, 0x03, 0x04, 0x05  // serial number
0x00, 0x00, 0x00, 0x00        // session ID
0x00, 0x00, 0x00, 0x00        // transaction ID
0x00, 0x2C                    // message length (44)
0x43, 0x04                    // message class, data class
0x02                          // NumDataBlock: (2)

0x07, 0x04                    // data block #1: size (7), type (ArgVal)
0x02                          // NumArgValBlock: (2)
0x03                          // block #1: ArgID: hardware port type (3)
0x05                          // block #1: ArgVal: Ethernet (5)
0x04                          // block #2: ArgID: hardware port ID (4)
0x01                          // block #2: ArgVal: (1)

0x22, 0x03                    // data block #2: size (34), type (ParmVal)
0x02                          // NumParmValBlock: (2)
0x0E                          // block #1: size (14)
0x30                          // block #1: parm ID (EthMACAddress)
0x06, 0x05, 0x04, 0x03,       // Ethernet MAC address AC:7A:42:12:34:56
0x02, 0x01, 0x02, 0x04,
0x0A, 0x07, 0x0C, 0x0A
0x11                          // block #2: size (17)
0x32                          // block #2: parm ID (EthCurrentIP)
0x0A, 0x4F, 0x78, 0x00, 0x08  // current IP address (169.254.0.8)
0x0F, 0x7F, 0x7C, 0x00, 0x00  // current subnet mask (255.255.0.0)
0x0A, 0x4F, 0x78, 0x00, 0x01  // current gateway address (169.254.0.1)

0xxx                          // checksum
0xF7                          // footer
```

Example:SetParmVal (Ethernet port 1)

```
0xF0, 0x00, 0x01, 0x73, 0x7D  // header
0x00, 0x05                    // product ID
0x01, 0x02, 0x03, 0x04, 0x05  // serial number
0x00, 0x00, 0x00, 0x00        // session ID
0x00, 0x00, 0x00, 0x00        // transaction ID
0x00, 0x21                    // message length (33)
0x10, 0x04                    // message class, data class
0x02                          // NumDataBlock: (2)

0x07, 0x04                    // data block #1: size (7), type (ArgVal)
0x02                          // NumArgValBlock: (2)
0x03                          // block #1: ArgID: hardware port type (3)
0x05                          // block #1: ArgVal: Ethernet (5)
0x04                          // block #2: ArgID: hardware port ID (4)
0x01                          // block #2: ArgVal: (1)

0x17, 0x03                    // data block #2: size (23), type (ParmVal)
```

```
0x02                            // NumParmValBlock: (2)
0x03                            // block #1: size (3)
0x34                            // block #1: parm ID (EthIPMode)
0x00                            // IPMode: static
0x11                            // block #2: size (17)
0x35                            // block #2: parm ID (EthStaticIP)
0x0C, 0x05, 0x20, 0x02, 0x64    // static IP address (192.168.1.100)
0x0F, 0x7F, 0x7F, 0x7E, 0x00    // static subnet mask (255.255.255.0)
0x0C, 0x05, 0x20, 0x02, 0x01    // static gateway address (192.168.1.1)

0xxx                            // checksum
0xF7                            // footer
```

5.5 MIDIInfo (Data Class = 0x05)

This data class contains global parameters related to the MIDI capabilities and configuration of a device.

Message Class Support:

Message Class	Notes
GetParmDef RetParmDef	
GetParmVal SetParmVal RetParmVal NotParmVal	No argument value block. All parameters are global and do not support scenes.

Applicable Arguments:

ArgID	Name	Description
0x01	Area	Area selection (0 – N), optional. Only applies to devices that support areas. 0 is used for the work area, 1 – N is used for the shadow areas. Work area is used if this argument is not supplied.

Parameters:

ID	Name	Description
0x01	PortCount	Total number of MIDI ports supported by device (1 – N) (1 byte).
0x02	DINPortCount	Number of DIN MIDI ports supported by device (0 – N) (1 byte).
0x03	CtrlPortCount	Number of (internal) MIDI control ports supported by device (0 – N) (1 byte).

ID	Name	Description
0x04	USBDPortCount	Number of USBD MIDI ports supported by device (0 – N) (1 byte).
0x05	USBHPortCount	Number of USBH MIDI ports supported by device (0 – N) (1 byte).
0x06	EthPortCount	Number of RTP-MIDI sessions supported by device (0 – N) (1 byte).
0x07	MIDIPortName Max	Maximum length allowed for MIDI port name (MIDIPortInfo:PortNameIn & PortNameOut) (1 byte).
0x08	USBDPortName Max	Maximum length allowed for USB device MIDI port name (MIDIPortInfo:USBDPortName) (1 byte).
0x09	EthSesnName Max	Maximum length allowed for Ethernet session name (MIDIPortInfo:EthSesnName) and remote session name (MIDIPortInfo:EthSesnNameR) (1 byte).
0x0A	PortFeatureFlags	MIDI port feature flags (1 byte): bits 7 - 4: reserved (always 0). bit 3: set if AMP is supported. bit 2: set if MIDI remap for channel messages is supported. bit 1: set if MIDI filter for channel messages is supported. bit 0: set if MIDI filter for system messages is supported.
0x0B	AMPAlgMax	Maximum number of AMP algorithms supported by device (1 byte).
0x0C	AMPOpMax	Maximum number of AMP operators supported by device (1 byte).
0x0D	AMPCRMMax	Maximum number of AMP custom route maps supported by device (1 byte).
0x0E	AMPLUTMax	Maximum number of AMP lookup tables supported by device (1 byte).
0x0F	AMPOPAMax	Maximum number of AMP operators-per-algorithm supported by device (1 byte).
0x10	AMPAlg NameMax	Maximum length allowed for AMP algorithm name (1 byte).
0x11	AMPAlg UserDataMax	Max length allowed for AMP algorithm user data (1 byte).
0x12	PortMonitorIn	Bitmap indicating activity on MIDI inputs (length depends on the number of MIDI ports). See below.
0x13	PortMonitorOut	Bitmap indicating activity on MIDI outputs (length depends on the number of MIDI ports). See below.

PortMonitorIn and PortMonitorOut Parameter Block:

Bitmap indicating MIDI activity on all ports (BAx2 format). Bit 0 is port #1, bit 1 is port #2, etc. Least significant byte is first, most significant byte is last. The most significant 4 bits of each byte must be 0 which allows 4 ports to be specified per byte. Unused bits should be set to 0. The number of ports is

given in the MIDIInfo:PortCount parameter. The number of bytes (in the bitmap) is an even value so that the bitmap (when unpacked) is a multiple of 8 bits:

number of bytes (using INTEGER math) = (((number of MIDI ports - 1) / 8) + 1) x 2

Example (device has 20 ports):

