MIDI SysEx Tutorial

In an effort to bring System Exclusive (SysEx) messages to the masses, I embarked on a mission to produce this tutorial which will help people plumb the depths of this wonderful and powerful area of MIDI.

To understand SysEx messages you will need to have an understanding of hexadecimal.

The first part deals with the actual contents of the SysEx message and their function in the scheme of things.

The second part deals with the infamous Roland checksum.

By the end of this you may be disappointed that it is all quite easy after all!

System Exclusive (SysEx) Tutorial

Part 1 : The Anatomy of a SysEx Message (or Do this please!)

This discussion on SysEx is aimed at people using Roland equipment, but will put you in good stead to apply the knowledge to other makes of MIDI equipment.

The idea of SysEx is to change settings in a synth that cannot be accessed by any other means. Usually, anything that can be changed in a synth can be done via SysEx, but because of its unwieldy structure and length, the more common parameters can be accessed via controllers (like volume et al) and special events (like patch change, pitch bend et al). SysEx is the only means of retrieving data from a synth.

The Roland SysEx message is made up of nine parts. All notation is in hex, but without the trailing "h" for simplicity's sake.

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
F0	41	10	42	12	40007F	00	41	F7

I will discuss each part briefly before going into detail on the usage of SysEx messages.

Parts **[1]**, **[2]** and **[9]** are part of the MIDI specification and are required by all SysEx messages. What is in between is specific to the manufacturer, identified by part **[2]**, which is 41h in Roland's case.

Part **[3]** is known as the Device ID. Most Roland MIDI devices use the default of 10h, but is provided for us to change as we see fit. The idea behind the Device ID is that if you have more than one MIDI device of the same type in a daisy chain (connected to one another) you can change the Device ID on each of them so that you can send SysEx messages that will be accepted by only one of them, not all.

Part **[4]** is the Model ID. GS synths will all respond to SysEx with a Model ID of 42h, however they generally have their own Model ID as well.

Part **[5]** specifies whether we are sending (12h) or requesting (11h) information. If a synth receives a SysEx message it recognizes, it will look this part to determine whether it needs to change an internal setting or reply (with its own SysEx message) with the value of a setting.

Part **[6]** is the start address on which the SysEx intends to act. It is at this address that you may wish to put a value (or values) or retrieve the current value(s). It always contains three bytes. Most synth manuals will provide you with a long "**address map**" table which explains what lives at each address. Although daunting on a first perusal, once you understand its function it becomes a wonderful resource.

Part **[7]** has two functions. If part **[5]** is 12h (sending data) then part **[7]** contains the *data* we are sending and can be one byte or many bytes in length. If it is 11h (requesting data) then it is the *number* of bytes we want the synth to send us. I will expand on this later with examples.

Part **[8]** is the infamous Roland checksum which gets a whole section to itself in this tutorial.

That, in a nutshell, is what a SysEx message is made up of. If you're still a little hazy then don't be concerned. By way of examples you will see SysEx in action and will soon realize that there is no mystery to SysEx after all.

Example 1 : The GS reset. (or Sending a single byte.)

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
F0	41	10	42	12	40007F	00	41	F7

Open your text books...er...manuals to the address map for your synth. If it is a GS synth, run your finger down the left most column (which usually contains the address location) and find 40 00 7F. Different manuals describe it differently (how consistent!) but in essence it is the famous GS Reset. (The SC-88 manual describes it as "MODE SET 00=GS Reset")

What this message is telling the synth is "put the value 00h at address 40007F". What the synth does in this particular instance is perform an initialization.

Example 2 : Patch Change. (or Sending more than one byte.)

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
F0	41	10	42	12	401100	0801	26	F7

We can send more than one byte of data in a single SysEx message. This is very useful when we want to set the value of several settings that follow each other in the address map. By sending more than one byte in part **[7]**, which is the data part, the first byte will be put in the address specified and the following bytes in each successive address location. Each address location can only hold one byte of data.

In the example above we are putting the value 08h into address 401100h and the value 01h into address 401101h. Think about it this way, if we send more than one byte to an address then each successive byte will be put in the address below the previous one in the map in the manual.

For the above example, the SC-88 manual has the following in the address map:

Address(H)	Size (H)	Data (H)	Parameter	Description
40 1**x** 00	00 00 02	00 - 7F	Tone Number	**CC#00** Value 0-127
40 1x 01#		00 - 7F		**P.C.** Value 1-128

A quick time out here to explain the manual notation:

1. **x** in the address represents the synth part number, 1 though F.
   Thus x=1 for part 1,
   hence 40 11 00 is the start address for part 1.
2. **#** in the address means that we cannot send data to this address, we
   must send multiple bytes to a previous address that does not have a #.
3. **CC#00** is the GS variation number.
4. **P.C.** is the regular patch change.

