



The plugin that connects FileMaker to MIDI.

[www.fm2midi.com](http://www.fm2midi.com)

Version 2.2

FM2MIDI Copyright © 2015 FileRocket Consulting, LLC. All Rights Reserved. 24u Plug-In Template is Copyright © 2002 - 2012 24U s.r.o. All rights reserved. The FileMaker FMPluginSDK is Copyright © 1998 - 2012 FileMaker, Inc. All rights reserved. RtMidi: RtMidi: realtime MIDI i/o C++ classes Copyright (c) 2003-2017 Gary P. Scavone

Updated: January 26, 2018

## **What is FM2MIDI?**

FM2MIDI is a FileMaker Pro plugin for Windows and Macintosh that allows you to communicate with external MIDI devices. The plugin is intended for storing and manipulating the “sysex” or “System Exclusive” data of synthesizers, drum machines, effect processors, etc.

The functions only offer real-time communications. There is no support for playing or recording MIDI files. The plugin will not turn FileMaker into a MIDI sequencer. However, the power of FileMaker Pro is in data storage and manipulation, this makes FileMaker an excellent tool for building editor-librarians. It may even be the ultimate universal editor-librarian.

## **Requirements**

The plugin works with FileMaker Pro 12 - 16 and FileMaker Pro 12 - 16 Advanced on Mac OS X and Windows. The plugin is intended only for client applications and will not function as a server plugin. However, it will work fine with files hosted with FileMaker Server and FileMaker Server Advanced.

## **FM2MIDI FUNCTIONS**

FM2MIDI\_PortCount( PortType )  
FM2MIDI\_PortName( PortType ; PortNumber )  
FM2MIDI\_MIDISend( PortNumber ; Message )  
FM2MIDI\_MIDIReceive( InPort ; { OutPort ; Request } )  
FM2MIDI\_NumToBin( NumToConvert )  
FM2MIDI\_BinToNum( BinToConvert )  
FM2MIDI\_NumToHex( NumToConvert )  
FM2MIDI\_HexToNum( HexToConvert )  
FM2MIDI\_ValueToBytes( ValueToConvert ; NumOfBytes ; LSBFirst )  
FM2MIDI\_BytesToValue( BytesToConvert ; LSBFirst )  
FM2MIDI\_Checksum( NumbersToSum )  
FM2MIDI\_SysexRead( PathToFile )  
FM2MIDI\_SysexWrite( DataToSave ; PathToFile )  
FM2MIDI\_Version( { Format } )

## PLUGIN INSTALLATION

FM2MIDI requires you copy the plugin file to FileMaker's Extension folder.

### **Macintosh:**

In the Mac OS plugin is both 32-bit and 64-bit. Copy it to the FileMaker Pro Extension folder here:

Applications/FileMaker Pro XX/Extensions/

### **Windows:**

#### **32-bit for FileMaker 12, 13, or 14**

If you are running FileMaker Pro a Windows 32-bit environment copy the 32-bit FM2MIDI into the FileMaker Pro Extension folder here:

C:/Program Files/FileMaker/FileMaker Pro XX/Extensions/

#### **64-bit for FileMaker 14, 15, or 16**

If you are running FileMaker Pro a Windows 64-bit environment copy the 64-bit FM2MIDI plugin into the FileMaker Pro Extension folder here:

C:/Program Files/FileMaker/FileMaker Pro XX/Extensions/

### Working with a Comma-Delimited List of Numbers

The plugin uses a comma-delimited list of values to represent the various bytes of MIDI data. Usually each byte of MIDI data is represented in hexadecimal (00,FF) format. This is quite common on many MIDI implementation charts. However, most digital synthesizer parameters, such as the patch volume, are usually represented by a number, from 0 to 100 for example. Therefore, I'm converting these bytes to numbers (integers) to make them easier to work with.

To get a specific value from the list of values you can use the Substitute function to convert the list to a return-delimited value list, then use the GetValue function to return the value at a given position.

```
GetValue ( Substitute ( ListOfNumbers ; "," ; "¶" ) ; ValueNumber )
```

This turns: 32,46,64,21,18,36

Into:

32 ¶

46 ¶

64 ¶ <- If ValueNumber = 3 then the result would be 64.

21 ¶

18 ¶

36

Use the conversion functions to further convert your data. You can for example use the new NumToBin function to convert a number to its binary sequence. Flip a bit, convert it back to a number and send it back to the instrument.