```
0x01    // MIDI activity for ports 4 through 1 (port 1 has activity)
0x04    // MIDI activity for ports 8 through 5 (port 7 has activity)
0x0C    // MIDI activity for ports 12 through 9 (ports 12 and 11 have activity)
0x03    // MIDI activity for ports 16 through 13 (ports 14 and 13 have activity)
0x08    // MIDI activity for ports 20 through 17 (port 20 has activity)
0x00    // MIDI activity for ports 24 through 21 (padding)
```

5.6 MIDIPortInfo (Data Class = 0x06)

This data class contains parameters related to specific MIDI ports. MIDI port ID is a required argument for all parameters in this data class. MIDI channel number is a required argument for the FilterChannelIn, FilterChannelOut, RemapChannelIn, and RemapChannelOut parameters.

Message Class Support:

Message Class	Notes
GetParmDef RetParmDef	
GetParmVal SetParmVal RetParmVal NotParmVal	Argument value block is required. Some parameters can be specified for individual scenes.

Applicable Arguments:

ArgID	Name	Description
0x01	AreaID	Area selection (0 – N), optional. Only applies to devices that support areas. 0 is used for the work area, 1 – N is used for the shadow areas. Work area is used if this argument is not supplied.
0x02	SceneID	Scene selection (0 – N). Only applies to devices and parameters that support scenes. 0 is used for the currently active scene, 1 – N is used to specify a particular scene. Use 0 for globals and parameters that do not support scenes.
0x05	MIDIPortID	MIDI port ID (1 – N). Required for all parameters.
0x06	MIDIChannel	MIDI channel number (1 – 16). Required only for the FilterChannelIn, FilterChannelOut, RemapChannelIn, and RemapChannelOut parameters. Channel 1 is used if this argument is not supplied.

MIDI Port Types:

Value	Description
0x01	DIN
0x02	USB Device
0x03	USB Host Controller
0x04	Ethernet
0x05	Control (see Control Port Types table below)

Control Port Types:

Value	Description
0x01	Preset & Scene Selector
0x02	Failover Control
0x03	Automation Control

Parameters for All MIDI Port Types:

ID	Name	Description
0x01	PortType	MIDI port type (1 byte), see MIDI Port Types table.
0x02	PortIdentifier	MIDI port identifier (2 bytes), depends on PortType: DIN: Byte 1: DIN IN port ID (1 – N, 0 if port has no hardware DIN IN port). Byte 2: DIN OUT port ID (1 – N, 0 if port has no hardware DIN OUT port). USB Device: Byte 1: USB device port ID (1 – N). Byte 2: MIDI port number (1 – 16). USB Host Controller : Byte 1: USB host controller ID (1 – N). Byte 2: USB host controller port (1 – N). Ethernet : Byte 1: Ethernet port (1 - N). Byte 2: RTP-MIDI session number (1 - N). Control: Byte 1: Control Port ID (1 – N). Byte 2: Control Port Type (1 – N), see Control Port Types table.

ID	Name	Description
0x03	PortConnectFlags	MIDI port connection flags (1 byte): bits 7 - 1: reserved (always 0). bit 0: set if this port has an active connection. Active USB device ports return 1 for bit 0 if a host is connected. Active USB host ports return 1 for bit 0 if a device is connected. Active Ethernet ports return 1 for bit 0 if a session is connected. Active DIN and Control ports always return 1 for bit 0. Inactive ports (all types) always return 0 for bit 0.
0x04	PortActiveFlags	MIDI port active flags (1 byte): bits 7 - 1: reserved (always 0). bit 0: set if this port is active. MIDI ports can be made inactive by limiting the number of MIDI ports available on USB device, USB host controller, and Ethernet ports using the HardwareInfo parameters USBDMIDIPortCount, USBHMIDI PortCount and EthMIDIPortCount respectively. For these types of ports, writing to bit 0 is not supported. For DIN and Control ports, bit 0 can be cleared to make the port inactive. An inactive port still exists inside the device but is non-functional and is not visible to the host via USB or Ethernet. A host application should not display inactive ports to the user (they should be considered non-existent).
0x05	PortSupportFlags	MIDI port support flags (1 byte): bits 7 - 3: reserved (always 0). bit 2: set if this port supports running status on MIDI output. bit 1: set if this port supports MIDI output. bit 0: set if this port supports MIDI input.
0x06	PortEnableFlags	MIDI port enable flags (1 byte). These bits can only be set if the matching bit is set in PortSupportFlags. bits 7 - 3: reserved (always 0). bit 2: set if running status is enabled on MIDI output. bit 1: set if the MIDI output port is enabled. bit 0: set if the MIDI input port is enabled.
0x07	PortRoute	Bitmap indicating routing for MIDI events entering at the input for this port (BAx2 format). Each bit represents a MIDI port output. Bit 0 is routing from this port to port #1, bit 1 is routing from this port to port #2, etc. Least significant byte is first, most significant byte is last. The most significant 4 bits of each byte must be 0 which allows 4 ports to be specified per byte. Unused bits should be set to 0. The number of MIDI ports is given in the MIDIInfo:PortCount parameter. The number of bytes (in the bitmap) is an even value so that the bitmap (when unpacked) is a multiple of 8 bits: number of bytes (using INTEGER math) = (((number of MIDI ports - 1) / 8) + 1) x 2. Example: device has 20 ports, route MIDI events coming into this port to ports 2, 3, 7, 11-14, 20:

ID	Name	Description
		0x06 // routing for ports 4 through 1 (ports 3 and 2 are selected) 0x04 // routing for ports 8 through 5 (port 7 is selected) 0x0C // routing for ports 12 through 9 (ports 12 and 11 are selected) 0x03 // routing for ports 16 through 13 (ports 14 and 13 are selected) 0x08 // routing for ports 20 through 17 (port 20 is selected) 0x00 // routing for ports 24 through 21 (padding)
0x08	PortFeatureFlags In	MIDI port feature enable flags for MIDI input (1 byte). These bits can only be set if the matching bit is set in MIDIInfo:PortFeatureFlags: bits 7 - 4: reserved (always 0). bit 3: set if AMP is enabled. bit 2: set if MIDI remap for channel messages is enabled. bit 1: set if MIDI filter for channel messages is enabled. bit 0: set if MIDI filter for system messages is enabled.
0x09	PortFeatureFlags Out	MIDI port feature enable flags for MIDI output (1 byte). These bits can only be set if the matching bit is set in MIDIInfo:PortFeatureFlags: bits 7 - 4: reserved (always 0). bit 3: set if AMP is enabled. bit 2: set if MIDI remap for channel messages is enabled. bit 1: set if MIDI filter for channel messages is enabled. bit 0: set if MIDI filter for system messages is enabled.
0x0A	PortNameIn	MIDI port name to use for incoming MIDI events (7-bit ASCII string). Maximum length is given by MIDIInfo:MIDIPortNameMax.
0x0B	PortNameOut	MIDI port name to use for outgoing MIDI events (7-bit ASCII string). Maximum length is given by MIDIInfo:MIDIPortNameMax.
0x0C	FilterSystemIn	Filter settings for MIDI system messages on MIDI input (2-4 bytes). See below.
0x0D	FilterSystemOut	Filter settings for MIDI system messages on MIDI output (2-4 bytes). See below.
0x0E	FilterChannelIn	Filter settings for MIDI channel messages on MIDI input (2 bytes). See below. MIDI Channel is a required ArgVal for this parameter
0x0F	FilterChannelOut	Filter settings for MIDI channel messages on MIDI output (2 bytes). See below. MIDI Channel is a required ArgVal for this parameter
0x10	RemapChannelIn	Remap settings for MIDI channel message on MIDI input (2-14 bytes). See below. MIDI Channel is a required ArgVal for this parameter.
0x11	RemapChannel Out	Remap settings for MIDI channel message on MIDI output (2-14 bytes). See below. MIDI Channel is a required ArgVal for this parameter.
0x12	AMPAlgorithmIn	AMP algorithm ID for MIDI input (0 – N) (1 byte). Use 0 for no algorithm.
0x13	AMPAlgorithmOut	AMP algorithm ID for MIDI output (0 – N) (1 byte). Use 0 for no algorithm.

FilterSystemIn and FilterSystemOut Parameter Block:

Each block contains a set of sub-ID & value pairs. RetParmVal messages always include all sub-IDs (4 bytes total). SetParmVal messages can include both sub-IDs (4 bytes total) or just one sub-ID (2 bytes total).

Sub-ID	Description
0x01	MIDI system message filter flags #1 (1 byte): bit 7: always 0. bit 6: set if port should filter MIDI Song Select events (0xF3). bit 5: set if port should filter MIDI Song Position Pointer events (0xF2). bit 4: set if port should filter MIDI Time Code Quarter Frame events (0xF1). bit 3: set if port should filter MIDI Stop events (0xFC). bit 2: set if port should filter MIDI Continue events (0xFB). bit 1: set if port should filter MIDI Start events (0xFA). bit 0: set if port should filter MIDI Timing Clock events (0xF8).
0x02	MIDI system message filter flags #2 (1 byte): bits 7 - 4: always 0. bit 3: set if port should filter MIDI System Exclusive events (0xF0). bit 2: set if port should filter MIDI Reset events (0xFF). bit 1: set if port should filter MIDI Active Sensing events (0xFE). bit 0: set if port should filter MIDI Tune Request events (0xF6).

FilterChannelIn and FilterChannelOut Parameter Block:

Each block contains a set of sub-ID & value pairs for MIDI channel message filtering. One block is used for each of the 16 MIDI channels (using the MIDI Channel Argval). RetParmVal messages always return all sub-IDs for the selected MIDI channel (always 2 bytes since there is only one sub-ID). SetParmVal messages can send any subset of the sub-IDs for the selected MIDI channel (always 2 bytes since there is only one sub-ID).

Sub-ID	Description
0x01	MIDI channel message filter flags (1 byte): bit 7: always 0. bit 6: set if port should filter MIDI Pitch Bend events (0xEn). bit 5: set if port should filter MIDI Channel Pressure events (0xDn). bit 4: set if port should filter MIDI Program Change events (0xCn). bit 3: set if port should filter MIDI Control Change events (0xBn). bit 2: set if port should filter MIDI Poly Key Pressure events (0xAn). bit 1: set if port should filter MIDI Note On events (0x9n). bit 0: set if port should filter MIDI Note Off events (0x8n). Note that enabling the filter for Control Change messages will filter out all controllers on that MIDI channel; to filter out specific controllers use AMP.

RemapChannelIn and RemapChannelOut Parameter Block:

Each block contains a set of sub-ID & value pairs to select which MIDI channel messages are to be remapped and the MIDI channel to use for the modified message. One block is used for each of the 16 MIDI channels (using the MIDI Channel ArgVal). RetParmVal messages always return all sub-IDs for the selected MIDI channel (14 bytes). SetParmVal messages can send any subset of the sub-IDs for the selected MIDI channel (e.g. 2 bytes for one remap, 14 bytes for all remaps).

Sub-ID	Description
0x01	MIDI channel number for remapping MIDI Note Off events (0x8n) (1 – 16) (1 byte).
0x02	MIDI channel number for remapping MIDI Note On events (0x9n) (1 – 16) (1 byte).
0x03	MIDI channel number for remapping MIDI Poly Key Pressure events (0xAn) (1 – 16) (1 byte).
0x04	MIDI channel number for remapping MIDI Control Change events (0xBn) (1 – 16) (1 byte). Note that enabling remapping for Control Change messages will remap all controllers on that MIDI channel; to remap specific controllers use AMP.
0x05	MIDI channel number for remapping MIDI Program Change events (0xCn) (1 – 16) (1 byte).
0x06	MIDI channel number for remapping MIDI Channel Pressure events (0xDn) (1 – 16) (1 byte).
0x07	MIDI channel number for remapping MIDI Pitch Bend events (0xEn) (1 – 16) (1 byte).

Parameters for DIN Ports:

ID	Name	Description
	none	

Parameters for USB Device Ports:

ID	Name	Description
0x1F	USBDPortName	USBD port name (7-bit ASCII string). Maximum length is given by MIDIInfo:USBDPortNameMax.

Parameters for USB Host Controller Ports:

These parameters provide information about USB-MIDI devices on a USB host controller port that are connected to a MIDI port. To find all USB-MIDI devices connected to a host comtroller port, regardless of whether or not they are connected to a MIDI port, use the parameters in the HardwareInfo section for USB host controller MIDI ports.

ID	Name	Description
0x20	USBHVID	Hosted device's USB vendor ID (16-bit value encoded in 3 bytes, 16x3 format). 0 if no device is hosted.
0x21	USBHPID	Hosted device's USB product ID (16-bit value encoded in 3 bytes, 16x3 format). 0 if no device is hosted.
0x22	USBHVName	Hosted device's vendor name from USB descriptor (7-bit ASCII string).
0x23	USBHPName	Hosted device's product name from USB descriptor (7-bit ASCII string).
0x24	USBHSerialNum	Hosted device's serial number from USB descriptor (7-bit ASCII string). Most USB-MIDI devices do not have a serial number in the USB descriptor.
0x25	USBHPort CountIn	Number of MIDI IN ports supported by the hosted device (1 – 16) (1 byte). 0 if no device is hosted.
0x26	USBHPort CountOut	Number of MIDI OUT ports supported by the hosted device (1 – 16) (1 byte). 0 if no device is hosted.
0x27	USBHIdentifier	A unique ID assigned to each hosted device (1 – N) (1 byte). This can be used to distinguish between multiple devices that have the same USBHVID and USBHPID with a blank USBHSerialNum. 0 if no device is hosted.
0x28	USBHPortNum	MIDI port number on the hosted device (1 – 16) (1 byte). 0 if no device is hosted.
0x29	USBHReserve	Indicates if the port is reserved for a specific USB-MIDI device (1 byte). Devices are reserved using USBHVIDR, USBHPIDR, USBHPortNumR, and (optionally) USBHSerialNumR. 0 = port is not reserved 1 = port is reserved
0x2A	USBHVIDR	Reserved device's USB vendor ID (16-bit value encoded in 3 bytes, 16x3 format). Value should always be copied from USBHVID.
0x2B	USBHPIDR	Reserved device's USB vendor ID (16-bit value encoded in 3 bytes, 16x3 format). Value should always be copied from USBHPID.
0x2C	USBHVNameR	Reserved device's vendor name from USB descriptor (7-bit ASCII string). Value should always be copied from USBHVName. This parameter is information only and is not used for reserving a device.
0x2D	USBHPNameR	Reserved device's product name from USB descriptor (7-bit ASCII string). Value should always be copied from USBHPName. This parameter is information only and is not used for reserving a device.
0x2E	USBHSerialNumR	Reserved device's serial number from USB descriptor (7-bit ASCII string). Most USB-MIDI devices do not have a serial number in the USB descriptor. Set this parameter to an empty string to not use serial number when searching for reserved devices. Value should either be an empty string or copied from USBHSerialNum.

ID	Name	Description
0x2F	USBHPortNumR	MIDI port number on the reserved device (1 – 16) (1 byte).

Parameters for Ethernet Ports:

ID	Name	Description
0x30	EthSesnFlags	Session configuration flags (1 byte): bits 7 - 3: reserved (always 0). bit 2: initiator mode: 0 = use IP address & port number (EthIPAddressR & EthPortNumberR), 1 = use session name (EthSesnNameR). bit 1: role: 0 = responder, 1 = initiator. bit 0: enable: 0 = disabled, 1 = enabled.
0x31	EthPortNumber	Port number used for this session (16-bit value encoded in 3 bytes, 16x3 format).
0x32	EthSesnName	Session name (Bonjour name) to use for this session on the network. Actual name used on the network (EthSesnNameN) may be different following Bonjour name resolution. Maximum length is given by MIDIInfo:EthSesnNameMax.
0x33	EthSesnNameN	Actual session name (Bonjour name) for this session on the network (7-bit ASCII string). Only valid if the Ethernet port is connected to a network, is active, and has finished Bonjour name resolution.
0x34	EthIPAddressX	IP address of the device on the other side of this connection (four-byte big-endian value encoded in five-bytes, 32x5 format). Only valid if the session has an active connection. Example: 0x0C, 0x05, 0x20, 0x02, 0x64 // IP address (192.168.1.100)
0x35	EthPortNumberX	Port number for the device on the other side of this connection (16-bit value encoded in 3 bytes, 16x3 format). Only valid if the session has an active connection.
0x36	EthSesnNameX	Session name (Bonjour name) for the device on the other side of this connection (7-bit ASCII string). Only valid if the session has an active connection.
0x37	EthIPAddressR	IP address to use when initiating a connection (four-byte big-endian value encoded in five-bytes, 32x5 format). Only used when EthSesnFlags role = 1 and initiator mode = 0. Example: 0x0C, 0x05, 0x20, 0x02, 0x64 // IP address (192.168.1.100)
0x38	EthPortNumberR	Port number to use when initiating a connection (16-bit value encoded in 3 bytes, 16x3 format). Only used when EthSesnFlags role = 1 and initiator mode = 0.

ID	Name	Description
0x39	EthSesnNameR	Session name (Bonjour name) to use when initiating a connection (7-bit ASCII string). Only used when EthSesnFlags role = 1 and initiator mode = 1. Maximum length is given by MIDIInfo:EthSesnNameMax.

Parameters for Preset & Scene Selector Control Ports:

ID	Name	Description
0x40	PresetSelector	Preset/Scene selector parameter block. See notes below.
0x41	SceneSelector	Preset/Scene selector parameter block. See notes below.

PresetSelector and SceneSelector Parameter Block:

Preset & Scene selector control ports can be configured to support either MIDI Program Change (0xCn) or MIDI Control Change (0xBn) events. When using Program Change events, byte #2 of the MIDI event is used to select the preset/scene number. When using Control Change events, byte #3 of the MIDI event is used to select the preset/scene number. For Control Change events, the controller number must be specified as well as the control type (continuous or pushbutton) and the range of controller values to use (i.e. byte #3 in the MIDI event). For continuous controls, the values between *MIDI minimum* and *MIDI maximum* are equally distributed among the presets/scenes so that the *MIDI minimum* value selects preset #1 and the *MIDI maximum* value selects the highest preset number. For pushbutton controls, the *MIDI on* value defines the transition point between the first and second preset/scene only (all other presets/scenes are ignored). For example, if *MIDI on* is 0x40 then values between 0x00 to 0x3F will select scene #1 and values between 0x40 and 0x7F will select scene #2. An invert flag can be used to reverse the logic so that values between 0x00 and 0x3F will select scene #2 and values between 0x40 and 0x7F will select scene #1. The invert flag can also be used for continuous controls so that the *MIDI maximum* value will select preset #1 and the *MIDI minimum* value will select the highest preset number.

Sub-ID	Description
0x01	Selector configuration flags (1 byte): bits 7-4: reserved (always 0). bit 3: set if selector should invert the logic when using pushbutton control. Only used if bit 1 = 1 and bit 2 = 0. bit 2: set if selector should use continuous control, clear if selector should use pushbutton control. Only used if bit 1 = 1. bit 1: set if selector should use MIDI Control Change event (0xBn), clear if selector should use MIDI Program Change event (0xCn). bit 0: set if selector is enabled, clear if selector is disabled.
0x02	MIDI channel number bitmask indicating which MIDI channels to use for selector (0x0001 = MIDI channel 1, 0x0002 = MIDI channel 2 … 0x8000 = MIDI channel 16). More than 1 bit can be set (16-bit value encoded in 3 bytes, 16x3 format).
0x03	MIDI Control Change event number (0x00 – 0x7F) (1 byte). Only used if selector configuration

Sub-ID	Description
	flags bit 1 = 1.
0x04	MIDI minimum value (0x00 – 0x7F) (1 byte). Only used if selector configuration flags bit 1 = 1 and bit 2 = 1. Value must be lower than MIDI maximum value. Values less than MIDI minimum will be treated as MIDI minimum.
0x05	MIDI maximum value (0x00 – 0x7F) (1 byte). Only used if selector configuration flags bit 1 = 1 and bit 2 = 1. Value must be higher than MIDI minimum value. Values greater than MIDI maximum will be treated as MIDI maximum.
0x06	MIDI on value (0x00 – 0x7F) (1 byte). Only used if selector configuration flags bit 1 = 1 and bit 2 = 0. Values lower than MIDI on will be treated as OFF. Values greater than or equal to MIDI on will be treated as ON. Use selector configuration flags bit 2 to inverse the logic.

5.7 MIDIFeature (Data Class = 0x07)

This data class contains parameters for advanced MIDI features. All parameters in this data class require an argument value block.

Message Class Support:

Message Class	Notes
GetParmDef RetParmDef	
GetParmVal SetParmVal RetParmVal NotParmVal	Argument value block is required. All parameters are global and do not support scenes.

5.7.1 Advanced MIDI Processor (AMP)

Applicable Arguments:

ArgID	Name	Description
0x01	AreaID	Area selection (0 – N), optional. Only applies to devices that support areas. 0 is used for the work area, 1 – N is used for the shadow areas. Work area is used if this argument is not supplied.
0x07	AMPID	AMP ID (1 – N). This specifies the algorithm ID, the operator ID, the custom route map ID, or the lookup table ID (depends on the parameter ID).

Parameters for AMP Algorithms:

AMP algorithm parameters require an AMPID argument value block to specify the algorithm ID.

ID	Name	Description
0x01	AMPAlgName	AMP algorithm name (7-bit ASCII string).
0x02	AMPAlgUserData	AMP algorithm user data. Free field for storing user data (all bytes must be in the range 0x00 – 0x7F). Bytes 1: index (i.e. starting address) Bytes 2-N: user data The first byte is an index into the user data field (i.e. the starting address). The remaining bytes are the user data. RetParmVal will always return the entire user data field (index is always 0, user data is always MIDIInfo:AMPAlgUserDataMax bytes in length). SetParmVal can use a non-zero value for the offset byte and can write just a portion of the user data field.

Parameters for AMP Operators:

AMP operator parameters require an AMPID argument value block to specify the operator ID.

AMP operators have one input pin and three output pins (Y, N, M). The Y and N output pins are from the output of the Match function. The M output pin is from the output of the Modify function. The Match function input is always connected to the input pin. The Modify function input can be connected to the Y or N output pins. Operators are assigned to algorithms, up to a maximum of N operators per algorithm (see MIDIInfo:AMPOPAMax).

If the Match function uses 2x8 mode (see Match Header Flags in AMPOpMatchHeaderBlock) then two instances of the Match function 2x8 block are used: one for byte 2 in the MIDI event (AMPOpMatch2x8b2), and another for byte 3 in the MIDI event (AMPOpMatch2x8b3). If the Match function uses 1x16 mode then one instance of the Match function 1x16 block is used (AMPOpMatch1x16).

If the Modify function uses 2x8 mode (see Modify Header Flags in AMPOpModifyHeaderBlock) then two instances of the Modify function 2x8 block are used: one for byte 2 in the MIDI event (AMPOpModify2x8b2), and another for byte 3 in the MIDI event (AMPOpModify2x8b3). If the Modify function uses 1x16 mode then one instance of the Modify function 1x16 block is used (AMPOpModify1x16).

ID	Name	Description
0x03	AMPOp Connection	Operator connection block. See notes below.
0x04	AMPOpMatch Header	Operator match function header block. See notes below.

ID	Name	Description
0x05	AMPOpMatch 2x8b2	Operator match function 2x8 block for byte 2. See notes below.
0x06	AMPOpMatch 2x8b3	Operator match function 2x8 block for byte 3. See notes below.
0x07	AMPOpMatch 1x16	Operator match function 1x16 block. See notes below.
0x08	AMPOpModify HeaderBlock	Operator modify function header block. See notes below.
0x09	AMPOpModify 2x8b2	Operator modify function 2x8 block for byte 2. See notes below.
0x0A	AMPOpModify 2x8b3	Operator modify function 2x8 block for byte 3. See notes below.
0x0B	AMPOpModify 1x16	Operator modify function 1x16 block. See notes below.

Operator Connection Block:

Each block contains a set of sub-ID & value pairs. RetParmVal messages always include all sub-IDs SetParmVal messages can include any selection of sub-IDs.

Sub-ID	Description
0x01	Algorithm ID (1 – N) that this operator is assigned to (1 byte). Use 0 if this operator is not assigned to any algorithm.
0x02	Connection Flags (1 byte): bits 7 - 2: always zero bit 1: set if the N output pin is connected to the Modify function input. bit 0: set if the Y output pin is connected to the Modify function input.
0x03	Input pin connection for this operator (1 byte): 0: input pin is not connected to anything. 1 – N: input pin connects to operator output with this ID. 0x7F: input pin connects to algorithm Entry point.
0x04	Pin ID of the operator output that connects to the input of this operator (only used if sub-ID 0x03 is not 0 or 0x7F) (1 byte): 0: no operator output pin is connected to the input of this operator. 1: the Y output pin of an operator is connected to the input of this operator. 2: the N output pin of an operator is connected to the input of this operator. 3: the M output pin of an operator is connected to the input of this operator.
0x05	MIDI channel to use for incoming MIDI event when the M output of another operator is connected

Sub-ID	Description
	to the input of this operator (1 byte). Only used if sub-ID 0x03 is a valid operator ID and sub-ID 0x04 has the value 3. 1 = MIDI channel 1, 2 = MIDI channel 2 … 16 = MIDI channel 16. Use 0 if incoming MIDI event should use the same MIDI channel as the original event.
0x06	Output Order (1 byte). Sets the order of MIDI events from Y, N, and M outputs to algorithm Exit point. 0: YNM (Y output first, N output second, M output third). 1: YMN (Y output first, M output second, N output third). 2: NYM (N output first, Y output second, M output third). 3: NMY (N output first, M output second, Y output third). 4: MYN (M output first, Y output second, N output third). 5: MNY (M output first, N output second, Y output third).
0x07	Y output pin connection of this operator to algorithm Exit point (1 byte): 0: Y output pin does not connect to algorithm Exit point. 1-N: Y output pin connects to algorithm Exit point via custom route map with this ID. 0x7F: Y output pin connects to algorithm Exit point via default route map.
0x08	N output pin connection of this operator to algorithm Exit point (1 byte): 0: N output pin does not connect to algorithm Exit point. 1-N: N output pin connects to algorithm Exit point via custom route map with this ID. 0x7F: N output pin connects to algorithm Exit point via default route map.
0x09	M output pin connection of this operator to algorithm Exit point (1 byte): 0: M output pin does not connect to algorithm Exit point. 1-N: M output pin connects to algorithm Exit point via custom route map with this ID. 0x7F: M output pin connects to algorithm Exit point via default route map.

Operator Match Function Header Block:

Each block contains a set of sub-ID & value pairs. RetParmVal messages always include all sub-IDs SetParmVal messages can include any selection of sub-IDs.

Sub-ID	Description
0x01	Match Header Flags (1 byte): bits 7 - 1: always zero bit 0: set if Match function uses 1x16 mode, clear if Match function uses 2x8 mode.
0x02	Match Event Type (1 byte). Bitmask indicating which MIDI event types to match. More than 1 bit can be set: bit 7: always 0 bit 6: Pitch Bend (0xEn, 3-byte MIDI event) bit 5: Mono Aftertouch (0xDn, 2-byte MIDI event) bit 4: Program Change (0xCn, 2-byte MIDI event) bit 3: Control Change (0xBn, 3-byte MIDI event) bit 2: Poly Aftertouch (0xAn, 3-byte MIDI event) bit 1: Note On (0x9n, 3-byte MIDI event) bit 0: Note Off (0x8n, 3-byte MIDI event)

Sub-ID	Description
0x03	Match Channel: Bitmask indicating which MIDI channels to match (0x0001 = MIDI channel 1, 0x0002 = MIDI channel 2 … 0x8000 = MIDI channel 16). More than 1 bit can be set (16-bit value encoded in 3 bytes, 16x3 format).

Operator Match Function 2x8 Block:

Each block contains a set of sub-ID & value pairs. RetParmVal messages always include all sub-IDs SetParmVal messages can include any selection of sub-IDs. If Match 2x8 Flags (sub-ID 0x01) is included, it should always be the first sub-ID in the message. This parameter is only used if Match Header Flags bit 0 is clear.

Sub-ID	Description
0x01	Match 2x8 Flags (1 byte): bits 7 - 2: always zero bit 1: set if a lookup table is used, clear if minimum value and maximum value are used. bit 0: set if Match function logic is inverted.
0x02	Lookup table ID (1 – N) (1 byte). Only used if bit 1 is set in Match 2x8 Flags. Set to 0 if unused.
0x03	Lookup table value that is used for matching (0x00 – 0x7F) (1 byte). Only used if bit 1 is set in Match 2x8 Flags.
0x04	Minimum value that is used for matching (0x00 – 0x7F) (1 byte). Must be less than or equal to the maximum value (sub-ID 0x05). Only used if bit 1 is clear in Match 2x8 Flags.
0x05	Maximum value that is used for matching (0x00 – 0x7F) (1 byte). Must be greater than or equal to the minimum value (sub-ID 0x04). Only used if bit 1 is clear in Match 2x8 Flags.

Operator Match Function 1x16 Block:

Each block contains a set of sub-ID & value pairs. RetParmVal messages always include all sub-IDs SetParmVal messages can include any selection of sub-IDs. If Match 1x16 Flags (sub-ID 0x01) is included, it should always be the first sub-ID in the message. This parameter is only used if Match Header Flags bit 0 is set.

Sub-ID	Description
0x01	Match 1x16 Flags (1 byte): bits 7 - 3: always zero bit 2: set if byte-3 (MSB) in MIDI event is used for lookup table offset, clear if byte-2 (LSB) in MIDI event is used for lookup table offset. bit 1: set if a lookup table is used, clear if minimum value and maximum value are used. bit 0: set if Match function logic is inverted.
0x02	Lookup table ID (1 – N) (1 byte). Only used if bit 1 is set in Match 1x16 Flags. Set to 0 if unused.

Sub-ID	Description
0x03	Lookup table value that is used for matching (0x00 – 0x7F) (1 byte). Only used if bit 1 is set in Match 1x16 Flags.
0x04	Minimum value that is used for matching (0x0000 – 0x3FFF, 14x2 format, 2 bytes). Must be less than or equal to the maximum value (sub-ID 0x05). Only used if bit 1 is clear in Match 1x16 Flags.
0x05	Maximum value that is used for matching (0x0000 – 0x3FFF, 14x2 format, 2 bytes). Must be greater than or equal to the minimum value (sub-ID 0x04). Only used if bit 1 is clear in Match 1x16 Flags.

Operator Modify Function Header Block:

Each block contains a set of sub-ID & value pairs. RetParmVal messages always include all sub-IDs SetParmVal messages can include any selection of sub-IDs.

Sub-ID	Description
0x01	Modify Header Flags (1 byte): bits 7 - 1: always zero bit 0: set if Modify function uses 1x16 mode, clear if Modify function uses 2x8 mode.
0x02	Modify In Event Type (1 byte). Bitmask indicating which MIDI event types to modify. More than 1 bit can be set: bit 7: always 0 bit 6: Pitch Bend (0xEn, 3-byte MIDI event) bit 5: Mono Aftertouch (0xDn, 2-byte MIDI event) bit 4: Program Change (0xCn, 2-byte MIDI event) bit 3: Control Change (0xBn, 3-byte MIDI event) bit 2: Poly Aftertouch (0xAn, 3-byte MIDI event) bit 1: Note On (0x9n, 3-byte MIDI event) bit 0: Note Off (0x8n, 3-byte MIDI event)
0x03	Modify In Channel: Bitmask indicating which MIDI channels to modify (0x0001 = MIDI channel 1, 0x0002 = MIDI channel 2 … 0x8000 = MIDI channel 16). More than 1 bit can be set (16-bit value encoded in 3 bytes, 16x3 format).
0x04	Modify Out Event Type (1 byte). Sets the output MIDI event type. Only 1 bit can be set. Use 0x00 to select the same event type as the input MIDI event. bit 7: always 0 bit 6: Pitch Bend (0xEn, 3-byte MIDI event) bit 5: Mono Aftertouch (0xDn, 2-byte MIDI event) bit 4: Program Change (0xCn, 2-byte MIDI event) bit 3: Control Change (0xBn, 3-byte MIDI event) bit 2: Poly Aftertouch (0xAn, 3-byte MIDI event) bit 1: Note On (0x9n, 3-byte MIDI event) bit 0: Note Off (0x8n, 3-byte MIDI event)
0x05	Modify Out Channel: Bitmask indicating which MIDI channels to use for outgoing MIDI events

Sub-ID	Description
	(0x0001 = MIDI channel 1, 0x0002 = MIDI channel 2 … 0x8000 = MIDI channel 16). More than 1 bit can be set (16-bit value encoded in 3 bytes, 16x3 format). Use 0 to select the same channel as the input MIDI event.

Operator Modify Function 2x8 Block:

Each block contains a set of sub-ID & value pairs. RetParmVal messages always include all sub-IDs SetParmVal messages can include any selection of sub-IDs. If Modify 2x8 Flags (sub-ID 0x01) is included, it should always be the first sub-ID in the message. This parameter is only used if Modify Header Flags bit 0 is clear.

Sub-ID	Description
0x01	Modify 2x8 Flags (1 byte): bits 7 - 3: always zero bit 2: set if Modify Default Value should be used when input is a 2-byte MIDI event. bit 1: set if this output byte (either byte 2 or byte 3) uses input byte 3, clear if this output byte (either byte 2 or byte 3) uses input byte 2. bit 0: set if a lookup table is used.
0x02	Lookup table ID (1 – N) (1 byte). Only used if bit 0 is set in Modify 2x8 Flags. Set to 0 if not used.
0x03	Minimum in value that is used by Modify function (0x00 – 0x7F) (1 byte). Must be less than or equal to the maximum in value (sub-ID 0x04).
0x04	Maximum in value that is used by Modify function (0x00 – 0x7F) (1 byte). Must be greater than or equal to the minimum in value (sub-ID 0x03).
0x05	Minimum out value that is used by Modify function (0x00 – 0x7F) (1 byte). Can be greater than maximum out value (sub-ID 0x06) to allow for inversion.
0x06	Maximum out value that is used by Modify function (0x00 – 0x7F) (1 byte). Can be less than minimum out value (sub-ID 0x05) to allow for inversion.
0x07	Minimum Default Mode (1 byte). Selects the processing behaviour for when the input byte value is less than Minimum In: 0: output value is set to input value, lookup table is not used. 1: output value is set to input value, lookup table is used if enabled. 2: output value is set to Minimum In value, lookup table is used if enabled. 3: output value is set to Minimum Default Value (sub-ID 0x08), lookup table is not used.
0x08	Minimum Default Value (1 byte). Value to use when Minimum Default Mode (sub-ID 0x07) is set to 3 (0x00 – 0x7F).
0x09	Maximum Default Mode (1 byte). Selects the processing behaviour for when the input byte value is greater than Maximum In: 0: output value is set to input value, lookup table is not used. 1: output value is set to input value, lookup table is used if enabled. 2: output value is set to Maximum In value, lookup table is used if enabled.

Sub-ID	Description
	3: output value is set to Maximum Default Value (sub-ID 0x0A), lookup table is not used.
0x0A	Maximum Default Value (1 byte). Value to use when Maximum Default Mode (sub-ID 0x09) is set to 3 (0x00 – 0x7F).
0x0B	Modify Default Value (1 byte). Value to use when output is a 3-byte MIDI event but input is a 2-byte MIDI event (0x00 – 0x7F).

Operator Modify function 1x16 block:

Each block contains a set of sub-ID & value pairs. RetParmVal messages always include all sub-IDs SetParmVal messages can include any selection of sub-IDs. If Modify 1x16 Flags (sub-ID 0x01) is included, it should always be the first sub-ID in the message. This parameter is only used if Modify Header Flags bit 0 is set.

Sub-ID	Description
0x01	Modify 1x16 Flags (1 byte): bit 7: always zero. bit 6: set if MSB of the result is used for byte-2, clear if LSB of the result is used for byte-2 (only used when output is 2-byte MIDI event). bit 5: set if byte-2 is used for MSB and 0x00 is used for LSB, clear if byte-2 is used for LSB and 0x00 is used for MSB (only used when input is 2-byte MIDI event). bit 4: set if using a fixed value for byte-3 (MSB) rather than a lookup table value. bit 3: set if using a fixed value for byte-2 (LSB) rather than a lookup table value. bit 2: set if input byte-3 (MSB) value is used, clear if 0x00 is used for input byte-3. bit 1: set if input byte-2 (LSB) value is used, clear if 0x00 is used for input byte-2. bit 0: set if a lookup table is used.
0x02	Lookup table ID (1 – N) for byte-2 (LSB) (1 byte). Only used if bit 0 is set and bit 3 is clear in Modify 1x16 Flags. Set to 0 if not used.
0x03	Lookup table ID (1 – N) for byte-3 (MSB) (1 byte). Only used if bit 0 is set and bit 4 is clear in Modify 1x16 Flags. Set to 0 if not used.
0x04	Byte-2 (LSB) fixed value to use instead of lookup table value (0x00 – 0x7F) (1 byte). Only used if bit 0 and bit 3 are both set in Modify 1x16 Flags.
0x05	Byte-3 (MSB) fixed value to use instead of lookup table value (0x00 – 0x7F) (1 byte). Only used if bit 0 and bit 4 are both set in Modify 1x16 Flags.
0x06	Minimum in value that is used by Modify function (0x0000 – 0x3FFF, 14x2 format, 2 bytes). Must be less than or equal to the maximum in value (sub-ID 0x07).
0x07	Maximum in value that is used by Modify function (0x0000 – 0x3FFF, 14x2 format, 2 bytes). Must be greater than or equal to the minimum in value (sub-ID 0x06).
0x08	Minimum out value that is used by Modify function (0x0000 – 0x3FFF, 14x2 format, 2 bytes). Can be greater than maximum out value (sub-ID 0x09) to allow for inversion.

Sub-ID	Description
0x09	Maximum out value that is used by Modify function (0x0000 – 0x3FFF, 14x2 format, 2 bytes). Can be less than minimum out value (sub-ID 0x08) to allow for inversion.
0x0A	Minimum Default Mode (1 byte). Selects the processing behaviour for when the input value is less than minimum in value: 0: output value is set to input value, lookup table is not used. 1: output value is set to input value, lookup table is used if enabled. 2: output value is set to Minimum in value, lookup table is used if enabled. 3: output value is set to Minimum Default Value (sub-ID 0x0B), lookup table is not used.
0x0B	Minimum Default Value, value to use when Minimum Default Mode (sub-ID 0x0A) is set to 3 (0x0000 – 0x3FFF, 14x2 format, 2 bytes).
0x0C	Maximum Default Mode. Selects the processing behaviour for when the input value is greater than maximum in value: 0: output value is set to input value, lookup table is not used. 1: output value is set to input value, lookup table is used if enabled. 2: output value is set to Maximum in value, lookup table is used if enabled. 3: output value is set to Maximum Default Value (sub-ID 0x0D), lookup table is not used.
0x0D	Maximum Default Value, value to use when Maximum Default Mode (sub-ID 0x0C) is set to 3 (0x0000 – 0x3FFF, 14x2 format, 2 bytes).

Example: single operator algorithm, Match and Modify functions both use 2x8 mode

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
0x00, 0x05                      // product ID
0x01, 0x02, 0x03, 0x04, 0x05    // serial number
0x00, 0x00, 0x00, 0x00          // session ID
0x00, 0x00, 0x00, 0x00          // transaction ID
0x01, 0x01                      // message length (129)
0x43, 0x07                      // message class, data class
0x02                            // NumDataBlock: (2)

0x05, 0x04                      // data block #1: size (5), type (ArgVal)
0x01                            // NumArgValBlock: (1)
0x07                            // ArgID: AMPID (7)
0x09                            // ArgVal: operator ID (9)

0x79, 0x03                      // data block #2: size (121), type (ParmVal)
0x07                            // NumParmValBlock: (7)

                                // ParmVal 1: Operator Connection Block
0x14                            // ParmSize: (20)
0x03                            // ParmID: (3)
0x01, 0x01                      // algorithm ID (1)
0x02, 0x01                      // flags: Y output pin is connected to Modify input
0x03, 0x7F                      // operator input is connected to algorithm Entry point (0x7F)
0x04, 0x00                      // no operator output pin is connected to the input of this operator
0x05, 0x00                      // MIDI channel for incoming event (not used)
0x06, 0x00                      // output order is YNM
```

```
0x07, 0x00                  // Y output pin does not connect to Exit point (0)
0x08, 0x04                  // N output pin connects to Exit point via custom route map #4
0x09, 0x7F                  // M output pin connects to Exit point via default route map (0x7F)

                            // ParmVal 2: Match Function Header Block
0x0A                        // ParmSize: (10)
0x04                        // ParmID: (4)
0x01, 0x00                  // flags: use 2x8 mode
0x02, 0x03                  // match event types: note on and note off
0x03, 0x02, 0x01, 0x05      // match channels: 1, 3, 8, 16

                            // ParmVal 3: Match Function 2x8 Block for byte 2
0x0C                        // ParmSize: (12)
0x05                        // ParmID: (5)
0x01, 0x02                  // flags: a lookup table is used
0x02, 0x08                  // lookup table ID (8)
0x03, 0x40                  // lookup table value used for matching (64)
0x04, 0x00                  // minimum value (not used)
0x05, 0x00                  // maximum value (not used)

                            // ParmVal 4: Match Function 2x8 Block for byte 3
0x0C                        // ParmSize: (12)
0x06                        // ParmID: (6)
0x01, 0x01                  // flags: logic is inverted
0x02, 0x00                  // lookup table ID (not used)
0x03, 0x00                  // lookup table value used for matching (not used)
0x04, 0x20                  // minimum value (32)
0x05, 0x80                  // maximum value (96)

                            // ParmVal 5: Modify Function Header Block
0x10                        // ParmSize: (16)
0x08                        // ParmID: (8)
0x01, 0x00                  // flags: use 2x8 mode
0x02, 0x08                  // modify in event types: controller
0x03, 0x03, 0x7F, 0x7F // modify in channels: all
0x04, 0x00                  // modify out event type: same as input (0)
0x05, 0x00, 0x00, 0x00      // modify out channel: same as input (0)

                            // ParmVal 6: Modify Function 2x8 Block for byte 2
0x18                        // ParmSize: (24)
0x09                        // ParmID: (9)
0x01, 0x01                  // flags: a lookup table is used
0x02, 0x07                  // lookup table ID (7)
0x03, 0x00                  // minimum in value (0)
0x04, 0x7F                  // maximum in value (127)
0x05, 0x00                  // minimum out value (0)
0x06, 0x7F                  // maximum out value (127)
0x07, 0x00                  // minimum default mode (0)
0x08, 0x00                  // minimum default value (not used)
0x09, 0x00                  // maximum default mode (0)
0x0A, 0x00                  // maximum default value (not used)
0x0B, 0x00                  // modify default value (0)

                            // ParmVal 7: Modify Function 2x8 Block for byte 3
0x18                        // ParmSize: (24)
0x0A                        // ParmID: (10)
```

```
0x01, 0x02                  // flags: use input byte 3
0x02, 0x00                  // lookup table ID (not used)
0x03, 0x0A                  // minimum in value (10)
0x04, 0x64                  // maximum in value (100)
0x05, 0x00                  // minimum out value (0)
0x06, 0x7F                  // maximum out value (127)
0x07, 0x03                  // minimum default mode (3)
0x08, 0x05                  // minimum default value (5)
0x09, 0x00                  // maximum default mode (2)
0x0A, 0x00                  // maximum default value (not used)
0x0B, 0x00                  // modify default value (0)

0xxx                        // checksum
0xF7                        // footer
```

