Max and MIDI Machine Control

MIDI machine control is a protocol designed to allow remote and automatic operation of tape recorders and similar devices. MMC is intended to control a wide variety of machines, including some not yet invented. The protocol is chock full of such interesting possibilities as deferred variable play and VITC insert enable.

Currently available machines are simpler, but on any you should be able to at least push the panel buttons, and probably ask for the reading on the tape counter. On ADATs you have the ability to control transport activity (even eject the tape), arm and disarm tracks, read which tracks are armed on the first machine (but not from the slaves) read the tape counter, locate to an arbitrary point, and set and goto 8 location points. Darwin does all this, plus variable rate play, and can be set to inform you of changes in status.

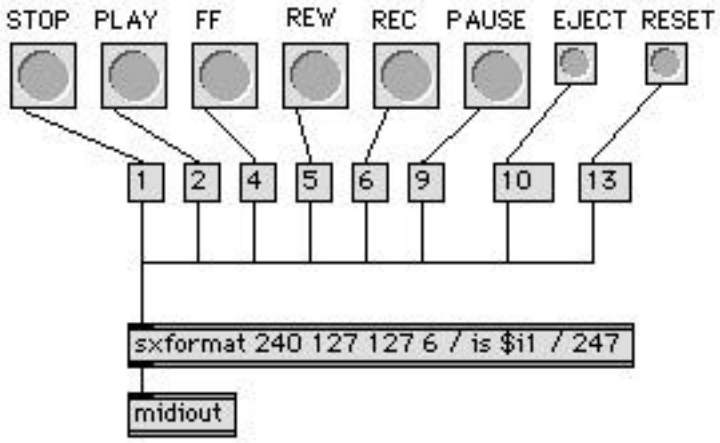
Message Format

MMC messages are another example of Universal System Exclusive messages. The basic format is

{240 127 ID 6or7 <a bunch of data> 247}

The ID may direct this message to a particular machine (0 means unit 1) or an ID of 127 affects everything. 6 in the 4th byte means this is a command, 7 means this is a response to a request for data. Generally, you will send a machine commands and watch for responses.

Transport Commands



The easiest commands are the ones that move the tape around. This patcher implements a simple remote control. The sxformat is the heart of it. It is built on the basic string with the command number as the only data. (I used the universal unit ID. You can set a unit number if desired by inputting it to a right inlet.) You make things happen by passing one of the following into the left inlet:

1 stop
2 play
4 fast forward
5 rewind
6 record Strobe
10 eject

The more obscure transport commands:

3 deferred play. Play when you finish what you are doing.

7 record exit. Punchout

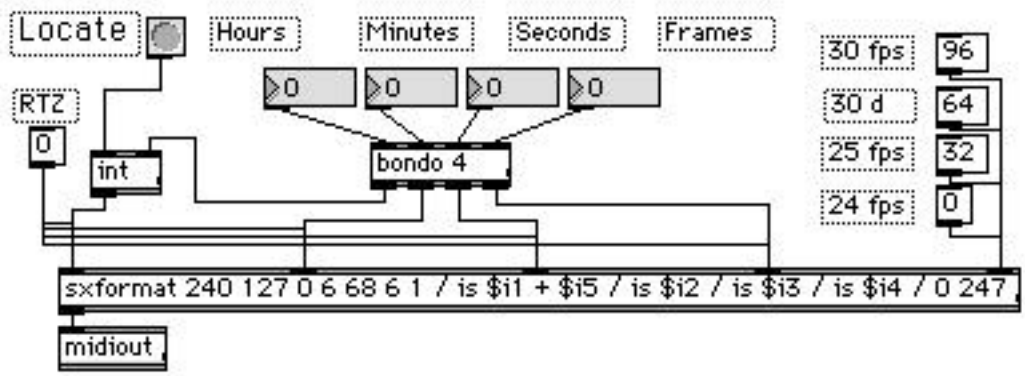
8 record pause. This takes some explaining. On some machines you get into record by going through a sequence Pause, Record Pause, Record Strobe. On ADATs, Darwin, and DA-88 all you have to send is record strobe whether stopped or playing.

