

## Overview

### Header:

0xF0	- start of system exclusive
0x00, 0x01, 0x73	- iConnectivity's manufacturer ID code
0x7F	- device ID (all devices)

### Body:

Product ID	- one byte
Command ID	- one byte
Data Length	- one byte, number of bytes that follow in Command Data (<128)
Command Data	- length varies depending on command
Checksum	- one byte, 2s complement of sum of all bytes in body, excluding checksum byte - (in other words, summing all the bytes in the body should result in 0x00)

### Footer:

0xF7	- end of system exclusive
------	---------------------------

Note that all bytes in the body have the most significant bit clear (as required by MIDI).

Product ID is always 0x01 for the first version of iConnectMIDI. Future versions of iConnectMIDI may have different configurations of USB and DIN ports and will require different commands to configure them (and thus a unique Product ID).

## Commands

### 1. Filter Configuration (Command ID = 0x7B)

This command indicates which MIDI messages should be filtered (i.e. muted) for each port. Each port has both an incoming and outgoing filter. Input filtering is applied when a MIDI message is received by a port. Output filtering is applied to each MIDI message sent to a port. This command is sent to iCM to set the current filter configuration. iCM will respond with an Ack command (0x71).

The current filter configuration can be retrieved from iCM by sending an "Info" command to iCM with the "Get Filter Configuration" code. iCM will return the entire filter configuration in one sysex message.

#### Command Data

Command data is specified in 3 byte blocks: 1 block = input or output filter for one port. Multiple blocks are allowed in a single sysex message. The minimum number of blocks is 1, the maximum number is 24 (i.e. one input filter and one output filter for all 12 ports). Data length must be a multiple of 3.

#### Block Format

Byte #1:

Lower nibble indicates source port as follows:

DIN 1 = 0x0  
 DIN 2 = 0x1  
 USB D1 = 0x2  
 USB D2 = 0x3  
 USB H1 = 0x4  
 USB H2 = 0x5  
 USB H3 = 0x6  
 USB H4 = 0x7  
 USB H5 = 0x8  
 USB H6 = 0x9  
 USB H7 = 0xA  
 USB H8 = 0xB

Upper nibble indicates "input filter" or "output filter" as follows:

Output = 0x0  
 Input = 0x1

Byte #2: bitmap indicating filter status for the following MIDI events:

Bit 0: set if port should filter MIDI note events (0x8n, 0x9n)  
 Bit 1: set if port should filter MIDI poly key pressure events [poly aftertouch] (0xA<sub>n</sub>)  
 Bit 2: set if port should filter MIDI control change events (0xB<sub>n</sub>)  
 Bit 3: set if port should filter MIDI program change events (0xC<sub>n</sub>)  
 Bit 4: set if port should filter MIDI channel pressure events [mono aftertouch] (0xD<sub>n</sub>)  
 Bit 5: set if port should filter MIDI pitch bend events (0xE<sub>n</sub>)  
 Bit 6: set if port should filter MIDI reset events (0xFF)  
 Bit 7: always 0

Byte #3: bitmap indicating filter status for the following MIDI events:

Bit 0: set if port should filter MIDI system exclusive events (0xF0)  
 Bit 1: set if port should filter MIDI time code (MTC) events (0xF1)  
 Bit 2: set if port should filter MIDI song position pointer events (0xF2)  
 Bit 3: set if port should filter MIDI song select events (0xF3)  
 Bit 4: set if port should filter MIDI tune request events (0xF6)  
 Bit 5: set if port should filter MIDI realtime events (not including Active Sensing and Reset) (0xF8-0xFD)  
 Bit 6: set if port should filter MIDI active sensing events (0xFE)  
 Bit 7: always 0

*Example #1: filter active sensing on DIN 1 input*

```

0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01 // product ID
0x7B // command ID
0x03 // data length
0x10, 0x00, 0x40 // command data block #1 (active sensing on DIN 1 input)
0x31 // checksum
0xF7 // footer
  
```

*Example #2: filter realtime on USB H5 output and mono aftertouch on DIN 2 input*

```

0xF0, 0x00, 0x01, 0x73, 0x7F // header
  
```

```

0x01          // product ID
0x7B          // command ID
0x06          // data length
0x08, 0x20, 0x00 // command data block #1 (realtime on USB H5 output)
0x11, 0x10, 0x00 // command data block #2 (mono aftertouch on DIN 2 input)
0x35          // checksum
0xF7          // footer

```

## 2. Route Configuration (Command ID = 0x79)

This command indicates how incoming MIDI messages should be routed. The default factory configuration routes incoming data on DIN ports to all ports including itself, and from USB ports to all other ports except itself. This command is sent to iCM to set the current route configuration. iCM will respond with an Ack command (0x71).

