MIDI Museum 1.10

a shareware MIDI librarian

by Philippe Gagné, ©1994

Philippe Gagné
86 Boul Université Ouest
Chicoutimi, Qc
Canada
G7J 1T3

email: gagne@phy.ulaval.ca
gagne@fourier.phy.ulaval.ca

1 Presentation

1.1 Introduction

Welcome to MIDI Museum! With this program, you can save and retrieve the digital information stored in your MIDI synthesizer, sequencer, beat box, etc. It includes a small programming language that enables you to use it with any kind of MIDI hardware.

This is a shareware program: if you use it, I ask you to send me $30. I will then mail you (or email if you ask for it) a password to disable the opening registering window. Please, remember that it took me more than 6 months to write this program and by sending me your shareware fee you'll encourage me to write new programs and improve old ones.

THIS SOFTWARE AND ACCOMPANYING WRITTEN MATERIAL (INCLUDING THIS MANUAL) ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. FURTHER, THE AUTHOR DO NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF SOFTWARE OR WRITTEN MATERIALS IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU.

1.2 What do you need to use MIDI Museum

Two things are needed to use MIDI Museum: a MIDI interface and the Apple MIDI Manager. You can find the Apple MIDI Manager in some demo version of commercial software. They are easy to find in various bbs: for example

the applications “MiBAC™ Jazz Demo” or “Lime”. You can also buy it from your authorized Apple Dealer.

2 Using MIDI Museum

2.1 What is MIDI?

There is nothing magical about MIDI. It's just a standard way for synthesizer to exchange numbers. In other words, MIDI is a digital interface used to communicate information among synthesizers. This information can be either voice messages (note-on, note-off, pedal, etc.), or system messages (patches, performance data, sequences, etc.).

Like in the computer world, those messages are numbers, called bytes, ranging from 0 to 255. It is easier not to write bytes in decimal form, but in hexadecimal, *i.-e.* in 16–base. In hexadecimal, you count as in decimal, except you have six new numbers: A, B, C, D, E and F. Lets try it and count to 32 (in base 10):

```
0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A,
0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15,
0x16, 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x20
```

Note that we have prefixed each number with “0x” to indicate it is in hexadecimal. The number 0x10 is 16 (in base 10), the number 0x20 is equal to 32 (in base 10) and 0xFF means 255 (in base 10).

Lets look at the seven MIDI voice messages. The highest number (the one on the left) is the kind of message and the lower one (on the right side of the byte) is the MIDI channel:

Byte #1	Byte #2	Byte #3	Meaning
0x8n	note number	velocity value	Note off
0x9n	note number	velocity value	Note on
0xAn	note number	pressure value	Polyphonic key pressure (aftertouch)
0xBn	control number	value	Control change
0xCn	program number		Program change
0xDn	pressure value		Channel pressure (aftertouch)
0xEn	value (LSB)	value (MSB)	Pitch bend change

Note: n is the MIDI channel (a value from 0 for channel 1, to F for the channel 16)

For example, to play a loud middle C on a synthesizer set to respond to MIDI channel 3, the computer will sends the MIDI message: 0x92 0x3C 0x7f. To release that note, it will sends a note–off: 0x82 0x3C 0x00. It is important to note that the MIDI voice messages are real-time, *i.-e.* the time relationship of their occurence is important.

When a synthesizer send its internal memory content, it uses another kind of MIDI message: the System–Exclusive, that we will write here “SysEx”. A SysEx begins with a 0xF0, then there is a byte indicating who is the manufacturer of the synthesizer, a certain number of data bytes will follow and the SysEx will terminate with a 0xF7 byte. Unlike the voice messages, the SysEx is not real-time.

You use a MIDI librarian, like MIDI Museum, to receive, transmit and save on disk the MIDI system messages (they are in fact computer data). In other words, MIDI Museum replaces the “ram cartridges” that many synthesizer use.

For some synthesizers, the communication of the data is simple: you press on a button (often named “SysEx dump” or “Memory dump”) on the synthesizer panel, then it sends its memory data in a big SysEx packet that MIDI Museum will grab and save in the computer (*i.-e.* MIDI Museum will receive{0xF0, all the data, 0xF7}).

Unfortunately, for some other synthesizer, the protocol used to communicate information is a lot more complex: send a “Want to send a file” block, receive a “Ready” block, send first block of data, receive an acknowledge block, send second block of data, etc. (*i.-e.* MIDI Museum must sends {0xF0 “send file” 0xF7}, then receive {0xF0 first data block 0xF7}, then send {0xF0 “acknowledge” 0xF7}, etc.)

