MIDI capabilities of the Euphonix CS series

Contents

1. MIDI-Ports and their purpose

There are several MIDI input and output ports to consider, some of them are quite obvious, as they are on the MOTU MIDI Express (PC or XT) which should be connected to the Euphonix PC, and another one not so obvious, as it is located underneath the Mix Controller as a 50-pin connector next to the Eugraphix port.

1.1. MOTU MIDI Express Port 1

This Port is used for bidirectional communication with the **ES108(a) dynamics modules**, where the out port of the MOTU MIDI Express would be connected to the first ES108's input, then (if more than one unit is present) daisy-chained through the following ES108 modules, until the output of the last unit is then connected to the input of the MOTU MIDI Express. The MIDI messages could be used to operate the ES108 units without a Euphonix console, but the implementation exceeds the scope of this guide. However, the MIDI implementation can be found on page 5-2 of the "ES108 Installation/Service Manual".

1.2. MOTU MIDI Express Port 2

On this port MixView is sending out the first 8 ASCII-Characters in HEX for each fader, 64 upper faders, 64 lower faders and 8 entries for the master section, just as it is listed in the "fader names" menu in MixVIew. It mirrors the display of the **"CleaR Displays"** modules that are (where) optionally available. If any fader is moved, then that fader's dB value gets is sent as text, and if it is released, then its name will be sent. The delay between release of fader and switching from dB value to name display can be adjusted from F2 (on computer keyboard), then F4 (on DSC), F4 (on DSC). MixView doesn't seem to react to incoming MIDI data on that port, but I may be mistaken…

1.3. MOTU MIDI Express Port 3

Here's where the communication of the **CC mappings and Program Change Messages** takes place. Everything you assign via F3 (System) -> F4 (MIDI) -> F1 (In) / F2 (Out) gets through that port. Note that the console ONLY sends MIDI from its Group Masters, so for any fader or mute button to actually send MIDI, it needs to become a Group Master (no slaves needed). The upside: no audio fader has to be sacrificed for a CC sending remote fader.

1.4. DSC Comm. Port (50-pin D-Type)

On Pins 23&39 (MIDI in), 40&7 (Out) and 5 (Ground) the DSC sends and receives **MMC (MIDI Machine Control)** commands. Those include transport controls, record arming of up to 48 separate tracks on a tape machine, and some others. MMC is usually to a basic degree natively understood by most DAWs. For more advanced features, such as Jog/Shuttle or record arming, one can translate the SysEx messages into “learnable” CCs or Note On/Off messages. More on that later…
Since I didn’t have a “DSC MIDI Breakout Cable” I had to solder my own according to following diagram in figure 1:

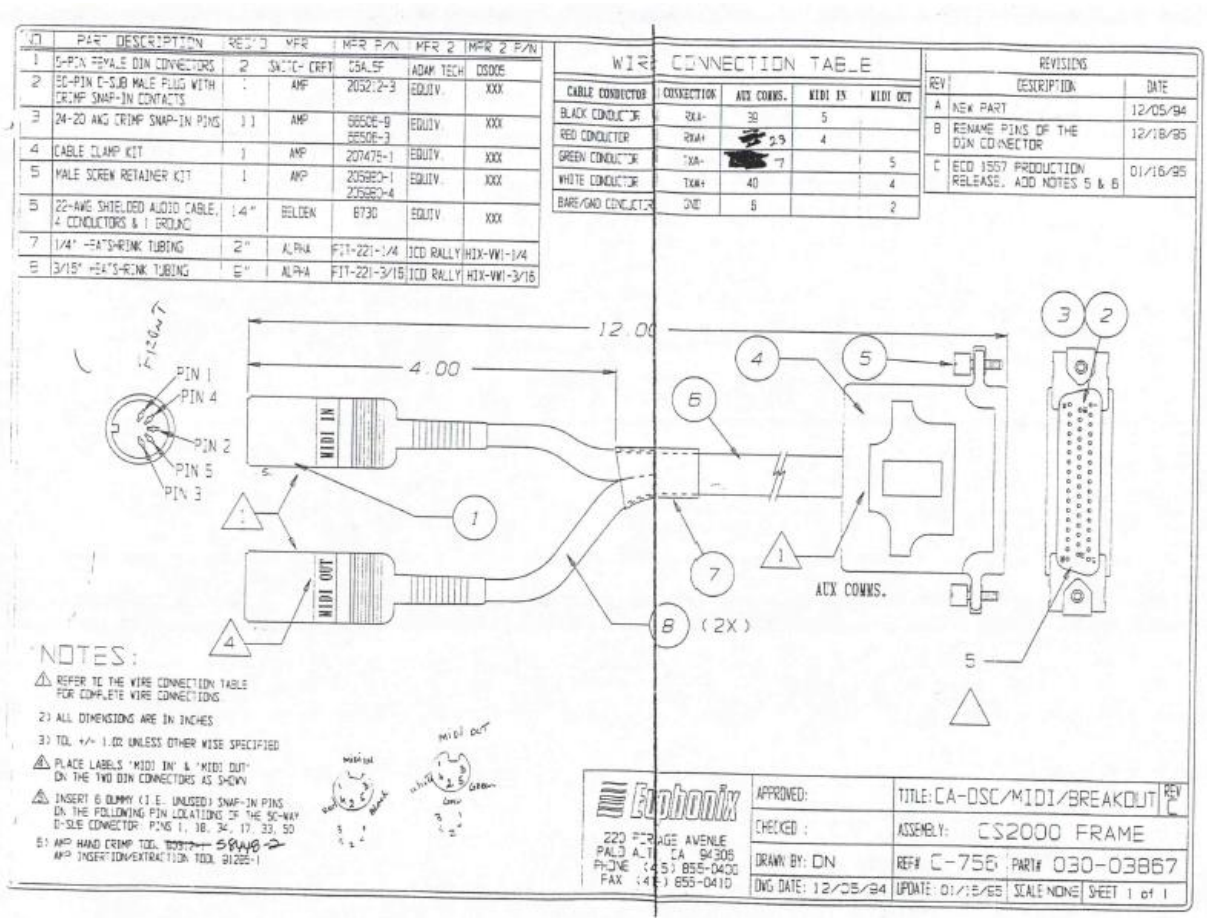

Figure 1: MIDI Breakout Cable

2. Data formats / message types

Now we take a look at how those commands are sent and how we can interpret and translate them into useful commands. MIDI messages (and bytes in general) are usually expressed in hexadecimal, which is a convenient way of reducing the number of digits of an 8-bit (binary) message. So a binary message of 1111 1111 would result in FF, 0111 1111 would become 7F and so on. Since the first bit is reserved for the start or end of messages, the highest value of a MIDI command cannot exceed 7F, which in decimal would be 127, which brings us to the typical 7-bit limit of 128 Notes (0 counts as value), 128 CC values, 128 whatever, if not used together with another MIDI message. If we use CC pairs (e.g. CC01/CC33) to target a controller, we get 14 bits resolution, which results in 16384 values, being more appropriate for sensitive things, such as volume faders on a console…

2.1. ES108 messages (MOTU Port 1)

Please refer to the "MIDI Implementation" pages in the "ES108 Installation/Service Manual", starting at page 5-1. By the time of writing I have no desire to hijack the data and reuse it for something different than what it already does best: controlling my ES108A

2.2. ASCII Characters for "CleaR Displays" (MOTU Port 2)

The messages sent from MixView look like this:

```
F0 2B 7F 14 11 40 20 20 20 48 65 6C 6C 6F F7
```

where F0 and F7 always mark the start and end of a SysEx-message,
2B 7F are usually the manufacturer IDs (so no other devices on that port accidentally react to that message), but in this case I'm not so sure, 14 is the command type for displaying text on the CleaR Displays, 11 targets channel eighteen (00 would be channel 1, 0F would be channel 16, etc…), 40 targets the lower fader (00 would target the upper fader), and finally:

```
20 20 20 48 65 6C 6C 6F
         H  e  l  l  o
```

are the 8 characters in ASCII-code to display, where 20 stands for empty space. If you named channel 18 Lower Fader "Hello", then that would be the SysEx message that's being sent on either loading the snapshot (along with all the other fader names), (re-)naming that fader, or releasing it after adjusting its level. During fader movement (either physical or automated) the dB value in 0.25 dB increments or decrements is sent as ASCII characters, which results in A LOT of consecutive messages.