The current route configuration can be retrieved from iCM by sending an “Info” command to iCM with the “Get Route Configuration” code. iCM will return the entire route configuration in one sysex message.

### Command Data

Command data is specified in 4 byte blocks: 1 block = 1 port. Multiple blocks are allowed in a single sysex message. The minimum number of blocks is 1, the maximum number is 12 (i.e. one route for each port). Data length must be a multiple of 4.

### Block Format

Byte #1: Indicates source port

```

DIN 1      = 0x00
DIN 2      = 0x01
USB D1     = 0x02
USB D2     = 0x03
USB H1     = 0x04
USB H2     = 0x05
USB H3     = 0x06
USB H4     = 0x07
USB H5     = 0x08
USB H6     = 0x09
USB H7     = 0x0A
USB H8     = 0x0B

```

Byte #2: bitmap indicating routing of source port to DIN and USB device ports

```

Bit 0: set if source port should send data to DIN 1
Bit 1: set if source port should send data to DIN 2
Bit 2: set if source port should send data to USB D1
Bit 3: set if source port should send data to USB D2
Bits 4-7: always 0

```

Byte #3: bitmap indicating routing of source port to USB H1 through USB H4 ports

```

Bit 0: set if source port should send data to USB H1

```

Bit 1: set if source port should send data to USB H2  
 Bit 2: set if source port should send data to USB H3  
 Bit 3: set if source port should send data to USB H4  
 Bits 4-7: always 0

Byte #4: bitmap indicating routing of source port to USB H5 through USB H8 ports

Bit 0: set if source port should send data to USB H5  
 Bit 1: set if source port should send data to USB H6  
 Bit 2: set if source port should send data to USB H7  
 Bit 3: set if source port should send data to USB H8  
 Bits 4-7: always 0

*Example #1: route DIN 1 to DIN 2 and USB D1*

```
0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01                          // product ID
0x79                          // command ID
0x04                          // data length
0x00, 0x06, 0x00, 0x00      // command data block #1 (DIN 1 to DIN 2 and USB D1)
0x7C                          // checksum
0xF7                          // footer
```

*Example #2: route DIN 2 to USB D2 and USB H1, route USB H4 to DIN 1, DIN 2 and USB H5-H8 ports*

```
0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01                          // product ID
0x79                          // command ID
0x08                          // data length
0x01, 0x08, 0x01, 0x00      // command data block #1 (DIN 2 to USB D2 and USB H1)
0x07, 0x03, 0x00, 0x0F      // command data block #2 (USB H4 to DIN 1, DIN 2 and USB H5-H8)
0x5B                          // checksum
0xF7                          // footer
```

### 3. Port Configuration (Command ID = 0x78)

This command indicates what is plugged into each of the USB ports. This command is never sent to iCM; it is sent from iCM to a Mac/PC, iOS device, or MIDI device. The port configuration is retrieved by sending an “Info” command (0x70) to iCM with the “Get Port Configuration” command type (0x01). iCM will return the entire route configuration in one sysex message.

#### Command Data

Data length is always 6. Byte #1 is the port number for the device that made the request (this byte answers the question: “Where am I on the iCM network?”). Byte #2 indicates USB device port connectivity. Bytes #3-6 indicate USB host port connectivity.

The following codes are used for the device type fields in bytes 2 through 6:

```
0x0 - nothing connected to port
0x1 - Mac/PC connected to port
0x2 - USB-MIDI device connected to port
```

- 0x3 - iOS device connected to port
- 0x4 - iConnectDMX device connected to port

Byte #1: Indicates port number of device that made the request

- DIN 1 = 0x00
- DIN 2 = 0x01
- USB D1 = 0x02
- USB D2 = 0x03
- USB H1 = 0x04
- USB H2 = 0x05
- USB H3 = 0x06
- USB H4 = 0x07
- USB H5 = 0x08
- USB H6 = 0x09
- USB H7 = 0x0A
- USB H8 = 0x0B