There is a small programming language included in MIDI Museum that enable you to tell the computer how to communicate with your synthesizer. This will be explained in a a following section.

2.2 The MIDI Museum windows

There is two kinds of window in MIDI Museum: “SysEx” and “Config”. The first one contains MIDI messages, while the other one is a (very) simple text editor where you type the definition of a new MIDI protocol.

2.2.1 The SysEx window

This is what a SysEx window looks like:

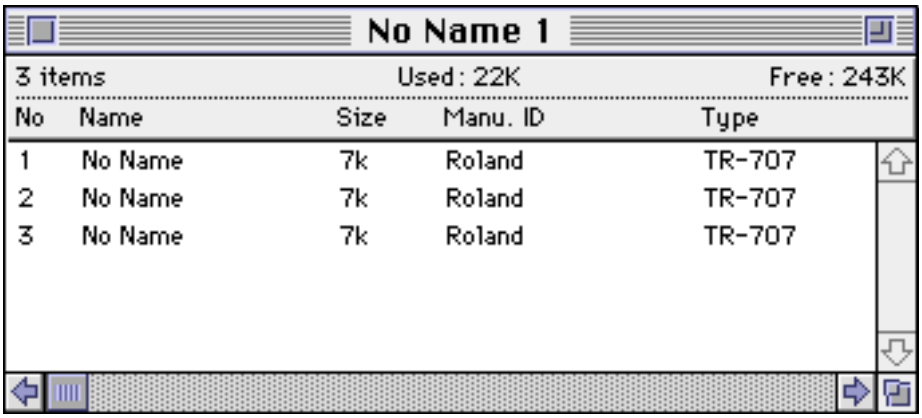

Fig. 1: A SysEx window

A new one will appear when you select the “File”–“New” menu or when you press ⌘–N. You can read from disk a saved one by selecting the “File”–“Open” menu or pressing ⌘–O.

Each line of that window is a different MIDI message. You select a MIDI message with a single click of the mouse on the chosen line. You can then use the “Cut”, “Copy”, “Paste” and “Clear” command to move, delete or copy it. You can also use the

"Edit"–"Transmit SysEx" menu or the ⌘–T command, or click on the remote-control's left icon to transmit the selected message to the synthesizer.

If you double-click on a line (or use the "Edit"–"Rename SysEx" menu, or press ⌘–R) a dialog will appear and you will be able to change the name of that message.

When you want to add a MIDI message to your SysEx window, you must first select the appropriate protocol that will be used (in the "Protocol" menu), then use the "Edit"–"Receive SysEx" menu or the ⌘–R command, or click on the remote-control's right icon to start the MIDI reception. Depending on the chosen protocol, you may have to press the right button on the synthesizer to start the transmission.

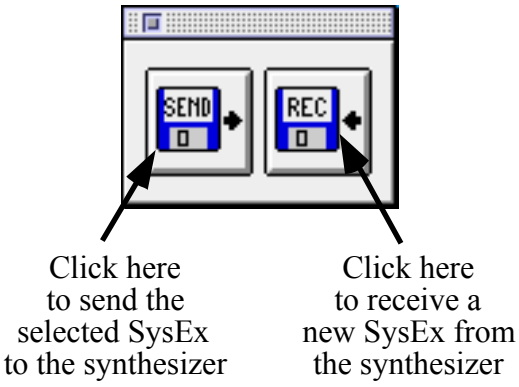

Fig 2: The remote-control window

You use the "File"–"Save" or "File"–"Save as" commands to keep the changes you have made to the SysEx window.

2.2.2 The Config window

This is what a Config window looks like:

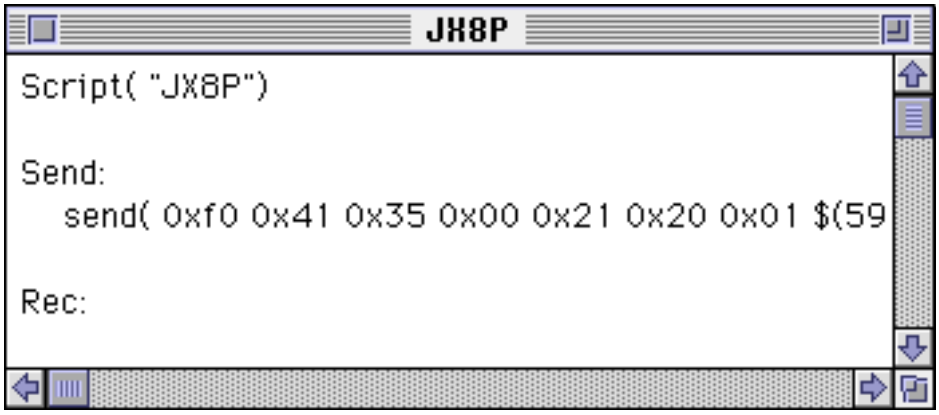

Fig 3: A Config window

You'll get a new one by selecting the "File"–"New Config" menu and you can open a saved one with the "File"–"Open Config" command. You use that type of window as a simple text editor. The "Cut", "Copy", "Paste" and "Clear" commands are supported.