Example: multi-operator algorithm, Match and Modify functions both use 1x16 mode

```
0xF0, 0x00, 0x01, 0x73, 0x7D  // header
0x00, 0x05                    // product ID
0x01, 0x02, 0x03, 0x04, 0x05  // serial number
0x00, 0x00, 0x00, 0x00        // session ID
0x00, 0x00, 0x00, 0x00        // transaction ID
0x00, 0x69                    // message length (105)
0x43, 0x07                    // message class, data class
0x02                          // NumDataBlock: (2)

0x05, 0x04                    // data block #1: size (5), type (ArgVal)
0x01                          // NumArgValBlock: (1)
0x07                          // ArgID: AMPID (7)
0x09                          // ArgVal: operator ID (9)

0x61, 0x03                    // data block #2: size (97), type (ParmVal)
0x05                          // NumParmValBlock: (5)

                              // ParmVal 1: Operator Connection Block
0x14                          // ParmSize: (20)
0x03                          // ParmID: (3)
0x01, 0x02                    // algorithm ID (2)
0x02, 0x02                    // flags: N output pin is connected to Modify input
0x03, 0x03                    // ID of operator that connects to this input pin of this operator (3)
0x04, 0x03                    // M output pin of operator #3 is connected to the input of this operator
0x05, 0x07                    // MIDI channel to use for MIDI event from operator 3 M output (7)
0x06, 0x05                    // output order is MNY
0x07, 0x00                    // Y output pin does not connect to Exit point (0)
0x08, 0x00                    // N output pin does not connect to Exit point (0)
0x09, 0x00                    // M output pin does not connect to Exit point (0)

                              // ParmVal 2: Match Function Header Block
0x0A                          // ParmSize: (10)
0x04                          // ParmID: (4)
0x01, 0x01                    // flags: use 1x16 mode
0x02, 0x40                    // match event types: pitch bend
0x03, 0x03, 0x7F, 0x7F // match channels: all

                              // ParmVal 3: Match Function 1x16 Block
0x0E                          // ParmSize: (14)
```

```
0x07                          // ParmID: (7)
0x01, 0x01                    // flags: logic is inverted
0x02, 0x00                    // lookup table ID (not used)
0x03, 0x00                    // lookup table value used for matching (not used)
0x04, 0x20, 0x00              // minimum value (0x1000)
0x05, 0x60, 0x00              // maximum value (0x3000)

                              // ParmVal 4: Modify Function Header Block
0x10                          // ParmSize: (16)
0x08                          // ParmID: (8)
0x01, 0x01                    // flags: use 1x16 mode
0x02, 0x40                    // modify in event types: pitch bend
0x03, 0x03, 0x7F, 0x7F // modify in channels: all
0x04, 0x00                    // modify out event type: same as input (0)
0x05, 0x00, 0x00, 0x20        // modify out channel: 6

                              // ParmVal 5: Modify Function 1x16 Block
0x22                          // ParmSize: (34)
0x0B                          // ParmID: (11)
0x01, 0x01                    // flags: a lookup table is used
0x02, 0x05                    // lookup table ID for LSB (5)
0x03, 0x06                    // lookup table ID for MSB (6)
0x04, 0x00                    // byte-2 fixed value (not used)
0x05, 0x00                    // byte-3 fixed value (not used)
0x06, 0x00, 0x00              // minimum in value (0x0000)
0x07, 0x7F, 0x7F              // maximum in value (0x3FFF)
0x08, 0x60, 0x00              // minimum out value (0x3000)
0x09, 0x20, 0x00              // maximum out value (0x1000)
0x0A, 0x02                    // minimum default mode (2)
0x0B, 0x00, 0x00              // minimum default value (not used)
0x0C, 0x02                    // maximum default mode (2)
0x0D, 0x00, 0x00              // maximum default value (not used)

0xxx                          // checksum
0xF7                          // footer
```