By the time of writing I don't know yet how to get a program to directly translate the MIDI message into ASCII characters, but I'm working on it. The long-time goal would be to build one's own version of those "CleaR Displays", either as an overlay on a computer screen, or as hardware with an Arduino and 8x1 character LCDs…

I also don't know if MixView can receive anything useful on that port, like naming the channels or displaying fancy character movements on the CleaR Displays from an external source. That would however be very low on the agenda…

2.3. CC Mappings and Program Change (MOTU Port 3)

This is the part that seems to be the easiest to grasp, since it's the best documented one. Nevertheless, it took me a while to figure it out, and I'm trying to guide you through the process as well. On this port you can basically use your faders and mute buttons as remote controllers for anything imaginable that accepts either 7-bit or 14-bit messages. Missing a LARC for your reverb unit? Wanna control the cutoff frequency of that cool filter plugin? No problem, just use faders on your console for any parameter you like. Or maybe you want it the other way round? Exactly draw panning or EQ moves of your console from the grid of your DAW? Let's dive in…

2.3.1. Controlling Euphonix CS from external device or DAW

If you navigate to F3 (System) -> F4 (MIDI) -> F1 (In), you'll see a pre-populated list with all the possible parameters of the console that can be controlled via external MIDI-CC-commands, all targeting channel 1 of the console as a starting point, and all switched to "OFF".

Each parameter is assigned a unique CC number, starting at 1 for "lower fader", 2 for "lower mute", and so on. The CC numbers go from 1 to 31, and then from 64 to 91, with room from 92 to 127. The omission of CCs 32 to 63 serves the purpose of leaving the possibility to build CC pairs for 14-bit messages (CC0/32, CC1/33,…,CC31/63). Those pairs are kind of a standard when it comes to 14-bit messages. We therefore end up (if we take CC number zero into account) with 97 possible CCs per MIDI Channel, which gives us 97 x 16 = 1,552 possible unique objects on the console to be remotely controlled, if nothing else is on that MIDI port. But why would it? By now you should have realized, that a 10-Port USB-MIDI-Interface is no overkill…

It is advisable to leave this list as it is, and simply make a copy of the item you want to automate. With the wheel, just navigate to that object, hit ENTER on the DSC Keypad (your copy has automatically been turned "ON") and adjust values for console channel and incoming MIDI CC and MIDI channel by navigating with the two arrow keys on the DSC keypad. You can decrease / increase the values with F3 / F4, or you can hit F2 and then turn the wheel to adjust the value (don't forget to hit F2 again when your done). If you want to use higher resolution 14-bit mode, navigate to the field "MIDI Width" and hit F4 to change between "7bit" and "14bit". Just make sure your external MIDI device or your DAW is also sending 14-bit messages on the same CC pairs.

To remove an entry from that list, just navigate to that object and hit "Del" on the DSC keypad. If you hit "Clr", then you need to confirm to clear all your assignments. The list then reverts to the pre-populated one as you first encountered it.

2.3.2. Controlling external devices or your DAW (PlugIns) from Euphonix CS

From F3 (System) -> F4 (MIDI) -> F2 (Out) you get to the remarkably shorter list of possible objects on the console that could control your external devices or your DAW (or plugins therein). Only the faders and mute buttons can send MIDI CC data. To be exact, not the faders and mute buttons themselves, but the Group Masters they represent. If I wanted the lower fader of channel 21 to send CC data, then I first would navigate in that list to “L GRP MASTR”, hit “Enter” on the DSC keypad, navigate to my copy (MIDI Mode set to “OUT”, which means “enabled”), set “Console Chan” to 21 and leave “MIDI Chan” on “1” and “Control Num” on “99” for now.

Now I turn “Group Mode” ON (hit “Esc” on DSC keypad until I arrive at the top level, then F4 (Grps) -> F4 (IND to MSTR)) and hit the attention key of channel 21 lower fader to set it as Group Master.

Et voilá: It’s sending a smooth continuous stream of values from 0 to 127!

