**MIDI Programming - A Complete Study  
Part 1 - MIDI File Basics**

Written by Stéphane Richard (Mystikshadows)

**INTRODUCTION:**

What are MIDI files exactly? We've all seen them at one point or another. These files have the .mid extension and hold data to play music, on a computer, through a sound card's Sound Synthesis electronics. If the sound card is of atleast good quality, MIDI files can play quite nicely. Not to mention that MIDI files hold the most amount of music related information in the smallest size. This is often the reason why they are used in games to hold musical sound tracks pertaining to the game.

As we all know, FreeBASIC supports a library called FMOD which allows us to play MIDI files, however, this is not what this series is about. We will be going beyond what FMOD allows in terms of MIDI files. This series is created for those of you wishing to know everything there is to know about the MIDI standard, the file formats, how to create, save and load the MIDI file format in order to allow the manipulation of it's contents at the byte level. With this knowledge, you'll understand enough about the MIDI files and how to support them in your programs without the need of external libraries such as FMOD. I will use The Windows API directly and detail everything you're likely to find in a MIDI for sound expression, tempo control and other MIDI related functionality so you can have complete control over what you want to do with a MIDI File.

This first part of the series will cover some basic knowledge and concept about MIDI and MIDI files which will serve as background knowledge to build on in the other parts of the series. So let's get started, we have alot of ground to cover. Basically, I'll be packing this first part with as much information that you're gonna need in the next part of the series. This series is for anyone interested in creating specific MIDI related applications whether it's simple tools, to full fledged software sequencers, to anything else that related to MIDI and Software Development. To those of you I say: "You wanted the gory details, the how tos and the what not? Here they are!".

**WHAT IS MIDI:**

M.I.D.I. stands for (Musical Instrument Digital Interface) and is essentially a standard that was created when electronic music was first created. When electronic music surfaced, back in the 70s, the need for two instruments to communicate quickly became apparent. Likewise, new instruments were created and obviously a need for a certain standard so that instruments created by different manufacturers could communicate in a networked environment so they could be controled from other instruments as easily as possible was also becoming quite mandatory.

Therefore, shortly after that, a group of electronic musical instruments manufacturers grouped together and created what we know to day as the [**MIDI Manufacturers' Association**](http://www.midi.org/). Since their creation, the association has been busy creating a standard means of communications among MIDI gear and Computers as well as allowing each manufacture to bring their own brand of uniqueness through the use of system exclusive data. So far, the creation of this association has brought quite some great advantages to the MIDI world.

Also, on typical MIDI gear (keyboard, sequencers, sound cards that support MIDI and the likes), a single MIDI port has has 16 channels available to send and/or receive data from. Technically, all channels have the ability to send and receive any type of data. However, it has been accepted as the norm that channel 10 be reserved for all drums and percussion information. So, if you music is to have drums, you want to put those on channel 10 just to avoid problem with the rest of your MIDI equipement which just might expect drums to be on channel 10.

**THE MIDI STANDARDS:**

Throughout the years, since the first MIDI standard was created, MIDI demands and needs have grown and somewhat changed to accomodate the growing needs of electronic MIDI music and their manufacturers. So then, let's take a look at the four most wide spread MIDI Standards.