Parameters for AMP Custom Route Maps:

AMP custom route map parameters require an AMPID argument value block to specify the custom route map ID.

ID	Name	Description
0x0C	AMPCRMRoute	Bitmap indicating routing for MIDI events entering at the input for this port (BAx2 format). Bit 0 is routing from this port to port #1, bit 1 is routing from this port to port #2, etc. Least significant byte is first, most significant byte is last. The most significant 4 bits of each byte must be 0 which allows 4 ports to be specified per byte. Unused bits should be set to 0. The number of MIDI ports is given in the MIDIInfo:PortCount parameter. The number of bytes (in the bitmap) is an even value so that the bitmap (when unpacked) is a multiple of 8 bits: number of bytes (using INTEGER math) = (((number of MIDI ports - 1) / 8) + 1) x 2.

ID	Name	Description
		Example: device has 20 ports, route MIDI events coming into this port to ports 2, 3, 7, 11-14, 20: 0x06 // routing for ports 4 through 1 (ports 3 and 2 are selected) 0x04 // routing for ports 8 through 5 (port 7 is selected) 0x0C // routing for ports 12 through 9 (ports 12 and 11 are selected) 0x03 // routing for ports 16 through 13 (ports 14 and 13 are selected) 0x08 // routing for ports 20 through 17 (port 20 is selected) 0x00 // routing for ports 24 through 21 (padding)

Parameters for AMP Lookup Tables:

AMP lookup table parameters require an AMPID argument value block to specify the lookup table ID. Lookup tables are always 128 bytes in length which is too long to be sent in a single parameter. Thus, the first 64 bytes of the lookup table are accessed with parameter ID = 0x0D and the last 64 bytes of the lookup table are accessed with parameter ID = 0x0E.

ID	Name	Description
0x0D 0x0E	AMPLUTData1 AMPLUTData2	AMP lookup table data (all bytes must be in the range 0x00 – 0x7F). Bytes 1: index (i.e. starting address) (0 – 63). Bytes 2-N: lookup table data (0x00 – 0x7F). The first byte is an index into this portion of the lookup table (i.e. the starting address for this portion). The remaining bytes are the lookup table data. RetParmVal will always return all 64 bytes for this portion of the lookup table (index is always 0, lookup table data is always 64 bytes in length). SetParmVal can use a non-zero value for the offset byte and can write just a portion of the lookup table.

5.8 BulkData (Data Class = 0x70)

This data class is used to perform bulk data transfer operations (i.e. backup and restore) between a device and a host (or between two devices where one device acts as a host). See the Bulk Data Transfer section for more information regarding bulk data transfer operations.

Message Class Support:

Message Class	Notes
BulkTransfer	

6 Commands

The following command IDs are defined:

Command IDs:

ID	Name	Description
0x01	DeviceMode	Used to change a device's operating mode.
0x02	SaveLoad	Used to manage globals and presets between RAM and non-volatile storage.
0x03	SetGroup	Used to set a group of parameters to a predefined state.
0x04	BulkRequest	Used to start a bulk data transfer operation.
0x05	Notification	Used to register for automatic parameter updates.

6.1 DeviceMode (Command ID = 0x01)

DeviceMode commands are used to change a device's operating mode. Note that all mode changes will result in a loss of any global and preset parameters residing in volatile storage (RAM).

Val	Name	Description
0x01	RebootApp	Reboot device into application mode. No command arguments.
0x02	RebootBL	Reboot device into bootloader mode. No command arguments.
0x03	EraseRebootApp	Erase all user settings (set globals and presets to factory defaults) and reboot device into application mode. No command arguments.
0x04	EraseRebootBL	Erase all user settings (set globals and presets to factory defaults) and reboot device into bootloader mode. No command arguments.
0x05	Shutdown	Power down device. Device can only be restarted by toggling the power supply or by pressing the power button on the device. No command arguments.