**FM2MIDI\_PortCount**

**Syntax:** FM2MIDI\_PortCount( PortType )

**Description:** Gets the number of MIDI ports based on port type.

Use this function to give you a limit for a looping script allowing you to get the port names. This will be a count of the ports beginning with 1. Keep in mind that the port numbers begin with 0. The last port number will be one fewer than the actual port count. If no ports are found this function returns a 0.

**Parameters:**

PortType - "in" for MIDI in ports and "out" for MIDI out ports.

**Result:** The number of ports currently available.

**Example:**

```
// Gets the number of available MIDI in ports  
FM2MIDI_PortCount( "in" )
```

```
// Gets the number of available MIDI out ports  
FM2MIDI_PortCount( "out" )
```

**FM2MIDI\_PortName**

**Syntax:** FM2MIDI\_PortName( PortType ; PortNumber )

**Description:** Gets the name of a MIDI port by port number.

Use this function to get the name of a MIDI port. The MIDI port numbers always begin with 0 as the first port. Use the FM2MIDI\_PortCount function to get the number of available ports and a looping script to get the port names.

**Parameters:**

PortType	- "in" for MIDI in ports and "out" for MIDI out ports.
PortNumber	- A whole number representing the port to query.

**Result:** The name of the MIDI port.

**Example:**

```
// Gets the name of the MIDI in port at port number 0
FM2MIDI_PortName( "in" ; 0 )
```

**FM2MIDI\_MIDISend**

**Syntax:** FM2MIDI\_MIDISend( PortNumber ; Message )

**Description:** Opens a MIDI port and sends a MIDI message.

Opens an output port and transmits the MIDI message. You can transmit any MIDI message with this command including patch changes, control changes, note on/off, and Sysex.

**Parameters:**

PortNumber	- A whole number representing the port to open.
MIDIMessage	- A MIDI message to send out the MIDI port.

**Result:** 0

**Example:**

```
// Send patch change on Port 0, channel 1, and patch 11.  
FM2MIDI_MIDISend( 0 ; "192,11" )
```

**FM2MIDI\_MIDIReceive**

**Syntax:** FM2MIDI\_MIDIReceive( InPort ; { OutPort ; MIDIRequest } )

**Description:** Gets MIDI input from the input buffer.

Opens an input port and waits for a MIDI message to fill the buffer. The function will timeout if no message is received. You can use the optional MIDI Request parameters to send a Data Dump Request to the MIDI instrument.

**Parameters:**

InPort	- A whole number representing the MIDI in port to open.
OutPort	- A whole number representing the MIDI out port to open.
MIDIRequest	- A MIDI message to send via the MIDI out port.

**Result:** The MIDI message received from the MIDI in port.

**Examples:**

```
// Sends the request on MIDI Port 1 and waits for a response on MIDI port 0.  
FM2MIDI_MIDIReceive( 0 ; 1 ; "240,64,0,0,0,3,0,0,240" )
```

```
// Open port 2 and wait for a response.  
FM2MIDI_MIDIReceive( 2 )
```



## **FM2MIDI\_NumToBin**

**Syntax:** FM2MIDI\_NumToBin( NumToConvert )

**Description:** Converts a number to the binary equivalent.

Takes a list of numbers and converts them to a list of binary values. For example the numbers "64,128" would be converted to "01000000,10000000".

**Parameters:**

NumToConvert - A coma delimited list of number values.

**Result:** A comma delimited list of binary values.

**Examples:**

```
// Convert the number list to the binary equivalent.  
FM2MIDI_NumToBin( "1,2,4,8,16,32,64,128" )
```

## **FM2MIDI\_BinToNum**

**Syntax:** FM2MIDI\_BinToNum( BinToConvert )

**Description:** Converts a list of binary values to their number equivalent.

Takes a list of 8-bit values and converts them to a list of numbers. For example the binary values "01000000,10000000" would be converted to "64,128".

**Parameters:**

BinToConvert        - A coma delimited list of binary values.

**Result:**            A comma delimited list of numbers.

**Examples:**

```
// Convert the binary value list to numbers.  
FM2MIDI_BinToNum( "00000001,00000010,00000100,00001000" )
```

## **FM2MIDI\_NumToHex**

**Syntax:** FM2MIDI\_NumToHex( NumToConvert )

**Description:** Converts a list of numbers to the hexadecimal equivalent.

Takes a list of numbers and converts them to a list of hexadecimal values. For example the numbers "77,127" would be converted to "4D,7F".