If we switch to “14bit” we are not given the full 16384 values though, because upon inspecting the data stream, we discover that the the LSB (least significant byte) of the CC pair only switches between 00 and 40, and the MSB (most significant byte) ranges from 01 to 7E. If we count that upwards, it looks like this:

```
01 00
01 40
02 00
02 40
03 00
03 40
…
7E 00
```

That gives us effectively 252 values, so basically “just” double the precision. Well, you gotta keep traffic low, I guess…

With the mute buttons it’s another story: The button’s “on” state sends 7F (127) and the button’s “off” state sends 7C (124). If we invert that button from within our list, then the “on” state sends 00 (00), and the “off” state sends 03 (03). As DAWs and other gear don’t usually interpret such small value discrepancies as on/off states (usually a CC is considered “on” when its value is anywhere from 64 to 127, and “off” from 63 down to 0, and notes are considered “on” with velocities from 1 to 127, leaving only velocity 0 as “note off” command), we hereby encounter our first necessity to use a MIDI translator as a middle man. Either your DAW has the capability to transform or translate an incoming MIDI message to your needs, or you need a dedicated program for that.

I am very pleased with “Bome MIDI Translator Pro” for that (from here on called BMTP), as it allows me to have several presets for several tasks, that I can also turn on and off via SysEx commands (or MIDI notes, or CC messages, or…).

2.3.3. Sending Program Change Messages (with snapshots)

Please refer to page 12-3 in the “Euphonix CS3000/2000 Operation Manual Version 3.0”.

MIDI Machine Control (MMC) (DSC Comm. Port)

To use the transport controls from the DSC to control your DAW, multitrack recorder or tape machine, you need to connect it to the “DSC Comm. Port” connector underneath the Mix Controller. You need a male 50-pin 3-row D-Sub plug on which you need to solder (or crimp, depending on type) two MIDI cables, as shown on figure 1 in the first chapter. Or maybe you already have such a breakout cable, good for you then…

The MMC commands follow a standard, which is described here:
https://en.wikipedia.org/wiki/MIDI_Machine_Control

I’ll try to elaborate anyway…

2.3.4. Basic transport commands

Here are our 12 transport keys:

1: go to start(?)	2: previous cue	3: next cue	4: jog / shuttle	5: loop	6: auto-punch
7: locate	8: rewind	9: forward	10: stop	11: play	12: record

My DAW of choice is REAPER, and it recognizes the following buttons:

7: locate `F0 7F 7F 06 44 06 01 hr mn sc fr 00 F7`
- enter a timecode, hit locate again -> REAPER jumps to that position

8: rewind `F0 7F 7F 06 05 F7`
- interpreted as “move backwards a bit, depending on zoom”

9: forward `F0 7F 7F 06 04 F7`
- interpreted as “move forward a bit, depending on zoom”

10: stop `F0 7F 7F 06 01 F7`

11: play `F0 7F 7F 06 07 F7` (record exit / punch out)
`F0 7F 7F 06 02 F7` (play)

12: record `F0 7F 7F 06 06 F7`

the other buttons work internally, so if your cue list is populated with different timecode entries, then buttons 2 and 3 will recall those timecodes and send them out as MIDI message, like this:

```
F0 7F 7F 06 44 06 01 01 02 03 04 00 F7
```

where `01 02 03 04` are the hours, minutes, seconds and frames, so we would jump to 01:02:03:04

2.3.5. Special keys

Above the keys “Mstr Ctrl” and “TC Slave” are 6 keys that also send unique MIDI messages:

Special Key 1: `F0 2B 7F 1F 00 F7`
Special Key 2: `F0 2B 7F 1F 01 F7`
Special Key 3: `F0 2B 7F 1F 02 F7`
Special Key 4: `F0 2B 7F 1F 03 F7`
Special Key 5: `F0 2B 7F 1F 04 F7`
Special Key 6: `F0 2B 7F 1F 05 F7`

I use those keys to switch between 6 different presets for the translated messages of the jog wheel and the “Talk” button, which (shared with the “ComMic” button) sends:

```
F0 2B 7F 08 04 00 00 F7
```

Attention: This message is sent upon press AND upon release of either button. To use these keys to send MIDI notes or CC data, a timer that prohibits a retrigger upon release (200ms works for me) is necessary. I included that in my BMTP presets.