Example: EraseRebootApp

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
0x00, 0x05                      // product ID
0x01, 0x02, 0x03, 0x04, 0x05    // serial number
0x00, 0x00, 0x00, 0x00          // session ID
0x00, 0x00, 0x00, 0x00          // transaction ID
0x00, 0x09                      // message length (9)
0x11, 0x00                      // message class, data class
```

```
0x01                    // NumDataBlock: (1)

0x06, 0x06              // data block #1: size (6), type (CmdVal)
0x01                    // NumCmdValBlock: (1)
0x03                    // block #1: CmdSize: (3)
0x01                    // block #1: CmdID: DeviceMode (1)
0x03                    // block #1: CmdVal: EraseRebootApp (3)

0xxx                    // checksum
0xF7                    // footer
```

6.2 SaveLoad (Command ID = 0x02)

SaveLoad commands are used to manage globals and presets between volatile (RAM) and non-volatile storage. For the area argument, 0 always refers to the work area and 1 – N refers to one of the shadow areas. The preset number is also a required argument whenever presets are saved or loaded.

Val	Name	Description
0x01	SaveGP	Save global and preset parameters (for a single preset) in RAM to non-volatile storage. The source area and destination preset are required arguments. CmdArg 1: source area in RAM (0 – N) (1 byte). CmdArg 2: destination preset number in non-volatile storage (1 – N) (1 byte).
0x02	SaveGlobal	Save global parameters in RAM to non-volatile storage. Preset parameters are not saved to non-volatile storage. The source area is a required argument. CmdArg 1: source area in RAM (0 – N) (1 byte).
0x03	SavePreset	Save preset parameters (for a single preset) in RAM to non-volatile storage. Global parameters are not saved to non-volatile storage. The source area and destination preset are required arguments. CmdArg 1: source area in RAM (0 – N) (1 byte). CmdArg 2: destination preset number in non-volatile storage (1 – N) (1 byte).
0x41	LoadGP	Load all global and preset parameters (for a single preset) from non-volatile storage into RAM. The destination area and source preset are required arguments. CmdArg 1: destination area in RAM (0 – N) (1 byte). CmdArg 2: source preset number in non-volatile storage (1 – N) (1 byte).
0x42	LoadGlobal	Load all globals from non-volatile storage into RAM. Presets are not loaded from non-volatile storage. The destination area is a required argument. CmdArg 1: destination area in RAM (0 – N) (1 byte).
0x43	LoadPreset	Load preset parameters (for a single preset) from non-volatile storage into RAM. Globals are not loaded from non-volatile storage. The destination area and source preset are required arguments. CmdArg 1: destination area in RAM (0 – N) (1 byte). CmdArg 2: source preset number in non-volatile storage (1 – N) (1 byte).

Example: SaveGP (save from work area to preset #8)

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x0B                     // message length (11)
0x11, 0x00                     // message class, data class
0x01                           // NumDataBlock: (1)

0x08, 0x06                     // data block #1: size (8), type (CmdVal)
0x01                           // NumCmdValBlock: (1)
0x05                           // block #1: CmdSize: (5)
0x02                           // block #1: CmdID: SaveLoad (2)
0x01                           // block #1: CmdVal: SaveAll (1)
0x00                           // block #1: CmdArg 1: source area (0)
0x08                           // block #1: CmdArg 2: destination preset (8)

0xxx                           // checksum
0xF7                           // footer
```

6.3 SetGroup (Command ID = 0x03)

SetGroup commands are used to set a group of parameters to a predefined state. For the area argument, 0 always refers to the work area and 1 – N refers to one of the shadow areas.

Val	Name	Description
0x01	Reset	Resets global and preset parameters (for a single preset) in RAM to a state suitable for creating a new preset. The destination area is a required argument. CmdArg 2 (also required) is one or more SetGroupReset parameter blocks, each block contains 2 bytes. CmdArg 1: destination area in RAM (0 – N) (1 byte). CmdArg 2: SetGroupReset Parameter Blocks (each block is 2 bytes).

SetGroupReset Parameter Block:

Each block contains one or more sub-ID & value pairs (always 2 bytes). Not all sub-IDs need to be included in the command. Any sub-ID not recognized is ignored (no error code returned). Any bits set to undefined values are also ignored (no error code returned).

Sub-ID	Description
0x01	MIDI port flags (1 byte), the following will be applied to all MIDI ports: bits 7 – 4: always 0. bit 3: if set, all output processing is disabled (all filters except Active Sense, remapping, AMP assignment). bit 2: if set, all input processing is disabled (all filters except Active Sense, remapping, AMP assignment). bit 1: if set, all routing is cleared.

Sub-ID	Description
	bit 0: if set, input and output ports are enabled and running status is disabled.
0x02	MIDI reservations and other parameters (1 byte): bits 7 – 3: always 0. bit 2: if set, all AMP algorithm and operator parameters are cleared. bit 1: if set, all MIDI Ethernet port information is cleared (all sessions set to responder, all reservations cleared). bit 0: if set, all MIDI USB Host Controller information is cleared (all reservations cleared).

Example: reset all parameters

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
0x00, 0x05                      // product ID
0x01, 0x02, 0x03, 0x04, 0x05    // serial number
0x00, 0x00, 0x00, 0x00          // session ID
0x00, 0x00, 0x00, 0x00          // transaction ID
0x00, 0x0E                      // message length (14)
0x11, 0x00                      // message class, data class
0x01                            // NumDataBlock: (1)

0x0B, 0x06                      // data block #1: size (11), type (CmdVal)
0x01                            // NumCmdValBlock: (1)
0x08                            // block #1: CmdSize: (8)
0x03                            // block #1: CmdID: SetGroup (3)
0x01                            // block #1: CmdVal: Reset (1)
0x00                            // block #1: CmdArg 1: Area (0)
0x01, 0x0F                      // block #1: CmdArg 2a: sub-ID (1), value (0x0F)
0x02, 0x07                      // block #1: CmdArg 2b: sub-ID (2), value (0x07)

0xxx                            // checksum
0xF7                            // footer
```

Example: reset all the USB host controller reservations

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
0x00, 0x05                      // product ID
0x01, 0x02, 0x03, 0x04, 0x05    // serial number
0x00, 0x00, 0x00, 0x00          // session ID
0x00, 0x00, 0x00, 0x00          // transaction ID
0x00, 0x0C                      // message length (12)
0x11, 0x00                      // message class, data class
0x01                            // NumDataBlock: (1)

0x09, 0x06                      // data block #1: size (9), type (CmdVal)
0x01                            // NumCmdValBlock: (1)
0x06                            // block #1: CmdSize: (6)
0x03                            // block #1: CmdID: SetGroup (3)
0x01                            // block #1: CmdVal: Reset (1)
0x00                            // block #1: CmdArg 1: Area (0)
0x02, 0x01                      // block #1: CmdArg 2: sub-ID (2), value (0x01)
```

```
0xxx                    // checksum
0xF7                    // footer
```

6.4 BulkRequest (Command ID = 0x04)

Bulk request commands are used to start a backup operation. A host sends a bulk request message to a device and the device returns an Ack message with an error code. If the Ack message returns with no error the device will begin the backup operation after a brief delay. The backup data is always read from non-volatile storage (not from RAM). See the Bulk Data Transfer section for more information regarding bulk data transfer operations.

Val	Name	Description
0x01	BackupAll	Backup globals and all presets from non-volatile storage. CmdArg 1: MIDI port to use for backup operation (0 – MIDIInfo:PortCount) (1 byte). Use 0 to use the same MIDI port that the BulkRequest message is sent on.
0x02	BackupPresetAll	Backup all presets from non-volatile storage. Globals are not backed up. CmdArg 1: MIDI port to use for backup operation (0 – MIDIInfo:PortCount) (1 byte). Use 0 to use the same MIDI port that the BulkRequest message is sent on.
0x03	BackupGlobal	Backup globals from non-volatile storage. Presets are not backed up. CmdArg 1: MIDI port to use for backup operation (0 – MIDIInfo:PortCount) (1 byte). Use 0 to use the same MIDI port that the BulkRequest message is sent on.
0x04	BackupPreset	Backup one preset from non-volatile storage. Globals are not backed up. CmdArg 1: MIDI port to use for backup operation (0 – MIDIInfo:PortCount) (1 byte). Use 0 to use the same MIDI port that the BulkRequest message is sent on. CmdArg 2: preset number in RAM (1 – N) (1 byte).
0x05	BackupGlobalPreset	Backup globals and one preset from non-volatile storage. CmdArg 1: MIDI port to use for backup operation (0 – MIDIInfo:PortCount) (1 byte). Use 0 to use the same MIDI port that the BulkRequest message is sent on. CmdArg 2: preset number in RAM (1 – N) (1 byte).

Example: Backup preset number 7 to MIDI port 3

```
0xF0, 0x00, 0x01, 0x73, 0x7D    // header
0x00, 0x05                      // product ID
0x01, 0x02, 0x03, 0x04, 0x05    // serial number
0x00, 0x00, 0x00, 0x00          // session ID
0x00, 0x00, 0x00, 0x00          // transaction ID
0x00, 0x0B                      // message length (11)
0x11, 0x00                      // message class, data class
0x01                            // NumDataBlock: (1)

0x08, 0x06                      // data block #1: size (8), type (CmdVal)
```

```
0x01                    // NumCmdValBlock: (1)
0x05                    // block #1: CmdSize: (5)
0x04                    // block #1: CmdID: BulkRequest (4)
0x04                    // block #1: CmdVal: BackupPreset (4)
0x03                    // block #1: CmdArg 1: MIDI port (3)
0x07                    // block #1: CmdArg 2: preset number (7)

0xxx                    // checksum
0xF7                    // footer
```

6.5 Notification (Command ID = 0x05)

Notification commands are used to register for automatic parameter updates. This allows a host to be informed about parameter changes without having to constantly poll a device. Once registered, a device will send NotParmVal messages to the host at regular intervals when any parameter is changed by another session or by some other control mechanism (e.g. front panel UI, MIDI control). A host should never respond to any NotParmVal messages. A device uses the session ID in the message header to track which host made parameter changes. The device always returns the transaction ID of the Register message in a NotParmVal message. The device requires that the host communicate with the device regularly (via sysex messages) otherwise the device will automatically cancel all notifications (even a ping message every few seconds will suffice). The timeout interval is given in DeviceInfo:NotificationTimeout.

Val	Name	Description
0x01	Register	Register one or more notification classes for automatic updates. CmdArg is a list of notification class IDs from the table below (1 byte each).
0x02	Unregister	Unregister one or more notification classes for automatic updates. CmdArg is a list of notification class IDs from the table below (1 byte each).

Notification Classes:

Arg	Name	Description
0x00	NotAll	All notification classes. Shortcut to register or unregister all notification classes for automatic updates.
0x02	NotDeviceInfo	DeviceInfo data class.
0x03	NotDeviceFeature	DeviceFeature data class.
0x04	NotHardwareInfo	HardwareInfo data class.
0x05	NotMIDIInfo	MIDIInfo data class. Use this to get automatic updates on MIDI activity.
0x06	NotMIDIPortInfo	MIDIPortInfo data class.
0x07	NotMIDIFeature	MIDIFeature data class.