* **General MIDI Level 1:**  
  This is the first Standard every created. In essence, this standard gave us the 128 instrument (called programs or patches) that we all know today. It also gave is the 64 percussion patches that any MIDI file can have. Finally it gave us some standard set of C.C.s (Continuous Controlers) that most MIDI programs offer today. In other words, anything MIDI you do today using any software out there wouldn't be there without this first standard. This also defines Channel 10 for drums and percussions.
* **General MIDI Level 2:**  
  After a couple of years, synthesizer technology evolved quite alot, it wasn't too long after that that 128 sounds weren't enough to answer the sound needs of today's musician. Some manufacturers of MIDI gear added their own twists and magic so to speak to offer more sounds, more features into their equipement that answered these needs. A bit after that, the M.M.A. started the process of creating the General MIDI Level 2 standard to accomodate for the new features since most manufacturers already added alot more than was available in the G.M. level 1 standard. G.M. Level 2 adds such things as Bank Switching (to have different banks of sound patches), RPN (Registered Parameter Number) to allow more control over the expressiveness of a sound and NRPN (Non Registered Parameter Numbers) to allow for the manufacturers to add their own blend of special features. G.M. Level 2 is completely G.M. Level 1 compliant.
* **Roland GS Standard:**  
  Once the G.M. Level 2 was established, Roland created it's own brand of sub standards (especially in System Exclusive data) And NRPNs to be implemented in all new MIDI gear from then on out. In essence, the Roland GS standard is G.M. Level 2 plus Roland specific features and functionality. Since the Roland GS specification isn't obvious to come by. It's best to go for G.M. Level 2 atleast until Roland makes the Roland GS standard publically available to everyone.
* **Yamaha XG Standard:**  
  Like Roland, Yamaha also created their own set of specific NRPN and Sysex features in order to give their MIDI gear their own blend of Yamaha specific Features. Yamaha is very open in sharing their specifications with anyone willing to have them. As such they have made their XG specifications available [**right here**](http://www.petesqbsite.com/sections/express/issue18/www.yamaha.co.uk/eurohome/library/midi/xgspec.pdf). Feel free to get this to see exactly what XG implements from G.M. Level 2 and adds to it too.

**THE MIDI FILE FORMAT:**

MIDI Files basically are a file format standard that was created to offer a standard means of exchanging data between computers and midi gear as well as between MIDI gear and MIDI gear. MIDI files typically have the .mid file extension. It holds information about the song to be played itself and information that controls the MIDI gear as the song is being played. All and all, a MIDI starts off with a header chunk and then is followed by any amount of track chunks. Let's explain these in details.

* **The Header Chunk:**  
  As the name implies, the header chunk is of course at the beginning of the MIDI file. It is there to help describe the file in three ways. The Header Chunk always looks like this:

4D 54 68 64 00 00 00 06 ff ff nn nn dd dd

which should be broken down as such:

[4D 54 68 64] [00 00 00 06] [ff ff] [nn nn] [dd dd]

* + The first four bytes [4D 54 68 64] are translated directly to "MThd".
  + Right after are the four following bytes [00 00 00 06] and this will always be the same. It represents the four byte size of the header.
  + Next we have 2 bytes [FF FF] which represent the file format. There are three such formats. They are:
    - 0 => single-track
    - 1 => multiple tracks, synchronous
    - 2 => multiple tracks, asynchronous

Single track is fairly self-explanatory - one track only. Synchronous multiple tracks means that the tracks will all be vertically synchronous, or in other words, they all start at the same time, and so can represent different parts in one song. Asynchronous multiple tracks do not necessarily start at the same time, and can be completely asynchronous.

* + The next two bytes [nn nn] is the number of tracks in the midi file.
  + Finally, the last two bytes [dd dd] is the number of delta-time ticks per quarter note. In other words it's the resolution (in time precision) that a note can be played at in a time span. More on this later.
* **The Track Chunk:**  
  After the header chunk is the beginning of the track chunk information. Like the header chunk, the track has a header that can be read as follows:

4D 54 72 6B xx xx xx xx

Which of course should be broken down as follows:

[4D 54 72 6B] [xx xx xx xx]

* + The first four bytes [4D 54 72 6B], like the header chunk, represent "MTrk" and is a header for the track itself.
  + The last four bytes [xx xx xx xx], represent the length of the track in bytes.

After these eight bytes is a series of MIDI events. These events are identical to the actual data sent and received by MIDI ports on a synth with one addition. A midi event is preceded by a delta-time. A delta time is the number of ticks after which the midi event is to be executed. A MIDI event has a structure, let's review that structure right now.