9 pause. Most machines make no distinction between stop and pause. Later, when we get motion status from an ADAT, we will see that it reports stop if the tape is unloaded and pause if it is not.

11 Chase. On a machine that is able to lock to SMPTE time code, puts it into chase mode so it follows the master.

13 MMC reset. Might be useful in case the machine gets tangled up and tries to locate going the wrong way or something. More about reset later, when we talk about updates.

Simple Location



The locate command is similar to the transport commands, but comes in two versions. The sysex string is:

{240 127 127 6 68 6 1 hr mn fm ss fs 247 }

The first four bytes you already know.
68 is the command number for locate.
6 means six bytes of data follow
1 means an immediate locate (0 would locate to stored value- we'll do that later). The following data is the target, in an almost familiar format. (If it's not familiar, see the section on MIDI Time Code.)

Hours includes the time code type as usual. When a location time is sent within an MMC command or responses, there is an extra byte of information. fs may be subframes or status. Which one is indicated by bit 5 of the frames byte, so if you are displaying a location obtained by MMC, you have to mask the frames with an [& 31]. You can test bit 5 with [& 32].

If bit 5 of the frames byte is set, fs indicates status, broken down like this:

bit 6 set = location is estimated
bit 5 set = this code is invalid
bit 4 set = 1st frame of a 4 or 8 frame video sequence
bit 3 set = Time code never received

If bit 5 of the frames byte is 0, fs is 1/100ths of a frame. Darwin will cue up this accurately, but I'm not sure about ADAT or DA-88.

Bit 6 of the frames byte is the sign; (you may need negative time code to enter an offset, for instance.) Also, bit 6 of the seconds byte might be set- If so, it would mean there is no time code in this field. You could use it to see if General Purpose register 0 had a value, for instance. Bit 6 of the minutes byte can be set to indicate color frame - (contrary to a common misconception, this does not mean the timecode is at 29.97fps, it indicates the use of a particular color subcarrier synchronization system used on 1 inch videotape.) The upshot is to be really robust, you should [& 63] or [& 31] everything, but in current implementations it's not necessary.

To execute a locate, simply send the string with the desired location filled in (all 0s is handy).

Reading And Writing Information Fields.

So far we have dealt with open loop MMC operations. You send a command to the machine and it responds (or doesn't). More sophisticated control requires a closed loop with MIDI output from the deck returning to your application. Information can be returned as a result of an inquiry, automatically when things change, or steadily at a desired rate.

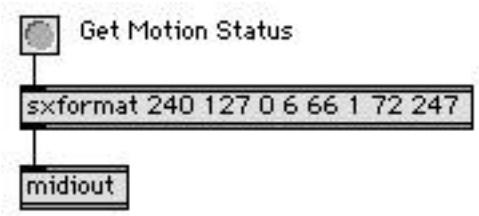
The MMC spec defines the kinds of information that may be available. The information is organized in "fields", which may include up to 60 bytes of information. In many cases, the information is packed into odd groups of bits within a byte, and you have to do some masking and shifting to extract the information. (If masking and shift operations are new to you, see the section on binary math.)

You can get the data in a field with the read command:

{240 127 ID 6 66 1 fn 247}

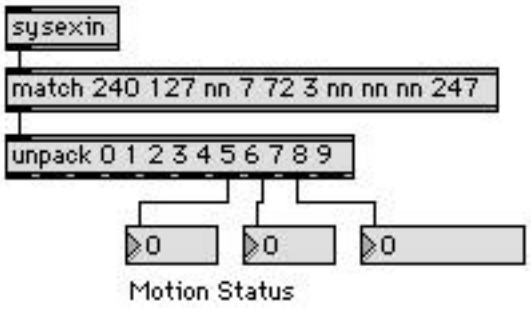
The 66 indicates a read command.
The 1 indicates there is 1 request in this message (You can string requests together- the total number goes in this spot) fn is the number of the field you want to read.