Now you can see how 08h goes to the address specified in the SysEx message, and 01h goes to the *next* address.

For those with an external Sound Canvas with a display, if you successfully send this example to your synth you will see the patch name "Piano 2w"!

If there is anything you do not understand, please e-mail me and try as best possible to explain what confuses you and what part (if not all) you still don't understand. I have discovered that the way things are explained can make perfect sense to one person, and absolute nonsense to another. It's all to do with the words used and the order of the explanation. (Thank goodness Yoda gave self-defense lessons and not English lessons!)

Example 3 : Receiving data. (or What are you thinking?)

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
F0	41	10	42	11	401100	000002	2D	F7

To find out what value(s) your synth has in an address (or range of addresses) we use the 11h value in part **[5]**.

Part **[6]** identifies the start address that we're interested in.

Part **[7]** is now an indication of the number of bytes we want back, as opposed to being the data we sent to the synth when using 12h in part **[5]**. This part must always contain three bytes (six hex digits).

That's it. The synth will respond with a SysEx message containing the value(s) we asked for.

This example is the converse of example 2. Instead of setting the Varaition/Patch Change, we are asking the synth to tell us what the current setting is. The synth's response is in the format of a SysEx message. The message will be exactly what could have put those values in those addresses in the first place!
In other words, if you successfully send "example 2" to the synth and then send "example 3" to the synth, then the contents of the synth's response will be the bytes that make up "example 2"!!

There should be more examples in your manual which you can now try. Try and predict the result before you send the example. If the result confirms your prediction then your SysEx world has just become a universe!

System Exclusive (SysEx) Tutorial

Part 2 : The Roland Checksum (or Did I get it right?)

The idea behind the checksum is simple. When we send a SysEx message to a synth, we want it to do only what we ask and nothing else. Consider the real world where messages don't always reach their destination in their original form. The results could be disasterous.

If a SysEx message gets corrupted during transmission to the synth, it is entirely possible that it could become a totally different (but valid) message! So a reverb change could conceivably become a GS Reset!!
To avoid this, Roland uses a checksum that is calculated using the **address** *and* **data** parts of the SysEx message, thus ensuring that if the address or data is corrupted somehow, they will no longer match the checksum and the SysEx message will be discarded.
(For the cynics out there: Yes it is conceivable that the address/data *and* the checksum could be corrupted in such a way that a different, valid message is formed. However, the likelyhood is *far* smaller than with no checksum present.)

Calculating the checksum follows a simple formula which is easily learned and applied. Following this formula will gaurantee correct checksums:

1. Convert the hex values to decimal.
2. Add the values together, but if the answer to any sum exceeds 127 then subtract 128.
3. Subtract the final answer from 128.
4. Convert the resulting value back to hex and use as the checksum.

Here is a practical example:

[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
F0	41	10	42	12	401100	4163	0B	F7

Remember the checksum is only calculated from the address **[6]** and the data **[7]**, thus we need only look at 40 11 00 41 63 in this example.

1. Convert hex to decimal:
   40h = **64**
   11h = **17**
   00h = **0**
   41h = **65**
   63h = **99**

2. Add values, keeping result under 128:
   64 + **17** = 81
   81 + **0** = 81
   81 + **65** = 146 (146 - 128 = 18)
   18 + **99** = 117

3. Subtract answer from 128
   128 - 117 = 11

4. Covert to hex:
   11 = 0Bh

It's that simple! Part **[8]** is the checksum and is indeed 0Bh in this instance.

The manual shows a different method which involves slightly more complicated arithmatic. If you prefer this method then by all means use it.

In essence it involves adding all the values together, dividing by 128 and subtracting the *remainder* from 128, if the remainder is greater than 0. Thus:

1. Convert hex to decimal:
   40h = 64
   11h = 17
   00h = 0
   41h = 65
   63h = 99

2. Add values:
   64 + 17 + 0 + 65 + 99 = 245

3. Divide by 128
   245 / 128 = 1 remainder 117

4. Subtract remainder from 128, if remainder is not 0*
   128 - 117 = 11

5. Covert to hex:
   11 = 0Bh

*If the remainder is 0 then the checksum is 0.

If you know for a fact that I have made any mistakes above, please let me know.

Feed back on clarity and semantics will also be greatly appreciated.

Thanks, Eddie Lotter