Byte #2: bitmap indicating connections to USB device ports

- Bits 0-2: indicate device type connected to USB D1
- Bit 3: always 0
- Bits 4-6: indicate device type connected to USB D2
- Bit 7: always 0

Byte #3: bitmap indicating connections to USB host ports

- Bits 0-2: indicate device type connected to USB H1
- Bit 3: always 0
- Bits 4-6: indicate device type connected to USB H2
- Bit 7: always 0

Byte #4: bitmap indicating connections to USB host ports

- Bits 0-2: indicate device type connected to USB H3
- Bit 3: always 0
- Bits 4-6: indicate device type connected to USB H4
- Bit 7: always 0

Byte #5: bitmap indicating connections to USB host ports

- Bits 0-2: indicate device type connected to USB H5
- Bit 3: always 0
- Bits 4-6: indicate device type connected to USB H6
- Bit 7: always 0

Byte #6: bitmap indicating connections to USB host ports

- Bits 0-2: indicate device type connected to USB H7
- Bit 3: always 0
- Bits 4-6: indicate device type connected to USB H8
- Bit 7: always 0

*Example #1: Mac connected to USB D1, iOS device connected to USB D2, USB-MIDI devices connected to USB H1 and USB H2, Mac makes the port configuration request (the following sysex message is sent to Mac)*

```
0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01 // product ID
0x78 // command ID
0x06 // data length
0x02 // Mac port number (Mac made the request)
0x31 // Mac on USB D1, iOS device on USB D2
0x22 // USB-MIDI devices on USB H1 and USB H2
0x00 // nothing on USB H3 and USB H4
0x00 // nothing on USB H5 and USB H6
0x00 // nothing on USB H7 and USB H8
0x0A // checksum
0xF7 // footer
```

*Example #2: iOS device connected to USB D2, USB-MIDI devices connected to USB H2 and USB H7, iOS device makes the port configuration request (the following sysex message is sent to iOS device)*

```
0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01 // product ID
0x78 // command ID
0x06 // data length
0x03 // iOS device port number (iOS device made the request)
0x30 // nothing on USB D1, iOS device on USB D2
0x20 // nothing on USB H1, USB-MIDI device on USB H2
0x00 // nothing on USB H3 and USB H4
0x00 // nothing on USB H5 and USB H6
0x02 // USB-MIDI device on USB H7, nothing on USB H8
0x2A // checksum
0xF7 // footer
```

## 4. Ack (Command ID = 0x71)

This command is sent by iCM in response to various other commands (e.g. Filter Configuration, Route Configuration, Info:Save/Restore Configuration).

### Command Data

Data length is always 1.

Byte #1: response to most recent command

```
0x00 - no error
0x01 - unknown command
0x02 - malformed message
0x03 - command failed
```

*Example:*

```
0xF0, 0x00, 0x01, 0x73, 0x7F // header
```

```

0x01          // product ID
0x71          // command ID
0x01          // data length
0x00          // response (no error)
0x0D          // checksum
0xF7          // footer

```

## 5. Info (Command ID = 0x70)

This command is used to retrieve version info about iCM as well as the current filter, route, and port configuration. It is also used to save/restore configuration parameters to/from FLASH and to restore the iCM configuration to the factory default.

### Command Data

Data length is always 2. Byte #1 is always the command type. Byte #2 varies depending on the command.

### ***Info Command Types (Byte #1):***

#### 5.1. Get Version Info (Type = 0x00)

This command instructs iCM to return version information in a Version Info (0x72) sysex message. Byte #2 is the parameter ID (see Version Info command for a list of parameter IDs). iCM will return an Ack command (0x71) with an error code if an invalid parameter ID is sent.

*Example: get manufacturer name*

```

0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01          // product ID
0x70          // command ID
0x02          // data length
0x00          // type (get version info)
0x01          // parameter id (get manufacturer name)
0x0C          // checksum
0xF7          // footer

```

#### 5.2. Get Port Configuration (Type = 0x01)

This command instructs iCM to return the current port configuration in a Port Configuration (0x78) sysex message.

*Example:*

```

0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01          // product ID
0x70          // command ID
0x02          // data length
0x01          // type (get port configuration)
0x00          // subtype (none)
0x0C          // checksum
0xF7          // footer

```

### 5.3. Get Route Configuration (Type = 0x02)

This command instructs iCM to return the current route configuration in a Route Configuration (0x79) sysex message.