**THE STRUCTURE OF A MIDI EVENT:**

A MIDI event is really (in general) a set of 3 bytes that is interpreted in 4 pieces of information. And this is enough to cover most of the possible MIDI messages you're likely to send to or receive from the MIDI port.

|  |  |  |
| --- | --- | --- |
| **MIDI Event Byte Level Information** | | |
| **Name** | **Lenght And Range** | **Description** |
| StatusByte ChannelByte DataByte1 DataByte2 | [0-F] [0-F] [0-FF] [0-FF] | Type of Midi message MIDI Channel Number First midi message data byte Second midi message data byte |

The status and channel Bytes are merged into one byte (00-FF) Because these messages have an MSB (Most Significant Byte) of 1 the command statuses actually begin at 80 hexadecimal (128 and up to 255) The LSB (Least Significant Byte takes a value of 0-F Hexadecimal (0 to 15) to specify which MIDI channel the command will be sent to. A command message tells the MIDI gear to perform certain types of things like play a note, change the volume, add effects, and other types of things. This table shows the different command status and what they do.

|  |  |  |
| --- | --- | --- |
| **Status** | **Expected Data** | **Comments** |
| 8x | note, velocity | Note off |
| 9x | note, velocity | Note on (velocity 0 = note off) |
| Ax | note, value | Polyphonic pressure |
| Bx | controller, value | Controller change |
| Cx | program | Program change |
| Dx | value | Channel pressure |
| Ex | value (two bytes: LSB then MSB. many devices will accept only one byte which will be interpreted as the MSB.) | Pitch bend |

Now, any of these messages (note on notably) expect a note and a volume. Volume can be 0-7F hexadecimal (0-127). The note also has a possible value of 0 to 127. The table below gives you the note values you can send.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **MIDI Music Note Velocity Chart** | | | | | | | | | | | | |
| **Octave #** | **C** | **C#** | **D** | **D#** | **E** | **F** | **F#** | **G** | **G#** | **A** | **A#** | **B** |
| **0** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **1** | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| **2** | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 |
| **3** | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| **4** | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| **5** | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 |
| **6** | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 |
| **7** | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 |
| **8** | 96 | 97 | 98 | 99 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 |
| **9** | 108 | 109 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 |
| **10** | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 |  | | | |

When you are sending a controller command [Bx cc vv] it is useful to know which controller you are actually sending and what it does. As such, here is another table that explains these controler numbers and what they are used for. This table shows some standardized controller numbers. Controller numbers 0 - 31 are continuous, MSB (most significant byte), numbers 32 - 63 are continuous, LSB (least significant byte), and 64 - 97 are switches.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Hex** | **Dec** | **Comments** |
|  |  |  | Controller numbers 00 - 1f [0 - 31 decimal] are continuous, LSB (least significant byte) |
| Mod Wheel | 01 | 1 |  |
| Breath Controller | 02 | 2 |  |
| Foot Controller | 04 | 4 |  |
| Portamento Time | 05 | 5 |  |
| Data Entry MSB | 06 | 6 |  |
| Volume | 07 | 7 |  |
| Balance | 08 | 8 |  |
| Pan | 0A | 10 |  |
| Expression Controller | 0B | 11 |  |
| General Purpose 1 | 10 | 16 |  |
| General Purpose 2 | 11 | 17 |  |
| General Purpose 3 | 12 | 18 |  |
| General Purpose 4 | 13 | 19 |  |
|  |  |  | 20 - 3f [32 - 63 decimal] are MSB (most significant byte) for 00 - 1f [0 - 31 decimal] |
| Sustain | 40 | 64 | **Momentary Switches** |
| Portamento | 41 | 65 |  |
| Sustenuto | 42 | 66 |  |
| Soft Pedal | 43 | 67 |  |
| Hold 2 | 45 | 69 |  |
| General Purpose 5 | 50 | 80 |  |
| Temp Change (General Purpose 6) | 51 | 81 |  |
| General Purpose 6 | 51 | 81 |  |
| General Purpose 7 | 52 | 82 |  |
| General Purpose 8 | 53 | 83 |  |
| Ext Effects Depth | 5B | 91 |  |
| Tremelo Depth | 5C | 92 |  |
| Chorus Depth | 5D | 93 |  |
| Detune Depth (Celeste Depth) | 5E | 94 |  |
| Phaser Depth | 5F | 95 |  |
| Data Increment (Data Entry +1) | 60 | 96 |  |
| Data Decrement (Data Entry -1) | 61 | 97 |  |
| Non-Registered Param LSB | 62 | 98 |  |
| Non-Registered Param MSB | 63 | 99 |  |
| Registered Param LSB | 64 | 100 |  |
| Registered Param MSB | 65 | 101 |  |
|  |  |  | **Channel mode message values** |
| Reset All Controllers | 79 | 121 | Val ?? |
| Local Control | 7A | 122 | Val 0 = off, 7F (127) = on |
| All Notes Off | 7B | 123 | Val must be 0 |
| Omni Mode Off | 7C | 124 | Val must be 0 |
| Omni Mode On | 7D | 125 | Val must be 0 |
| Mono Mode On | 7E | 126 | Val = # of channels, or 0 if # channels equals # voices in receiver |
| Poly Mode On | 7F | 127 | Val must be 0 |