Example: Register for notifications for all MIDI parameters

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x0C                     // message length (12)
0x11, 0x00                     // message class, data class
0x01                           // NumDataBlock: (1)

0x09, 0x06                     // data block #1: size (9), type (CmdVal)
0x01                           // NumCmdValBlock: (1)
0x06                           // block #1: CmdSize: (6)
0x05                           // block #1: CmdID: Notification (5)
0x01                           // block #1: CmdVal: Register (1)
0x05                           // block #1: CmdArg 1: NotMIDIInfo (5)
0x06                           // block #1: CmdArg 2: NotMIDIPortInfo  (6)
0x07                           // block #1: CmdArg 3: NotMIDIFeature  (7)

0xxx                           // checksum
0xF7                           // footer
```

Example: Unregister all notifications

```
0xF0, 0x00, 0x01, 0x73, 0x7D   // header
0x00, 0x05                     // product ID
0x01, 0x02, 0x03, 0x04, 0x05   // serial number
0x00, 0x00, 0x00, 0x00         // session ID
0x00, 0x00, 0x00, 0x00         // transaction ID
0x00, 0x0A                     // message length (10)
0x11, 0x00                     // message class, data class
0x01                           // NumDataBlock: (1)

0x07, 0x06                     // data block #1: size (7), type (CmdVal)
0x01                           // NumCmdValBlock: (1)
0x04                           // block #1: CmdSize: (4)
0x05                           // block #1: CmdID: Notification (5)
0x02                           // block #1: CmdVal: Unregister (2)
0x00                           // block #1: CmdArg 1: NotAll (0)

0xxx                           // checksum
0xF7                           // footer
```

7 Bulk Data Transfer

The BulkTransfer message and BulkData data class are used to perform bulk data transfer operations (i.e. backup and restore) between a device and a host (or between two devices where one device acts as a host). During a backup, the device being backed up sends BulkTransfer messages containing an image of the global or preset settings. During a restore, the device being restored receives BulkTransfer messages containing an image of the global or preset settings. Bulk data transfer operations support handshaking using the BulkAck parameter. Handshaking can speed up the data transfer but handshaking is not a requirement. Transaction ID (in the sysex header) is not incremented with each message; the entire bulk data transfer is considered to be a single transaction. The SequenceNumber parameter is used to ensure that all BulkTransfer messages are present in a bulk data transfer operation. The

SequenceNumber is 1 on the first message and increments by 1 on each subsequent message. In handshaking mode, the receiver sends a BulkTransfer message to the sender with a BulkAck parameter after each BulkTransfer message is received. If handshaking is used the receiver can request a resend of the last message by setting the BulkAck parameter value to Resend. The receiver can cancel the bulk data transfer operation by setting the BulkAck parameter value to Abort. Aborting works regardless of whether or not handshaking is used for the bulk data transfer operation.

7.1 BulkHdr (Data Block Type = 0x70)

This data block is used to define different packet types used in a bulk data transfer operation. This data block is always the first data block in a BulkTransfer message. There is only ever one BulkHdr data block.

Data Block Content:

Byte 3: Bulk packet type, see table below
Bytes 4-7: Sequence number (4-bytes, 28x4 format). Value is always 1 for the first message and increments by 1 on every subsequent message.
Bytes 8-N: Additional data, depends on bulk packet type, see table below.

Bulk Packet Types:

Type	Description
0x01	BulkStart: Always the first message in a bulk data transfer operation. Additional data: Bytes 8 – 9: device product ID (14x2 format) Bytes 10 – 14: device serial number (32x5 format) Bytes 15 – 18: device firmware version (28x4 format) Byte 19: number of chapters that are used in this bulk data transfer operation
0x02	BulkEnd: Always the last message in a bulk data transfer operation. No additional data bytes.
0x03	ChapterStart: Indicates the start of a new chapter. Additional data: Byte 8: chapter number (1 – N) Byte 9: preset number for this chapter (1 – N). Use 0 for globals.
0x04	ChapterEnd: Indicates the end of the current chapter. No additional data bytes.
0x05	PageData: Indicates that this message contains page data. No additional data bytes are included in this data block but additional data blocks are included in this message.
0x40	BulkAck: Used during handshaking mode or to cancel a bulk data transfer operation. Additional data: Byte 8: error code: 0 = OK: no error 1 = Resend: request resend of last BulkTransfer message 2 = Abort: abort bulk data transfer

Example: BulkAck data block (no error)