For instance [sxformat 270 127 0 6 66 1 72 247] will generate a message that tells if the deck is stopped, playing or in fast motion:



The message that comes back will look like this:

{240 127 ID 7 72 3 ms mp ss 247}



The header and ID are followed by 7 for response, the field you are getting (72 in this case), and the length (3) of the following data.

ms is the motion control state. These numbers are the same as the motion commands, except you won't get a 6 for recording. To find out if the deck is recording or not, you have to read another field.

mp is Motion Control Process. A process is a series of actions, such as rewind and play. Some defined codes are 11= chasing, 68 = locating, 127 = no process

ss is status and success level. In other words, if a deck takes a while to get up to speed, this tells if it's ready yet. Status for commands is in bits 0 - 2: 0 means "in progress", 1 means "finished" and 2 means it didn't work. Status for processes is in bits 4-7, with the same meanings

To find out about recording, ask for field 77. The response is

{240 127 ID 7 77 1 rs 247}

where rs has bits set to indicate various things. Decode by masking: bits 0-3 give
0 = not recording or rehearsing
1 = recording
4 = rehearsing
6 = record pause
bit 5 means record inhibited (a protected cassette, for instance)
bit 6 means rehearse inhibited
bit 7 means no tracks armed

Before I go any farther, I should point out that the response

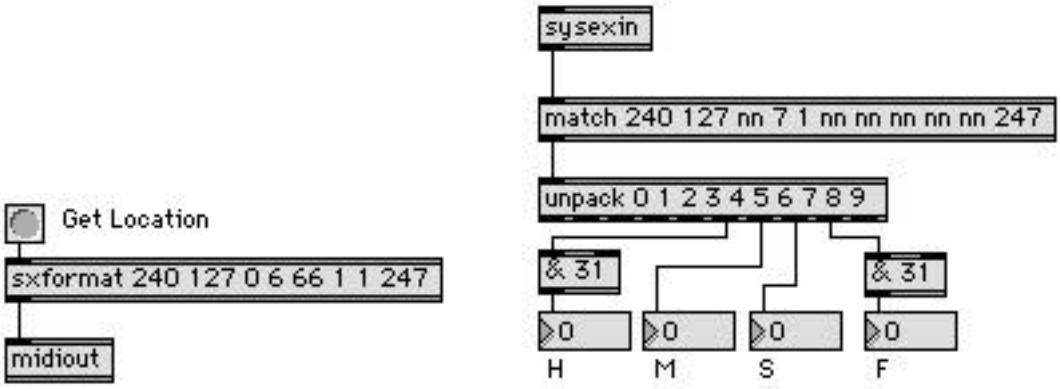
{240 127 ID 7 66 1 nn 247}

is always possible. It means field nn is not available for viewing.

There are about 70 fields defined in the MMC spec so I can't go through them all. One of the most interesting is field 1, which contains the current tape location. The response string will be

{240 127 ID 7 1 hr mn ss fr st 247}

with the hours and frames including time code format and status flags as I mentioned above.



If you are using the ADAT-Cooper system, you may be surprised by what you see. Right out of the box, the Cooper Datamaster reports location as if the 00 00 on the ADAT was 1:00:00:00. It is common practice to start time code just before the 1 hour mark, with the action beginning at the hour. This is because time code has to preroll a few seconds to allow machines time to lock up, and the time before 00:00:00:00 is 23:59:59:29. When decks see that, they zing into fast forward. You can set other offsets in the Datamaster for time code, but the MMC commands seem to keep the extra hour. However, if you locate to 1:00:00:00, it will run off the end of the tape.