*Example:*

```
0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01 // product ID
0x70 // command ID
0x02 // data length
0x02 // type (get route configuration)
0x00 // subtype (none)
0x0B // checksum
0xF7 // footer
```

### 5.4. Get Filter Configuration (Type = 0x03)

This command instructs iCM to return the current filter configuration in a Filter Configuration (0x7B) sysex message.

*Example:*

```
0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01 // product ID
0x70 // command ID
0x02 // data length
0x03 // type (get filter configuration)
0x00 // subtype (none)
0x0A // checksum
0xF7 // footer
```

### 5.5. Save/Restore Configuration (Type = 0x11)

These commands are used for saving/restoring configuration parameters to/from FLASH. iCM will respond with an Ack command (0x71). Byte #1 is always 0x11. Byte #2 is the command subtype.

#### ***Save/Restore Configuration Command Subtypes (Byte #2):***

#### **5.5.1. Save Current Configuration to FLASH (Subtype = 0x01)**

This command instructs iCM to save the current configuration to FLASH. iCM will use the configuration stored in FLASH at power up. Use the Route Configuration command (0x79) and Filter Configuration command (0x7B) to set the configuration in RAM first, then send this command to save the RAM configuration to FLASH.

*Example:*

```
0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01 // product ID
0x70 // command ID
0x02 // data length
```

```

0x11 // type (save/restore configuration)
0x01 // subtype (save current configuration to FLASH)
0x7B // checksum
0xF7 // footer

```

### 5.5.2. Restore Current Configuration from FLASH (Subtype = 0x41)

This command instructs iCM to retrieve the configuration from FLASH and store it in RAM as the current configuration. This is analogous to resetting iCM to its power up state.

*Example:*

```

0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01 // product ID
0x70 // command ID
0x02 // data length
0x11 // type (save/restore configuration)
0x41 // subtype (restore current configuration from FLASH)
0x3B // checksum
0xF7 // footer

```

### 5.5.3. Restore Current Configuration to Default Values (Subtype = 0x42)

This command instructs iCM to set the RAM configuration to the default values that iCM uses when it is shipped from the factory. Note that in order to make these settings the default on power up, it is necessary to first issue this command, then issue a Save Current Configuration to FLASH command.

*Example:*

```

0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01 // product ID
0x70 // command ID
0x02 // data length
0x11 // type (save/restore configuration)
0x42 // subtype (restore current configuration to default values)
0x3A // checksum
0xF7 // footer

```

## 6. Version Info (Command ID = 0x72)

This command indicates firmware, hardware, model number, manufacturer name and other parameters related to the particular build of iCM. This command is never sent to iCM; it is sent from iCM to a Mac/PC, iOS device, or MIDI device. Version info is retrieved by sending an “Info” command (0x70) to iCM with the “Get Version Info” command type (0x00) for byte #1 and a parameter ID for byte #2 (see below for a list of valid parameter IDs). iCM will return the requested info as an ASCII (7-bit) string in one sysex message. iCM will return an Ack command (0x71) with an error code if an invalid parameter ID is sent.

### Command Data

Data length varies depending on the length of the returned string. The returned strings are not NULL terminated. Use the data length fields to determine the string length.

Byte #1: parameter ID

0x00 - accessory name  
 0x01 - manufacturer name  
 0x02 - model number  
 0x03 - serial number  
 0x04 - firmware version  
 0x05 - hardware version  
 0x06 - unit number

Bytes #2 - #N: Parameter value, 7-bit ASCII string (length varies)

*Example:*

```
0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01                          // product ID
0x72                          // command ID
0x06                          // data length
0x04                          // parameter ID (firmware version)
0x31, 0x2E, 0x30, 0x2E, 0x37 // parameter value (the ascii string "1.0.7")
0x0F                          // checksum
0xF7                          // footer
```

## 7. Reset (Command ID = 0x7F)

This command instructs iCM to reset itself. iCM will respond with an Ack command (0x71) and then reset itself after a short delay (about 1 second).

### Command Data

Data length is always 1.

Byte #1: reset type

0x00 - hard reset

*Example:*

```
0xF0, 0x00, 0x01, 0x73, 0x7F // header
0x01                          // product ID
0x7F                          // command ID
0x01                          // data length
0x00                          // reset type (hard reset)
0x7F                          // checksum
0xF7                          // footer
```