```
0x08                        // DataBlockSize: (8)
0x70                        // DataBlockType: (BulkHdr)
0x40                        // Bulk Packet Type: (BulkAck)
0x00, 0x00, 0x00, 0x02      // Sequence Number: (2)
0x00                        // Error code (no error)
```

7.2 Backup Operation

A backup operation begins when a device receives a BulkRequest command. The device will acknowledge the command by sending an Ack message as a response. The device will then pause for several seconds before sending BulkTransfer messages. The first message contains a BulkStart packet to delineate the start of the backup operation. This is then followed by additional BulkTransfer messages containing ChapterStart, ChapterEnd, and PageData packets. A backup operation concludes with a BulkTransfer message containing a BulkEnd packet. While the backup operation is commencing, the host can send BulkTransfer messages with a BulkAck packet to the device but this is not a requirement. After sending the first BulkTransfer message (the one containing the BulkStart packet), the device will wait a short time for a BulkTransfer message with a BulkAck packet set to OK (or Resend) to continue the backup operation in handshaking mode. If no handshake message is received the device will not use handshaking and will continue the backup operation at a leisurely pace. If handshaking is used, the host must send a BulkTransfer message with a BulkAck packet after every BulkTransfer message that it receives, failure to do so will result in the device aborting the backup operation. A backup operation can be cancelled by sending a BulkTransfer message to the device with a BulkAck packet set to Abort. Aborting works regardless of whether or not handshaking is used.

Even though there are multiple messages involved with a backup operation, the transaction ID in the sysex header does not increment; the device will use the transaction ID from the BulkRequest command in all BulkTransfer messages. If the host uses handshaking, the host should not increment the transaction ID in the handshaking messages. The host can return the SequenceNumber of the most recently received BulkTransfer message in the BulkAck packet but this is not required (any value can be used for SequenceNumber).

The host does not need to be concerned about parsing the content of the BulkTransfer messages; the host only needs to be able to save the entire stream of BulkTransfer messages to a file which can then be sent back to the device in a restore operation.

Example: Backup, no handshaking

step	Host	Device
1	Send BulkRequest: Trans ID = 37	
2		Device returns Ack message (no error). Device enters backup mode.
3		Send BulkStart: TransID = 37, SeqNum = 1
4		Wait a short time for BulkAck (none received). Handshaking is not used.
5		Send ChapterStart: TransID = 37, SeqNum = 2
6		Send PageData: TransID = 37, SeqNum = 3

step	Host	Device
7		Send ChapterEnd: TransID = 37, SeqNum = 4
8		Send BulkEnd: TransID = 37, SeqNum = 5 Device exits backup mode.

Example: Backup with handshaking, no errors

step	Host	Device
1	Send BulkRequest: Trans ID = 37	
2		Device returns Ack message (no error). Device enters backup mode.
3		Send BulkStart: TransID = 37, SeqNum = 1
4	Send BulkAck: TransID = 37, SeqNum = 1 ErrCode = OK	Wait a short time for BulkAck (OK received). Handshaking is used.
5		Send ChapterStart: TransID = 37, SeqNum = 2
6	Send BulkAck: TransID = 37, SeqNum = 2 ErrCode = OK	
7		Send PageData: TransID = 37, SeqNum = 3
8	Send BulkAck: TransID = 37, SeqNum = 3 ErrCode = OK	
9		Send ChapterEnd: TransID = 37, SeqNum = 4
10	Send BulkAck: TransID = 37, SeqNum = 4 ErrCode = OK	
11		Send BulkEnd: TransID = 37, SeqNum = 5
12	Send BulkAck: TransID = 37, SeqNum = 5 ErrCode = OK	
13		Device exits backup mode.

Example: Backup with handshaking, host requests a resend

step	Host	Device
1	Send BulkRequest: Trans ID = 37	
2		Device returns Ack message (no error). Device enters backup mode.
3		Send BulkStart: TransID = 37, SeqNum = 1
4	Send BulkAck: TransID = 37, SeqNum = 1 ErrCode = OK	Wait a short time for BulkAck (received). Handshaking is used.
5		Send ChapterStart: TransID = 37, SeqNum = 2
6	Send BulkAck: TransID = 37, SeqNum = 2 ErrCode = OK	
7		Send PageData: TransID = 37, SeqNum = 3
8	Send BulkAck: TransID = 37, SeqNum = 3 ErrCode = Resend	
9		Resend PageData: TransID = 37, SeqNum = 3
10	Send BulkAck: TransID = 37, SeqNum = 3 ErrCode = OK	
11		Send ChapterEnd: TransID = 37, SeqNum = 4
12	Send BulkAck: TransID = 37, SeqNum = 4 ErrCode = OK	
13		Send BulkEnd: TransID = 37, SeqNum = 5
14	Send BulkAck: TransID = 37, SeqNum = 5 ErrCode = OK	
15		Device exits backup mode.

Example: Backup with handshaking, host aborts backup

step	Host	Device
1	Send BulkRequest: Trans ID = 37	
2		Device returns Ack message (no error). Device enters backup mode.
3		Send BulkStart: TransID = 37, SeqNum = 1

step	Host	Device
4	Send BulkAck: TransID = 37, SeqNum = 1 ErrCode = OK	Wait a short time for BulkAck (received). Handshaking is used.
5		Send ChapterStart: TransID = 37, SeqNum = 2
6	Send BulkAck: TransID = 37, SeqNum = 2 ErrCode = OK	
7		Send PageData: TransID = 37, SeqNum = 3
8	Send BulkAck: TransID = 37, SeqNum = 3 ErrCode = Abort	
9		Device exits bulk backup mode.

7.3 Restore Operation

A restore operation begins when the device receives a BulkTransfer message with a BulkStart packet. The device will then send a BulkTransfer message to the host containing a BulkAck packet for every BulkTransfer message that it receives. If handshaking is used (by the host) the host should parse the received BulkAck packet and resend messages or abort as requested by the device. If handshaking is not used (by the host) the BulkTransfer messages should be sent at a leisurely rate so as to give the device enough time to deal with the incoming data. If the message stream is interrupted or if there is too long a delay between messages the device will terminate the restore operation. The restore operation can also be terminated by the host sending a BulkAck packet set to Abort, the device will respond with a BulkAck packet also set to Abort.

Even though there are multiple messages involved with a restore operation, the transaction ID in the sysex header does not increment; the host should use the same transaction ID in all BulkTransfer messages and use the SequenceNumber parameter to differentiate between the messages.

The host does not need to be concerned about parsing the content of the BulkTransfer messages that it sends to the device; the host only needs to be able to send each message to the device and respond to BulkAck errors reported from the device.

Example: Restore with handshaking, no errors

step	Host	Device
1	Send BulkStart: TransID = 37, SeqNum = 1	Device enters restore mode.
2		Send BulkAck: TransID = 37, SeqNum = 1 ErrCode = OK
3	Send ChapterStart: TransID = 37, SeqNum = 2	

step	Host	Device
4		Send BulkAck: TransID = 37, SeqNum = 2 ErrCode = OK
5	Send PageData: TransID = 37, SeqNum = 3	
6		Send BulkAck: TransID = 37, SeqNum = 3 ErrCode = OK
7	Send ChapterEnd: TransID = 37, SeqNum = 4	
8		Send BulkAck: TransID = 37, SeqNum = 4 ErrCode = OK
9	Send BulkEnd: TransID = 37, SeqNum = 5	
10		Send BulkAck: TransID = 37, SeqNum = 5 ErrCode = OK Device exits restore mode.

Example: Restore with handshaking, device requests a resend

step	Host	Device
1	Send BulkStart: TransID = 37, SeqNum = 1	Device enters restore mode.
2		Send BulkAck: TransID = 37, SeqNum = 1 ErrCode = OK
3	Send ChapterStart: TransID = 37, SeqNum = 2	
4		Send BulkAck: TransID = 37, SeqNum = 2 ErrCode = OK
5	Send PageData: TransID = 37, SeqNum = 3	
6		Send BulkAck: TransID = 37, SeqNum = 3 ErrCode = Resend
7	Resend PageData: TransID = 37, SeqNum = 3	
8		Send BulkAck: TransID = 37, SeqNum = 3 ErrCode = OK
9	Send ChapterEnd: TransID = 37, SeqNum = 4	
10		Send BulkAck: TransID = 37, SeqNum = 4 ErrCode = OK

step	Host	Device
11	Send BulkEnd: TransID = 37, SeqNum = 5	
12		Send BulkAck: TransID = 37, SeqNum = 5 ErrCode = OK Device exits restore mode.

Example: Restore with handshaking, device aborts transfer

step	Host	Device
1	Send BulkStart: TransID = 37, SeqNum = 1	Device enters restore mode.
2		Send BulkAck: TransID = 37, SeqNum = 1 ErrCode = OK
3	Send ChapterStart: TransID = 37, SeqNum = 2	
4		Send BulkAck: TransID = 37, SeqNum = 2 ErrCode = OK
5	Send PageData: TransID = 37, SeqNum = 3	
6		Send BulkAck: TransID = 37, SeqNum = 3 ErrCode = Abort Device exits restore mode.
7	Host aborts restore operation.	

Example: Restore with handshaking, device is not able to perform a restore operation at this time

step	Host	Device
1	Send BulkStart: TransID = 37, SeqNum = 1	Device is not able to perform a restore operation.
2		Send BulkAck: TransID = 37, SeqNum = 1 ErrCode = Abort
3	Host aborts restore operation.	

Example: Restore with handshaking, host aborts transfer

step	Host	Device
1	Send BulkStart: TransID = 37, SeqNum = 1	Device enters restore mode.
2		Send BulkAck: TransID = 37, SeqNum = 1 ErrCode = OK
3	Send ChapterStart: TransID = 37, SeqNum = 2	
4		Send BulkAck: TransID = 37, SeqNum = 2 ErrCode = OK
5	Send BulkAck: TransID = 37, SeqNum = 3 ErrCode = Abort	
6		Send BulkAck: TransID = 37, SeqNum = 3 ErrCode = Abort Device exits restore mode.
7	Host aborts restore operation.	

Example: Restore with handshaking, host sends wrong sequence number

step	Host	Device
1	Send BulkStart: TransID = 37, SeqNum = 1	Device enters restore mode.
2		Send BulkAck: TransID = 37, SeqNum = 1 ErrCode = OK
3	Send ChapterStart: TransID = 37, SeqNum = 2	
4		Send BulkAck: TransID = 37, SeqNum = 2 ErrCode = OK
5	Send ChapterEnd: TransID = 37, SeqNum = 4	
6		Send BulkAck: TransID = 37, SeqNum = 4 ErrCode = Abort Device exits restore mode.
7	Host aborts restore operation.	

8 Frequently Used Data Conversions

System exclusive messages require that the most significant bit of each byte be clear. Since many of the parameter values do not meet this requirement, some parameters need to be converted into a system exclusive friendly format. The following encodings are frequently used.

8.1 32x5: 32-bit value encoded into 5 bytes

Value as seen in memory (big endian, b31 is the most significant bit):

Byte #1: b31, b30, b29, b28, b27, b26, b25, b24
Byte #2: b23, b22, b21, b20, b19, b18, b17, b16
Byte #3: b15, b14, b13, b12, b11, b10, b09, b08
Byte #4: b07, b06, b05, b04, b03, b02, b01, b00

Value as seen in sysex message (the most significant four bits of byte #1 are clear, the most significant bit of bytes #2-5 are clear):

Byte #1: 000, 000, 000, 000, b31, b30, b29, b28
Byte #2: 000, b27, b26, b25, b24, b23, b22, b21
Byte #3: 000, b20, b19, b18, b17, b16, b15, b14
Byte #4: 000, b13, b12, b11, b10, b09, b08, b07
Byte #5: 000, b06, b05, b04, b03, b02, b01, b00

Example: IP address = 192.168.1.100

Value as seen in memory: 0xC0A80164
Value as seen in sysex message: 0x0C, 0x05, 0x20, 0x02, 0x64

8.2 28x4: 28-bit value encoded into 4 bytes

Value as seen in memory (big endian, value is stored as a 32-bit value with the most significant four bits clear, b27 is the most significant bit):

Byte #1: 000, 000, 000, 000, b27, b26, b25, b24
Byte #2: b23, b22, b21, b20, b19, b18, b17, b16
Byte #3: b15, b14, b13, b12, b11, b10, b09, b08
Byte #4: b07, b06, b05, b04, b03, b02, b01, b00

Value as seen in sysex message (the most significant bit of bytes #1-4 are clear):

Byte #1: 000, b27, b26, b25, b24, b23, b22, b21
Byte #2: 000, b20, b19, b18, b17, b16, b15, b14
Byte #3: 000, b13, b12, b11, b10, b09, b08, b07
Byte #4: 000, b06, b05, b04, b03, b02, b01, b00

Example: Session ID = 19,088,743 (decimal)

Value as seen in memory (big endian): 0x01234567
Value as seen in sysex message: 0x09, 0x0D, 0x0A, 0x67

8.3 16x3: 16-bit value encoded into 3 bytes

Value as seen in memory (big endian, b15 is the most significant bit):

Byte #1: b15, b14, b13, b12, b11, b10, b09, b08
Byte #2: b07, b06, b05, b04, b03, b02, b01, b00

Value as seen in sysex message (the most significant six bits of byte #1 are clear, the most significant bit of bytes #2-3 are clear):

Byte #1: 000, 000, 000, 000, 000, 000, b15, b14
Byte #2: 000, b13, b12, b11, b10, b09, b08, b07
Byte #3: 000, b06, b05, b04, b03, b02, b01, b00

Example: Headphone attenuation of -2.5 dB (in 8.8 format):

Value as seen in memory: 0xFD80
Value as seen in sysex message: 0x03, 0x7B, 0x00

8.4 14x2: 14-bit value encoded into 2 bytes

Value as seen in memory (big endian, value is stored as a 16-bit value with the most significant two bits clear, b13 is the most significant bit):

Byte #1: 000, 000, b13, b12, b11, b10, b09, b08
Byte #2: b07, b06, b05, b04, b03, b02, b01, b00

Value as seen in sysex message (the most significant bit of bytes #1-2 are clear):

Byte #1: 000, b13, b12, b11, b10, b09, b08, b07
Byte #2: 000, b06, b05, b04, b03, b02, b01, b00

Example: Meter value of +1.5 dB (9736 where 0 dB = 8192):

Value as seen in memory: 0x2608
Value as seen in sysex message: 0x4C, 0x08

8.5 BAx2: byte array of length N encoded into byte array of length 2N

Each byte of the input array is split into two bytes such that the most significant nibble of each byte is 0 and the least significant nibble contains either the least significant nibble of the input byte, or the most significant nibble of the input byte. The least significant nibble occurs in the first byte, the most significant nibble occurs in the second byte.

Value as seen in memory:

Byte #1: b15, b14, b13, b12, b11, b10, b09, b08
Byte #2: b07, b06, b05, b04, b03, b02, b01, b00

Value as seen in sysex message (the most significant four bits of bytes #1-4 are clear):

Byte #1: 000, 000, 000, 000, b03, b02, b01, b00
Byte #2: 000, 000, 000, 000, b07, b06, b05, b04
Byte #3: 000, 000, 000, 000, b11, b10, b09, b08
Byte #4: 000, 000, 000, 000, b15, b14, b13, b12

Example: single byte encoded into two bytes

Value as seen in memory: 0xDA
Value as seen in sysex message: 0x0A, 0x0D

Example: 6 bytes encoded into 12 bytes (MAC address = AC:7A:42:12:34:56)

Value as seen in memory: 0xAC, 0x7A, 0x42, 0x12, 0x34, 0x56
Value as seen in sysex message: 0x06, 0x05, 0x04, 0x03, 0x02, 0x01, 0x02, 0x04, 0x0A, 0x07, 0x0C, 0x0A

9 Product IDs

PID	Device
1 (0x0001)	iConnectMIDI (does not support the commands in this document)
2 (0x0002)	mio10 (does not support the commands in this document)
3 (0x0003)	mio (does not support the commands in this document)
4 (0x0004)	iConnectMIDI1 (does not support the commands in this document)
5 (0x0005)	iConnectMIDI2+ (does not support the commands in this document)
6 (0x0006)	iConnectMIDI4+ (does not support the commands in this document)
7 (0x0007)	iConnectAUDIO4+ (does not support the commands in this document)
8 (0x0008)	iConnectAUDIO2+ (does not support the commands in this document)
9 (0x0009)	mio2 (does not support the commands in this document)
10 (0x000A)	mio4 (does not support the commands in this document)
11 (0x000B)	PlayAUDIO12 (does not support the commands in this document)
12 (0x000C)	AUDIO4c (does not support the commands in this document)
13 (0x000D)	ConnectAUDIO2/4 (does not support the commands in this document)
14 (0x000E)	mioXL
15 (0x000F)	mioXM
16 (0x0010)	mioXC

10 History

#	Date/Author	Changes
1b15	2020-06-27 S. Juskiw	1. Changed SaveRestore commands to SaveLoad to avoid confusion with the Backup/Restore operations.
1b14	2020-04-23 S. Juskiw	1. Complete rewrite of everything related to Bulk Data Transfer.
1b13	2020-02-12 S. Juskiw	1. Added SetGroup command.
1b12	2019-09-26 S.Juskiw	1. Fixed errors in the bitmap for MIDI system message filter flags #1.
1b11	2019-09-12 S.Juskiw	1. Added ArgID:PresetID. Modified DeviceFeature:PresetName to allow using PresetID to get preset names from non-volatile storage without having to load each preset into the working area or shadow area.
1b10	2019-06-07 S.Juskiw	1. Added additional comments to several parameters for clarification. 2. Added USBHMIDIID argval for USB host controller MIDI ports (was previously using MIDIPortID which is a different argument with different values).
1b9	2019-05-06 S.Juskiw	1. Added USBH-MIDI parameters to HardwareInfo.
1b8	2019-05-03 S.Juskiw	1. Various changes to MIDIInfo Preset/Scene Selector parameters. 2. General cleanup of MIDIFeature:AMP parameters.
1b7	2019-04-22 S.Juskiw	1. Modifed the text descriptions for some of the MIDIPortInfo parameters.
1b6	2019-04-13 S.Juskiw	1. Various changes to MIDIPortInfo. 2. Added MIDIChannel ArgVal and renumbered the ArgVals. 3. Consolidated AMP ArgVals and changed structures to use sub-IDs. 4. Added additional Ack message error codes.
1b5	2019-03-29 S.Juskiw	1. Moved several of the MIDIInfo parameters to HardwareInfo. 2. Changed the parameter IDs for HardwareInfo. 3. Added several parameters to MIDIInfo.
1b4	2019-02-19 S. Juskiw	1. Many changes to presets, scenes, areas and how these are managed. 2. Removed segment designator data block, merged into argument list. 3. Removed iOS related flags and parameters. 4. Added flags for USB device port speed and parameter for USB host jacks. 5. Changes to BAx2 data format.
1b3	2018-10-18 S. Juskiw	1. Removed CC filter and CC remap parameters (use AMP instead). 2. Changed system filters, channel filters, channel remaps. 3. Added USBDPortNameMax and EthSesnNameMax parameters. 4. Created PortConnectFlags and PortActiveFlags parameters and eliminated USBHConnect, EthConnect parameters.

#	Date/Author	Changes
		5. Changes to PortSupportFlags and PortEnableFlags. 6. Changes to segment designator data block. 7. Removed version bytes from everywhere. 8. Complete rewrite of the memory model. 9. Changes to SaveRestore command.
1b2	2018-04-06 S. Juskiw	1. Added support for parameter sub-IDs (break up some of the fixed structures).
1b1	2018-03-25 S. Juskiw	1. First version (RFC). No support for audio commands.