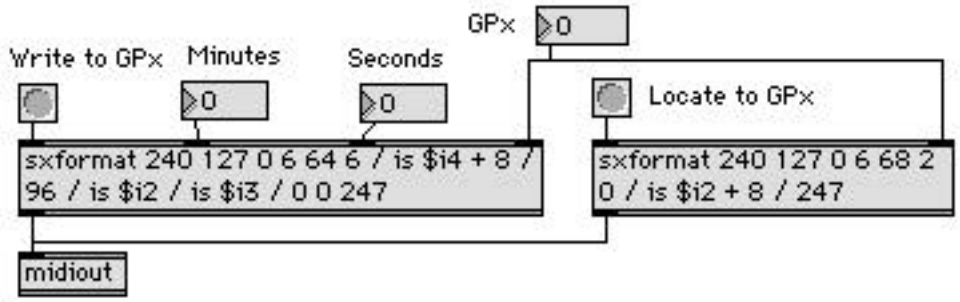
Writeable Fields

You can also write to some fields within an MMC device. Some of the most useful are 8 through 15, the general purpose location registers. These give you 8 locate points that are independent of Loc 1 or any other panel control. To use them, you first write the location wanted, then send a locate command.

To write to a GP register send the string

{240 127 ID 6 64 6 fn hr mn ss fr st 247}

where fn is the field number:



To go there, use the locate command with 0 as the subcommand:

{240 127 ID 6 68 2 0 fn 247 }

Arming Tracks

The current status of the tracks is kept in 2 fields: 78 = Track Record Status and 79 = Track Record Ready. The second is most useful because you can write to it to arm a track for recording. The field is organized as a "Track Bitmap", several bytes in which each track is represented by a single bit. If you query this field, the response will be

{240 127 ID 7 79 nn r0 r1 r2 ... 247}

nn is the number of data bytes following. It can vary a lot. If it is 0, no tracks are armed. The other bytes are the bitmap, which is set up like this:
r0
bit 0 video
bit 1 reserved (must be 0)
bit 2 time code track
bit 3 Aux track A
bit 4 Aux tack B
bit 5 Track 1 (finally!)
bit 6 Track 2

r1
bit 0 Track 3
bit 1 Track 4
bit 2 Track 5
bit 3 Track 6
bit 4 Track 7
bit 5 Track 8
bit 6 Track 9

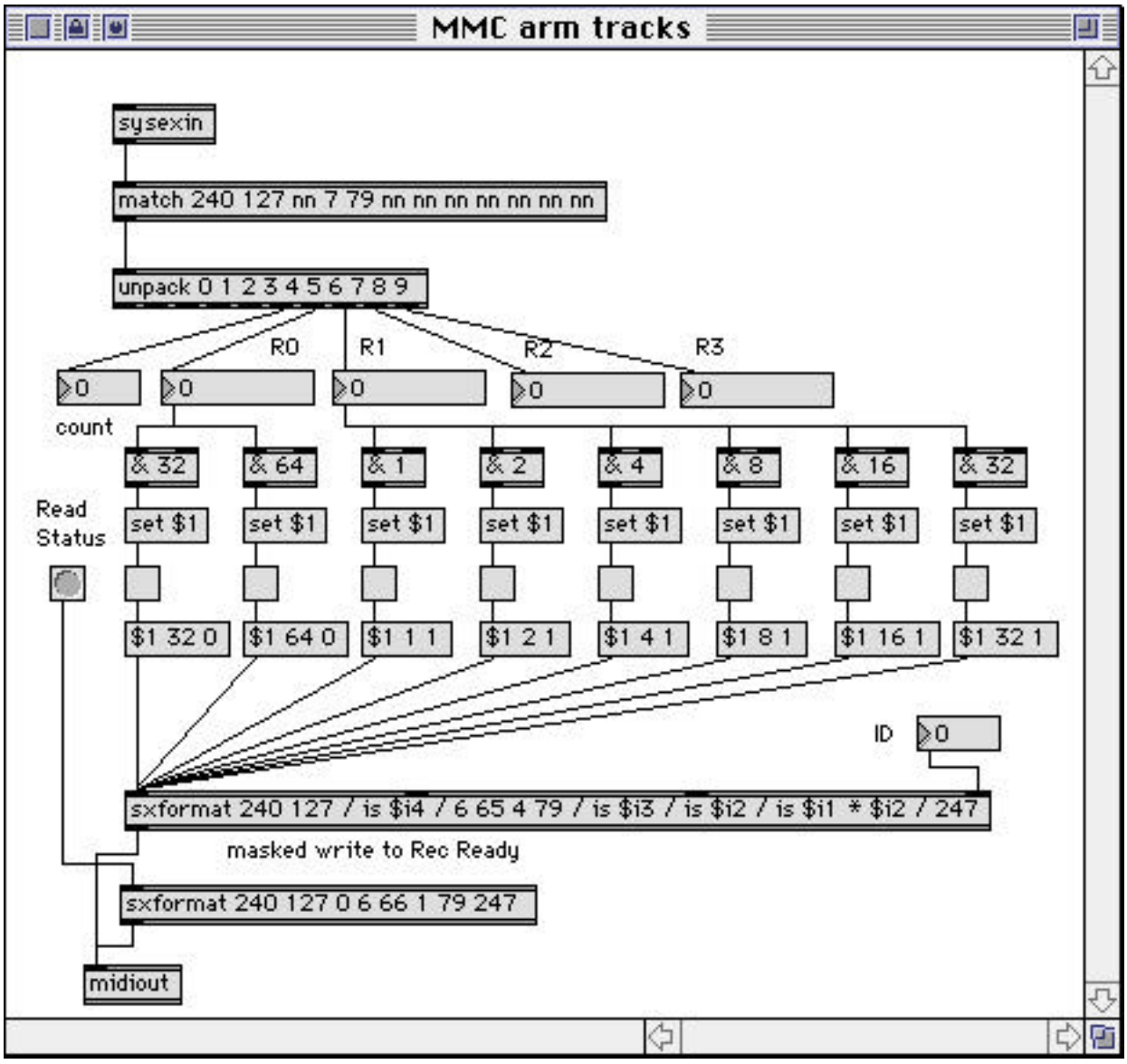
and so forth.