**Parameters:**

NumToConvert - A comma delimited list of number values.

**Result:** A comma delimited list of hexadecimal values.

**Examples:**

```
// Convert the number list to the hexadecimal equivalent.  
FM2MIDI_NumToHex( "1,2,4,8,16,32,64,128" )
```

## FM2MIDI\_HexToNum

**Syntax:** FM2MIDI\_HexToNum( HexToConvert )

**Description:** Converts a list of hexadecimal values to their number equivalent.

Takes a list of hexadecimal values and converts them to a list of numbers. For example the hexadecimal value "4D,7F" would be converted to "77,127".

**Parameters:**

HexToConvert - A coma delimited list of hexadecimal values.

**Result:** A comma delimited list of numbers.

**Examples:**

```
// Convert the list of hexadecimal values to numbers.  
FM2MIDI_HexToNum( "01,0F,10,2B,4D,7F,FF" )
```

**FM2MIDI\_BytesToValue**

**Syntax:** FM2MIDI\_BytesToValue( BytesToConvert ; LSBFirst )

**Description:** Converts a list of numbers that represent 7-bit bytes (0-127) into a single numeric value. Used to calculate multibyte values for SysEx or NRPNs

Takes a list of up-to-four 7-bit bytes (0-127) as numbers and converts them to a combined value. The order of the bytes is important. If for example the least significant byte was first, then the numbers "127,1" would equal 255. If the least significant byte was last the value would be 16257.

**Parameters:**

BytesToConvert	- A coma delimited list of numbers.
LSBFirst	- Calculate with the LSB first ("true" or "false")

**Result:** A single numeric value.

**Examples:**

```
// Convert the list of numbers into a value.  
FM2MIDI_BytesToValue( "127,1" ; "true" )
```

**FM2MIDI\_ValueToBytes**

**Syntax:** FM2MIDI\_ValueToBytes( NumToConvert ; NumOfBytes ;  
LSBFirst )

**Description:** Breaks a single number into bytes. Supports a maximum of up to 4 bytes.

Takes a number and converts it into 7-bit bytes. A 7-bit byte can only represent the values 0-127. The 7-bit bytes are used in MIDI for SysEx parameters and NPRNs (Non-Registered Parameter Number). In order to represent values larger than 127 you need to use multiple bytes. A large number like 16383 can be split into two bytes (127,127). Four 7-bit bytes are the max for backwards compatibility with 32-bit versions of FileMaker Pro and FileMaker Advanced.

**Parameters:**

BytesToConvert	- A coma delimited list of numbers.
NumOfBytes	- The number of bytes to return from "1" to "4".
LSBFirst	- Calculate with the LSB first ("true" or "false").

**Result:** A comma delimited list of numbers.

**Examples:**

```
// Convert the number into 2 bytes, the LSB is first.
FM2MIDI_ValueToBytes( 255 ; 2 ; "true" )
```

## FM2MIDI\_Checksum

**Syntax:** FM2MIDI\_Checksum( NumbersToSum )

**Description:** Sums a string of comma delimited numbers and returns the remainder of a mod function with a divider of 128.

Typically this is used to calculate checksum of a stored patch. Most manufactures will just sum the total of the parameters. Roland typically includes the MIDI header information as well. You will need to check the MIDI implementation for your specific instrument.

**Parameters:**

NumbersToSum - A comma delimited list of whole numbers.

**Result:** A whole number between 0 and 127.

**Example:**

```
// Sums number list and returns Mod of 128  
FM2MIDI_Checksum( "32,64,21,36,19,42,64,32" )
```

**FM2MIDI\_SysexWrite**

**Syntax:** FM2MIDI\_SysexWrite( DataToSave ; PathToFile )

**Description:** Writes data to a Sysex file.

Creates a new file if one doesn't exist and opens the file for writing. Converts a comma-delimited list of integers to bytes as it writes the data to the file. The Sysex data must be a complete Sysex message beginning with 240 and ending with 247.

Note: Use a file plugin by 24U or Troi to allow the user to set the file name & path.

<http://www.24usoftware.com>

<http://www.troi.com>

**Parameters:**

DataToSave	- A comma-delimited list of integers between 0 and 255.
PathToFile	- A file path, typically from the root directory.