When your script is ready, you must compile it with the "Protocol"–"Compile" command, or ⌘–K. The resulting binary protocol is then stored in the MIDI Museum preference file (in the system folder),ready to be used again and again.

If you want to remove an old protocol, use the "Protocol"–"Delete protocol" command.

3 The MIDI Museum language

We will now describe the simple programming language embedded in MIDI Museum. Please, don't be afraid by the words "programming language": read this section carefully and you will be able to use MIDI Museum in any situation, with any synthesizer and without to much pain!

3.1 The language structure

Here is the structure of a MIDI Museum protocol (words in italic are optional):

```
Script( "Protocol name" )
Variable1 = constant constant constant…
Variable2 = constant constant constant…
Variable3 = constant constant constant…
        etc…
Rec:
        commands…
Send:
        commands…
```

The script name is the name that will appear in the "Protocol" menu. That name has a 30 letters limit.

The optional variables definitions that follow are useful to simplify the script. The protocol's two main sections follow: the receive section (the commands following de "Rec:" keyword) and the send section (the commands following de "Send:" keyword).

3.2 The numbers

A MIDI transmission is a serie of numbers, each between 0 and 255. They are called "bytes". So 1 byte is 1 number, 10 bytes are 10 numbers, etc. There are 3 ways you can write numbers in MIDI Museum: normal (0, 10, 200), hexadecimal (in base sixteen noted 0x*nn*) and in binary (in base two, noted 0b00100101).

Normal (10 base)	Hexadecimal (16 base)	Binary (2 base)

0	0x00	0b00000000
1	0x01	0b00000001
2	0x02	0b00000010
3	0x03	0b00000011
4	0x04	0b00000100
5	0x05	0b00000101
…	…	…
9	0x09	0b00001001
10	0x0A	0b00001010
11	0x0B	0b00001011
12	0x0C	0b00001100
13	0x0D	0b00001101
14	0x0E	0b00001110
15	0x0F	0b00001111
16	0x10	0b00010000
…	…	…
255	0xFF	0b11111111

3.3 The commands

MIDI Museum understand four commands:

Send(*data*)	to send *data* to MIDI
Rec(*data*)	to Receive *data* from MIDI
Repeat(*n, Commands*)	to repeat *n* times the *commands* (those commands are any combination of send and rec)
Prompt(*string*)	to display a *string* on the computer screen. It can be useful to ask the user to press on a synthesizer's button.

“data” is a string of numbers in any combination of hexadecimal, decimal or binary. Optionally you can use variables (more on variables later in this section). Example:

```
0xF0 32 0b10001100 37 0xF7
```

“string” is a character string, limited to 30 lettres, numbers or symbols (!@ %&*()). You must enclose a string between “ " ”. Example:

```
"This is a string"
```

3.4 Defining a protocol

We will now see how to use the protocol definition commands by working out the Roland TR-707 protocol.

It was said earlier that in the simplest case of MIDI communication, the synthesizer sends all its information in one big packet. But there is other synthesizers for which MIDI Museum has to send a block to ask the memory data, then the synthesizer sends it in a number of blocks, one at a time, waiting for acknowledgment from the computer.

The Roland TR-707 uses that kind of complex protocol:

Ask a file	Computer sends to TR-707	0xF0 0x41 0x51 0xF7
Repeat 14 times	Receive a data block	0xF0 0x41 0x52 0x02 (514 Data bytes) 0xF7
	TR-707 says that it has other data	0xF0 0x41 0x54 0xF7
	Computer sends TR-707 an acknowledge	0xF0 0x41 0x53 0xF7
Last data block	receive the last data block	0xF0 0x41 0x52 0x02 (514 Data bytes) 0xF7
	TR-707 says that it has no other data	0xF0 0x41 0x55 0xF7
	computer sends TR-707 an acknowledge	0xF0 0x41 0x53 0xF7

That protocol can be coded in MIDI Museum as:

```
Script( "TR-707")

Rec:
      Send( 0xf0 0x41 0x51 0xf7 )
      Repeat( 14,
            Rec( 0xf0 0x41 0x52 0x02 $(514) 0xf7 )
            Rec( 0xf0 0x41 0x54 0xf7 )
            Send( 0xf0 0x41 0x53 0xf7 )  )
      Rec( 0xf0 0x41 0x52 0x02 $(514) 0xf7 )
      Rec( 0xf0 0x41 0x55 0xf7 )
      Send( 0xf0 0x41 0x53 0xf7 )
Send:
```

The `Send( )` keyword sends the numbers in the parenthesis to the synthesizer. The `Rec( )` keyword makes MIDI Museum waits for a MIDI packet, then check that its content is the same as the parenthesis content or else there will be an error and the transfer will be aborted.