This not hard to decode using masking, but it is tedious. I suggest using Chris Muir's Binary object to pick out the bits.

To set tracks, we use the "Masked Write" command. It looks like this:

{ 240 127 ID 6 65 4 79 byte mask data 247}

4 is the length of following information; masked writes can be concatenated just like ordinary writes.
79 is the target field.
byte is the target byte within the field.
mask allows you to zero in on a bit. If a bit is zero in the mask, it won't be affected by the write.
data is the value to put in the uncovered bits. This is also bit by bit. The easy way to use it is to match the mask to turn things on and send 0 to turn things off, but it is perfectly reasonable turn some bits on and others off in the same operation.



This example shows how to read and set tracks, for the first 8 anyway. I've included a track 9, but it won't always work. It turns out that various combinations of gear work in different ways:

If you chain ADATs together, using Cooper Datamaster to get MMC, each ADAT assumes a different ID number, so if you want to arm the first track of the second machine (the 9th track) you send the command to arm track 1 on machine ID 1. If you request track status, the reply will only include the first machine. You cannot get track status from slaves.

If you start the chain with a Fostex RD-8, arming track nine will work. If you read track status, you will get 20 bytes of data, covering all 128 possible ADAT tracks. If a track on a slave machine was armed via MMC, its status will show on the bitmap, but not if it was armed with a panel button.

Darwins behave in a similar manner, but you only get the bytes you really need from a read. I don't have any DA-88s to test.

Updating

You can set some machines to notify you if things change. You start this off with the Update command. {240 127 ID 6 67 2 ss fn 247}
2 is a byte count- you can concatenate these too-
ss 0 means start, 1 means stop
fn is the field number you want to watch. Useful ones are 72 and 79 for motion status and 79 for track status. There is a field called Update Rate (65). Machines that support continuous updating will send a status message every so many frames.

Signature

Not all machines support all commands. You can find out what commands are supported on your machine by reading the "signature", field 64. You will get 50 bytes of data back, which is a huge bitmap of all possible commands and responses. A one in a bit means the command is available. For instance, I just looked at the RD-8 signature, and I see that bit 3 in byte 22 is zero, meaning it doesn't do updates. You really need the official spec to sort this out.