**Result:** 0

**Example:**

```
// Write the data to the Sysex file.
FM2MIDI_SysexWrite( "240,64,0,0,0,3,0,0,247" ; "/Users/Me/Desktop/
MyFile.syx" )
```



## FM2MIDI\_SysexRead

**Syntax:** FM2MIDI\_SysexRead( PathToFile )

**Description:** Opens a Sysex file and returns the file data.

Opens the file specified by the PathToFile parameter and returns the data. Each byte is converted to an integer and returned as a comma-delimited list.

Note: Use a file plugin by 24U or Troi to get a file path from the user.

<http://www.24usoftware.com>

<http://www.troi.com>

### Parameters:

PathToFile - A file path, typically from the root directory.

**Result:** If successful returns the file data.

### Example:

```
// Read the file and return the data.  
FM2MIDI_SysexRead( "/Users/Me/Desktop/MyFile.syx" )
```

**FM2MIDI\_Version**

**Syntax:** FM2MIDI\_Version( { Format } )

**Description:** Returns the Plugin version in a number of different formats.

Use this function to determine the Plugin version installed. Use no parameters for the basic format.

**Parameters:**

Format                      - The format you would like returned.

**Result:**                  The plugin version in the format specified.

**Example:**

```
// Get the plugin version (short) format.  
// Returns "2.2.0"  
FM2MIDI_Version( "short" )
```

```
// Get the plugin version (Long) format.  
// Returns "FileMaker MIDI Plugin 2.2.0"  
FM2MIDI_Version( "long" )
```

```
// Get the plugin version (autoupdate) format.  
// Returns "02020000"  
FM2MIDI_Version( "autoupdate" )
```

**HEX Conversion Chart**

HX	INT	HX	INT	HX	INT	HX	INT	HX	INT	HX	INT	HX	INT
00	0	10	16	20	32	30	48	40	64	50	80	60	96
01	1	11	17	21	33	31	49	41	65	51	81	61	97
02	2	12	18	22	34	32	50	42	66	52	82	62	98
03	3	13	19	23	35	33	51	43	67	53	83	63	99
04	4	14	20	24	36	34	52	44	68	54	84	64	100
05	5	15	21	25	37	35	53	45	69	55	85	65	101
06	6	16	22	26	38	36	54	46	70	56	86	66	102
07	7	17	23	27	39	37	55	47	71	57	87	67	103
08	8	18	24	28	40	38	56	48	72	58	88	68	104
09	9	19	25	29	41	39	57	49	73	59	89	69	105
0A	10	1A	26	2A	42	3A	58	4A	74	5A	90	6A	106
0B	11	1B	27	2B	43	3B	59	4B	75	5B	91	6B	107
0C	12	1C	28	2C	44	3C	60	4C	76	5C	92	6C	108
0D	13	1D	29	2D	45	3D	61	4D	77	5D	93	6D	109
0E	14	1E	30	2E	46	3E	62	4E	78	5E	94	6E	110
0F	15	1F	31	2F	47	3F	63	4F	79	5F	95	6F	111

HX	INT	HX	INT	HX	INT	HX	INT	HX	INT	HX	INT	HX	INT
80	128	90	144	A0	160	B0	176	C0	192	D0	208	E0	224
81	129	91	145	A1	161	B1	177	C1	193	D1	209	E1	225
82	130	92	146	A2	162	B2	178	C2	194	D2	210	E2	226
83	131	93	147	A3	163	B3	179	C3	195	D3	211	E3	227
84	132	94	148	A4	164	B4	180	C4	196	D4	212	E4	228
85	133	95	149	A5	165	B5	181	C5	197	D5	213	E5	229
86	134	96	150	A6	166	B6	182	C6	198	D6	214	E6	230
87	135	97	151	A7	167	B7	183	C7	199	D7	215	E7	231
88	136	98	152	A8	168	B8	184	C8	200	D8	216	E8	232
89	137	99	153	A9	169	B9	185	C9	201	D9	217	E9	233
8A	138	9A	154	AA	170	BA	186	CA	202	DA	218	EA	234
8B	139	9B	155	AB	171	BB	187	CB	203	DB	219	EB	235
8C	140	9C	156	AC	172	BC	188	CC	204	DC	220	EC	236
8D	141	9D	157	AD	173	BD	189	CD	205	DD	221	ED	237
8E	142	9E	158	AE	174	BE	190	CE	206	DE	222	EE	238
8F	143	9F	159	AF	175	BF	191	CF	207	DF	223	EF	239