So far, in all these tables, the events and commands had to be directed to a specific MIDI channel. There are eight commands that do not apply to a specific MIDI channel but rather to the whole MIDI system in general. These are known as System Messages. Note that any non-realtime status byte ends a System Exclusive message; F7 (EOX) is not required at the end of a SysEx message. Realtime status bytes may appear any time in the MIDI data stream, including in the middle of a System Exclusive (SysEx) message. So then, here's a table to list these System Messages.

|  |  |  |
| --- | --- | --- |
| **Status** | **Name** | **Data** |
| F0 | System Exclusive | data, then EOX or any status byte |
| F1 | Time Code | one byte |
| F2 | Song Position Pointer | two bytes: lsb msb |
| F3 | Song Select | one byte: song number 0 - 127 |
| F4 | (undefined) |  |
| F5 | (undefined) |  |
| F6 | Tune Request | no data |
| F7 | EOX (End of System Exclusive) |  |

Finally, there are messages that like system messages do not apply to specific MIDI channels and basically affect the playing of the song in general. these are called rightfully called realtime messages. Here's a table with these messages, remember there is no data to supply to these commands.

|  |  |
| --- | --- |
| **Status** | **Comment** |
| F8 | Clock |
| F9 | (undefined) |
| FA | Start |
| FB | Continue |
| FC | Stop |
| FD | (undefined) |
| FE | Active Sensing |
| FF | System Reset |

With all this information, you should now be better equipped to understand the contents of a MIDI file. There is, however, a very differrent type of information that can reside in a MIDI file. This type of information does not affect the song itself but rather affects the hardware (sound card, keyboard, sound modules, drum machines and the likes) that the song will be played on. These are known as System Exclusive Messsage and this is what we'll be seeing right now.

**ALL ABOUT SYSTEM EXCLUSIVE MESSAGES:**

As I mentionned above, Different music gear manufacturers incorporate their own features into their equipment. The way they allow their equipment to be controlled is with the use of a special set of messages that are known as system exclusive messages. The wide spread term for these messages are **SysEx**. There is no specific structure for a SysEx message per se except that it has a beginning and an end. Here is the structure of the beginning of the SysEx message:

F0 XX nn... F7

Which can be read as:

[F0 XX] [nn...] [F7]

* **[FF]** is the command to indicate the start of a SysEx message.
* **[XX]** is a manufacturer ID (see [here](http://www.midi.org/about-mma/mfr_id.shtml) for a list of manufacturer IDs) to Identify the manufacturer. It will help to compare this against the MIDI gear to see if anything can actually process the message.
* **[nn...]** is a variable length list of bytes, it can be any lenght however, because of the speed of MIDI transmision they suggest keeping a block of SysEx to less than 64Kb to get acceptable response time.
* **[F7]** indicates the end of the SysEx message.

The contents of the SysEx message can be anything the MIDI gear can do. It can be a whole bunch of different things too that would be needed to set the keyboard into the right configuration to play the song for example. Could be asking the keyboard to dump it's current SysEx setup so that it can be used later as another example. SysEx, in itself, becomes more than useful quite quickly when you really want to control, at the finest detail, the performance of a song on a given MIDI gear.

There are four SysEx messages that are known as Universal SysEx Messages. This means that they do not apply to any specific manufacturer. Instead they server to help the whole MIDI system (or the computer) get information about the MIDI gear that is connected to the MIDI system. Here are these Universal SysEx messages.