You can see in some number strings the "`$`" or "`$(`*number*`)`". It's only when MIDI Museum reads a "`$`" keyword that it will actually record MIDI data. "`$(`*number*`)`" will store *number* data, while "`$`" records all data from there to the end of MIDI packet. If the synthesizer sends 0xf0 0x44 0x33 0xf7 and MIDI Museum has that command: Rec(0xf0 $), it will save to the SysEx window the bytes 0x44 0x33 0xf7. So it doesn't save the whole message. To send it back correctly, you will have to use the Send(0xf0 $) command.

There is the `Repeat(` *number*, *commands* `)` keyword, that repeats *number* times the *command(s)* `Rec()` and `Send()`.

Finally, there is the last keyword: `Prompt(` "*string*" `)` used to display a message on the computer's screen. When you have read the message, don't forget to click on the OK button to close the prompt dialog and let the computer continues its work.

There is a lot of number strings in that protocol, making it hard to read. You can simplify it if you use variables. This is what the TR-707 protocol looks now:

```
Script( "TR-707")

WSF = 0xf0 0x41 0x50 0xf7
RQF = 0xf0 0x41 0x51 0xf7
DAT = 0xf0 0x41 0x52 0x02 $(514) 0xf7
PAS = 0xf0 0x41 0x53 0xf7
CNT = 0xf0 0x41 0x54 0xf7
EOF = 0xf0 0x41 0x55 0xf7

Rec:
      Send( RQF )
      Repeat( 14,
            Rec( DAT )
            Rec( CNT )
            Send( PAS )  )
      Rec( DAT )
      Rec( EOF )
      Send( PAS )

Send:
```

The "RQF = 0xf0 0x41 0x51 0xf7" define the variable's name (RQF) and its content (0xf0 0x41 0x51 0xf7). From now on, each time MIDI Museum encounters the word "RQF" in that protocol, it will replace it with its content. It means that Send(RQF) is now understand as Send(0xf0 0x41 0x51 0xf7).

3.5 Some protocol examples

Lets now see some protocol examples, from the simple to the complex.

3.5.1 A generic protocol

It reads all data from the synthesizer in one shot. You must start the download from it (find on it a button labeled “memory dump” or see the owner’s manual).

```
Script("Anything")

Send:
      Send( $ )

Rec:
      Rec( $ )
```

To use that protocol, start the MIDI reception, then press the “memory dump” button on your synthesizer. But there is a problem with that protocol: it will read just one MIDI packet, without checking if its a “SysEx” packet. If (and it often happens) the synthesizer sends the “Memory dump key press” code, what this protocol will record is the single key press, not the “SysEx” that follow.

A better generic protocol would be:

```
Script("Generic")

Send:
      Send( 0xF0 $ )

Rec:
      Rec( 0xF0 $ )
```

MIDI Museum will now wait for a MIDI packet starting with 0xF0, *i.-e.* the start of a SysEx packet and records the following bytes, until the end of System-Exculsive code (the 0xf7 byte). It is important to note that here, the first byte isn't written. That's why you must sends it on the send section (Send(0xF0 $)).

3.5.2 The DX7 protocol

The Yamaha DX7 has a very simple protocol: it sends all of its memory in one shot. The DX7 protocol is like the Generic one, except that we will not save the first bytes, we will use them to identify the received message.

```
Script("DX7 - bank")

Send:
      Send( 0xF0 0x43 0x00 0x09 $ )

Rec:
      Rec( 0xF0 0x43 0x00 0x09 $ )
```

To use it start the MIDI reception of MIDI Museum, then start the "memory dump" of your DX7. Note that this protocol secifically wants a MIDI message begining with 0xF0, 0x43 0x00 0x09. Those last two bytes only occurs when the DX7 dwonloads a 32-sounds bank.

