Max and Sysex

One of the joys of Max is that it gives easy access to the hidden commands of your synthesizer, those that are only reachable by system exclusive messages.

System exclusive MIDI messages are those that are defined by the manufacturer of the instrument rather than the MIDI association. Usually sysex is very specific to the function of a particular instrument or family of instruments, although there are such things as "universal system exclusive"[1] messages.

You can't assume sysex is working for a particular instrument, or that an obvious feature exists. Since sysex is expensive to implement, manufacturers often leave if off of the budget instruments. You have to look in the fine print in the back of the manual to see what's available. Yamaha, Korg Roland and Emu generally have very complete sysex features. And I'll warn you right now- that fine print often contains mistakes and omissions. Doing sysex often involves some serious detective work.

The Standard

Sysex messages always starts with F0h[2] in the status byte. The next one or three bytes are the manufacturer's ID, which is assigned by the MMA. A good one to know is 7Dh, which is non commercial, and should be used by any home brew devices.

Sysex messages almost always end with F7h. The exception comes with very long messages that some manufacturers break into chunks. In a few cases only the final chunk is terminated with F7h.

What's in between is up to the manufacturer. Typically there will be:

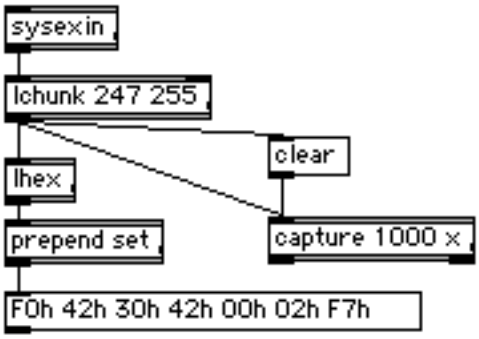
- Model ID, so a Korg NS5 doesn't try to use Trition settings. Sometimes a whole series of instruments, such as the original Proteous family share the same ID.
- Device ID, which the owner sets to distinguish between identical instruments. For many the ID 7Fh means any unit. The ID may be 0 when the user sees 1.
- Command code, or what this message is about.
- Data length, how many bytes to expect. Figuring out exactly what is included in the byte count can be a trick. For instance, is the length itself counted?
- Checksum, a code byte for error correction. Exactly what is included in the calculation, and how the calculation is made can be further mysteries. Sometimes 7Fh is used for "don't worry about the checksum".

[1] One of the better oxymorons. MIDI time code is an example.

[2] That's hexadecimal for 240. If you don't know hexadecimal, look at Max and Numbers. Get used to it, because that's how the specs are written. On the bright side, you seldom have to actually calculate in hex, just recognize some key values.

Sysex in Max

The midiin or sysexin objects will supply sysex as a series of bytes. Generally, I use [Lchunk 247 255] to gather the message into a single list. After that, I drop it into a capture (with a previous clear) if I expect long messages or a message box for short ones. Capture will show data in hex if you add x as an argument. For messages in hex, I use Lhex.



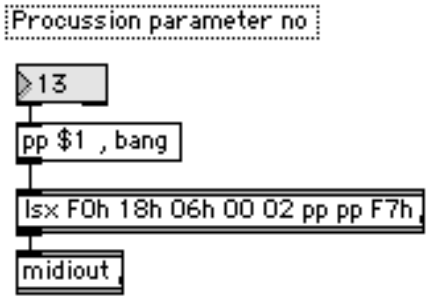
This works for messages up to 255 bytes. If you expect bigger ones, or multi part data dumps, you need to use Ldumpster.

Sending

To send sysex, I prefer to use Lsx instead of sxformat. It has some important features that make life easier:

- You can specify arguments in hexadecimal (as F0h) and they stay that way. With the 0xF0 system, the numbers will revert to decimal when the patch is saved. I seem to make most of my errors while translating decimal to hex and back. (Or more precisely, not translating)
- It handles nibbleization for me. The data bytes must be 0-127, but many parameters need more precision. Then the data is sent as two or more bytes, with 7 or 4 bits of the value in each.
- It handles negative numbers, converting to 2's compliment.
- There's no limit to the number of changeable parameters you can have. Changeable parameter are specified by symbols as args, such as pp. These are replaced with a value input with the symbol. If the symbol appears twice, the value is nibbleized into place.

Here's how I send a parameter value request to a Procussion:



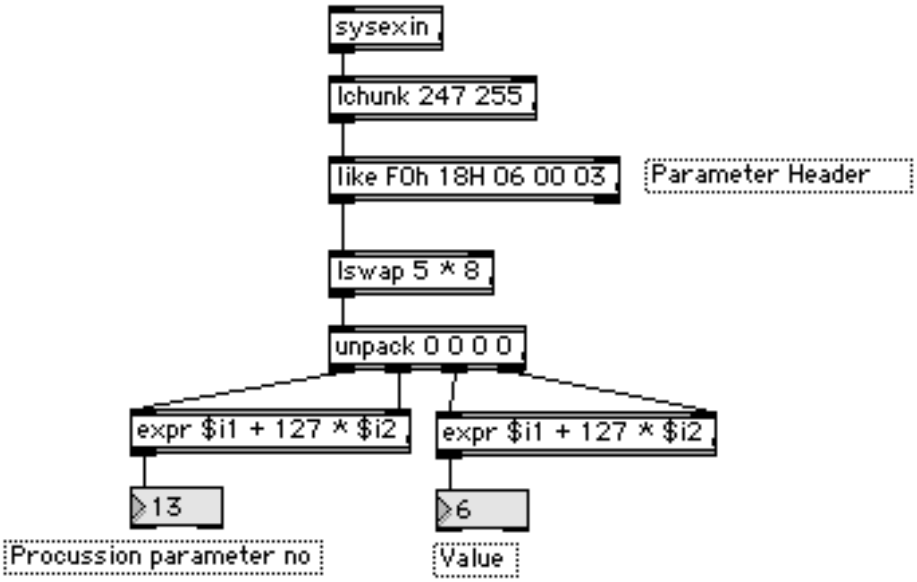
There are two common ways that instruments refer to data, one is by direct memory addressing, the other is via parameter numbers. The Procussion uses parameter numbers, which you get by reading the documentation. Then the message F0 18 06 id 02 pl pm F7 will cause Procussion to return the current value for the parameter. The bytes in that message mean:

F0 System exclusive
18 Emu corporation
06 Procussion
id Device Id
02 Parameter Request
pl the parameter number least significant[3] 7 bits
pm the parameter number most significant 7 bits
F7 End of Exclusive

I can use pp twice in lsx to get the nibbized numbers. There are various ways to nibbleize, with the most significant bits first or last, and with 4 or 7 bits per byte. Lsx has commands to set these modes.

Decoding

To decode the value, I use a patch like this:



The format for the return message is F0 18 06 id 03 pl pm vl vm F7, where pl and pm give the parameter number as before, and the vl and vm are the current value. The Like object will pass lists that start with the arguments[4]. Like also recognizes hex in the 00h format, and will consider any other symbol a wild card,

[3] Last is least in this case.

[4] Due to sloppy typing, I've shown the ways hex numbers can be entered into Like. A big H or little h after a number will do it. With numbers lower than 10 it doesn't matter. Lhex converts lists of numbers into this form, and llong converts them back.

so you can type id in where the device id is expected. The advantage of using Like here is I can sort incoming messages by their headers. Like sends unsuccessful list out the right outlet, so you can cascade them to reduce the number of objects that look at a message.

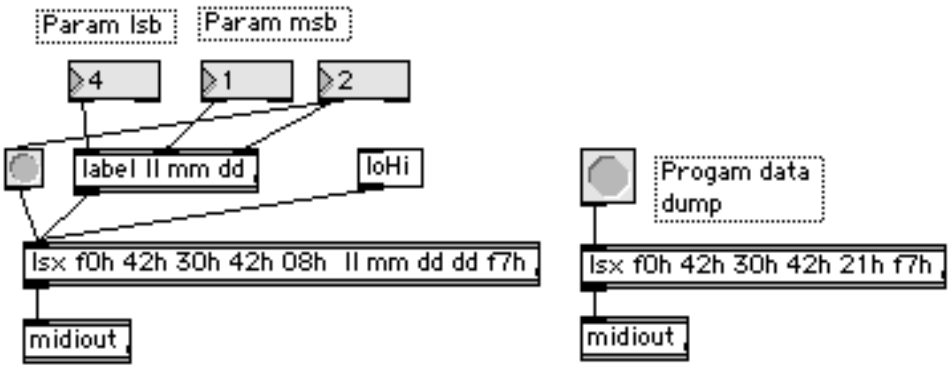
Lswap will extract the interesting bytes, which can be unpacked and converted back to a normal value.

Tricky Instruments

There are other factors that turn up in various sysex implementations. For instance, the Korg NS5 has modes which must be set before you can get at certain values. If you are editing effects, a particular request will set effects values, and if you are editing programs, the same command would set some program value. The message F0 42 30 42 00 02 F7 will put the instrument into program edit mode, and you can set program parameters. That message parses like this;

F0 system exclusive
42 Korg
30 the device ID + 30h (don't know why)
42 NS5 type
00 mode change request
02 Program edit mode
F7 EOX

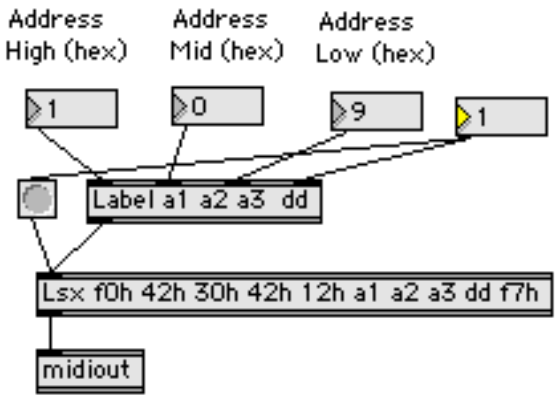
Once the mode is set to program edit, the message F0 42 30 42 21 F7 will get all of the parameters of the current program. The Korg doesn't have a single parameter request, so you have to sift through the whole works to find the number you are interested in. The easiest way to do this turns out to be changing parameters and then getting a dump to see what has changed.



The parameters are listed in a big chart, showing the parameter msb (which is 0 for common, 1 for oscillator 1 and 2 for oscillator 2) and the parameter lsb. There's also an offset number, which I would expect to show the number of bytes into the dump to find that value, but it's wrong. What's really happening is truly bizarre- the sign bits for various parameters that run from -128 to 127 are gathered together into bytes that are scattered through the dump.

Direct Address Mode

Some of the features of the NS5 are reached by memory address rather than parameter number. This is not much different, in this case you just specify a 3 part address (address high, address mid, address low) to reach something. The tables are a bit harder to read, but on some instruments you can set a series of values with a single message.



System Exclusive implementations vary widely from instrument to instrument, so there's not a lot of advice I can offer beyond these basic techniques. But by reading the manual carefully and experimenting with these tools, you should be able to access a lot of new features in your instruments.