* **General MIDI Enable:**  
  Today, a majority of keyboard offer their own set exclusive sounds. Alot of them internally support G.M. sound patches and in order to gain access to those, you eed to set the keyboard up to turn G.M. sound patches on. Here is the structure of this SysEx message.
* 0xF0 SysEx
* 0x7E Non-Realtime
* 0x7F The SysEx channel. Could be from 0x00 to 0x7F.
* Here we set it to "disregard channel".
* 0x09 Sub-ID -- GM System Enable/Disable
* 0xNN Sub-ID2 -- NN=00 for disable, NN=01 for enable

0xF7 End of SysEx

* **Master Volume:**  
  This Universal SysEx message adjusts a device's master volume. Remember that in a multitimbral device, the Volume controller messages are used to control the volumes of the individual Parts. So, we need some message to control Master Volume. Here it is.
* 0xF0 SysEx
* 0x7F Realtime
* 0x7F The SysEx channel. Could be from 0x00 to 0x7F.
* Here we set it to "disregard channel".
* 0x04 Sub-ID -- Device Control
* 0x01 Sub-ID2 -- Master Volume
* 0xLL Bits 0 to 6 of a 14-bit volume
* 0xMM Bits 7 to 13 of a 14-bit volume

0xF7 End of SysEx

* **Identity Request:**  
  Sometimes, a piece of MIDI equipment might wish to know what other devices are connected to it. For example, a Patch Editor software running on a computer may wish to know what devices are connected to the computer's MIDI port, so that the software can configure itself to accept dumps from those devices. The Identity Request Universal Sysex message can be sent by the Patch Editor software. When this message is received by some device connected to the computer, that device will respond by sending an Identity Reply Universal Sysex message back to the computer. The Patch Editor can then examine the information in the Identity Reply message to determine what make and model device is connected to the computer. Each device that understands the Identity Request will reply with its own Identity Reply message.   
    
  Here is the Identity Request message:
* 0xF0 SysEx
* 0x7E Non-Realtime
* 0x7F The SysEx channel. Could be from 0x00 to 0x7F.
* Here we set it to "disregard channel".
* 0x06 Sub-ID -- General Information
* 0x01 Sub-ID2 -- Identity Request

0xF7 End of SysEx

Here is the Identity Reply message:

0xF0 SysEx

0x7E Non-Realtime

0x7F The SysEx channel. Could be from 0x00 to 0x7F.

Here we set it to "disregard channel".

0x06 Sub-ID -- General Information

0x02 Sub-ID2 -- Identity Reply

0xID Manufacturer's ID

0xf1 The f1 and f2 bytes make up the family code. Each

0xf2 manufacturer assigns different family codes to his products.

0xp1 The p1 and p2 bytes make up the model number. Each

0xp2 manufacturer assigns different model numbers to his products.

0xv1 The v1, v2, v3 and v4 bytes make up the version number.

0xv2

0xv3

0xv4

0xF7 End of SysEx

* **Sample Dump Standard:**  
  Today, many keyboards offer sample based sound patches which means that to play a sound, they don't reply on oscilliators or other forms of sound synthesis to play the sound. Instead they use a digitalization of a sound stored in their internal memory. In orer to gain access to these sample based sounds, the M.M.A. created the SDS standard.  
    
  The SDS standard is a fairly complex system that doesn't simple use a message like the other three Universal SysEx. Instead it has different sets of commands to get different types of information or send different types of information to the Sample Based MIDI gear. To find out what these messages are you can [read this](http://www.borg.com/~jglatt/tech/sds.htm) to get the full details.

**IN CONCLUSION:**

Well I think I've packed enough information for a first part of a series. After you've digested this information there's no doubt in my mind that you'll know alot more about MIDI files than you do now. This is really only the beginning. After you read this once, you should just consider this as a reference to what lies ahead in the series. There's no need to remember every single piece of information (unless you really want to) as you can just refer to this page whenever you like. There was alot of ground to cover and I think we did just that.

In the next part of the series, we'll start putting all this information to good use by creating a program that can read a MIDI file, parse it's contents and put them in a structure we'll define in order to view them and manipulate them as we see fit. So brace yourself, there's plenty of coding up ahead in your future. As I mentioned, there's alot of information here. If anything isn't quite clear, you might want to email me your questions and we'll see to it that all becomes as clear as it can be. Until next time. Happy reading, understanding and coding!.

**MystikShadows**  
Stéphane Richard  
[mystikshadows@gmail.com](mailto:mystikshadows@gmail.com)