You can create a separate protocol to handle the case of the single patch message, or you can use a protocol less stringent, that will accept any Yamaha message:

```
Script("DX7")

Send:
      Send( 0xF0 0x43 $ )

Rec:
      Rec( 0xF0 0x43 $ )
```

3.5.3 The TR-707 protocol

You already know it form section 3.4:

```
Script( "TR-707")

WSF = 0xf0 0x41 0x50 0xf7
RQF = 0xf0 0x41 0x51 0xf7
DAT = 0xf0 0x41 0x52 0x02 $(514) 0xf7
PAS = 0xf0 0x41 0x53 0xf7
CNT = 0xf0 0x41 0x54 0xf7
EOF = 0xf0 0x41 0x55 0xf7

Rec:
      Send( RQF )
      Repeat( 14,
            Rec( DAT )
            Rec( CNT )
            Send( PAS )  )
      Rec( DAT )
      Rec( EOF )
      Send( PAS )

Send:
      Send( WSF )
      Rec( RQF )
      Repeat( 14,
            Send( DAT )
            Send( CNT )
            Rec( PAS )   )
      Send( DAT )
      Send( EOF )
      Rec( PAS )
```

To use that protocol, just start the MIDI reception. The protocol automatically starts the TR-707 transmission.

3.6 Compiling new protocols

MIDI Museum must compile a newly defined protocol to be able to use it. It's easily done with the "Protocol"–"Compile" menu command.

What happen is that MIDI Museum reads the text, transform it to a binary form it can understand and saves it as a resource in the "MIDI Museum Pref" file located in the system's preference folder.

The name of the protocol is appended to the protocol list (in the protocol menu) and if you select it, it will be used in the next MIDI reception.

Each time you receive a SysEx, the data is stored in the SysEx window, as is the necessary protocol needed to send it back to the synthesizer. It means that you can give your “SysEx” files to your friends, no matter if they have compiled the same protocols as you.

3.7 E-Z protocols

Let's end our discussion on protocols by looking at an easy way to create custum protocols. This technique can be used with synthesizers that sends their data in one packet like the DX7, for example.

First, use the generic protocol to receive one bank. Select the "Change SysEx Name" menu command. In the dialog you will see the exact MIDI message length. Note it. In that example (DX7), you should read 4096.

Now, open a Prog window and type:

```
Script("DX7 - one bank")

Send:
      Send( 0xF0 $( 4095 ) )

Rec:
      Rec( 0xF0 $( 4095 ) )
```

That script says to read 4096 bytes:1 header byte (0xf0) plus 4095 data bytes. You can now compile that script and use it when you receive MIDI data from your DX7.

4 Import and exporting data

With MIDI Museum you can read any disk file and put it in a SysEx window. You can also create new messages. To do that there are two commands: “File”–“Import File” command and the “File”–“Import from Script” command.

The “File”–“Import File” command is used to read any disk file and include it as a message in the active SysEx window.

You can use the “File”–“Import from Script” command two ways: to create a MIDI message directly from a script in a Config window, or to read a disk file and append to it bytes taken from a Config window.

4.1 Creating a MIDI message

Say that you want to have a message to change the patch number to the number 3, MIDI channel 1. It translates into the MIDI message 0xC0 0x03. How to make it?

Open a new Config window and give a name to that message by typing:

```
Script( "Patch #3")
```

Then, on the next line, type:

```
0xC0 0x03
```

Click now on the SysEx window in which you want to put that message to make it the active window. Then select the “File”–“Import from Script” command. The new message will be added to the selected SysEx window.

Something more useful would be to send a MIDI “start” message to your sequencer. The script would be:

```
Script( "Start")

0xFA
```

Note that the “File”–“Import from Script” command is enabled only if there is one and only one open Config window.

4.2 Include a file

When there is no open Config window, select the “File”–“Import File” command. The computer will then ask for the file to read and add it the active SysEx window.

4.3 Include a file and append some bytes

The file you want to import needs to have some bytes added. For example, you downloaded a DX7 file with the header stripped (i.–e. it lacks the beginning 0xF0 0x43 0x00 0x09).

Open a new config window, type:

```
Script( "The DX-7 file")
```

Then, on the next line, type:

```
0xF0 0x43 0x00 0x09 $
```

Then click on the SysEx window that will receive the File and select the “File”–“Import from Script” command.

Note again that the “File”–“Import from Script” command is enabled only if there is one and only one open Config window.

4.3 The export command

The export command is not a complicated one. You select it and it will save the SysEx data of the currently selected SysEx message in a file. You can then use it in other programs.

Be aware that it save only the data marqued actully recorded. For example, if your receiving protocol is something like that:

```
Rec( 0xF0 $ )
```

MIDI Museum recorded all the packet data EXCEPT 0xF0, the packet. header. So in the export file everything will be written except 0xF0.