2.3.6. Jog / Shuttle

Key number 4 (above the stop key) enables the big encoder wheel to function as a jog / shuttle dial.

If the console is in “virtual tape machine mode” (neither “Mstr Ctrl” nor “TC Slave” are lit), then only the jog function works (blinking button) and one can navigate frame by frame by turning the wheel. No MIDI is sent, as the transport is meant to control the internal “virtual tape machine”.

If the console is in “Mstr Ctrl” mode, then pressing button 4 will toggle between shuttle mode and jog mode, and the turning of the wheel transmits different MIDI messages, depending on status and shuttle speed.

2.3.6.1. Jog mode

Turning the jog wheel clockwise sends:
```
F0 7F 7F 06 48 01 01 F7
```

Turning the jog wheel counterclockwise sends:
```
F0 7F 7F 06 48 01 41 F7
```

You can translate both messages into MIDI notes or CCs and use it in REAPER to either move left / right by frame, subdivision, or by transients in selected media items, or to select next / previous items, whatever you can imagine. Make presets for each idea and assign different MIDI messages for each. Switch presets with special keys 1-6. Go crazy…

2.3.6.2. Shuttle mode

These are the 8 levels of shuttle speed in both directions:

```
F0 7F 7F 06 47 03 47 7F 00 F7 (furthest counterclockwise)
F0 7F 7F 06 47 03 45 00 00 F7
F0 7F 7F 06 47 03 43 00 00 F7
F0 7F 7F 06 47 03 42 00 00 F7
F0 7F 7F 06 47 03 41 00 00 F7
F0 7F 7F 06 47 03 40 7F 00 F7
F0 7F 7F 06 47 03 40 3F 00 F7
F0 7F 7F 06 47 03 40 1F 00 F7
F0 7F 7F 06 47 03 00 00 00 F7 (shuttle stop)
F0 7F 7F 06 47 03 00 1F 00 F7
F0 7F 7F 06 47 03 00 3F 00 F7
F0 7F 7F 06 47 03 00 7F 00 F7
F0 7F 7F 06 47 03 01 00 00 F7
F0 7F 7F 06 47 03 02 00 00 F7
F0 7F 7F 06 47 03 03 00 00 F7
F0 7F 7F 06 47 03 05 00 00 F7
F0 7F 7F 06 47 03 07 7F 00 F7 (furthest clockwise)
```

REAPER doesn't have any built in equivalent to shuttle, but you could build different translators for each SysEx message in BMTP, that trigger a CC or note in a loop, and decrease the delay timers between that loop points accordingly. So you can learn that CC or note in the actions menu to "move cursor right (or left) <one pixel | 10 pixels | one frame | whatever>". The shorter the delay, the faster the movement…

I already built a BMTP preset for that, and I will share my presets with anyone interested.

2.3.7. Track arming

To go into track arming mode on the DSC, you first need to enable “Mstr Ctrl”. Then, the rightmost one of the 4 small “Rec” buttons (the only one that stays lit after being pressed) ABOVE the “Auto Rec” button enables you to use the 48 keys above to individually arm up to 48 different tracks. REAPER doesn’t recognize those messages, so we need to extract the info inside it, to trigger single note or CC messages.

Here’s the catch though: The state of all tracks combined is transmitted in a single SysEx message each time the state changes (e.g. one track gets armed or disarmed).

Example:

```
F0 7F 7F 06 40 0A 4F 08  60  09  00  0D  2A  04  00  02  F7
                         1+2  3-9 10-16 17-23 24-30 31-37 38-44 45-48
```

states that the following tracks are armed: 1, 2, 3, 6, 17, 19, 20, 25, 27, 29, 33 & 46.

At first glance this looks quite cryptic, but it follows a rather simple logic, once you look at it from a different perspective. First off, the 8 bytes (the grey ones) that contain the values are holding a different group of up to seven tracks each. The first byte describes the state of tracks 1 & 2, the second byte tracks 3 - 9, and so on, until the remaining tracks 45 - 48 occupy the second nibble (half a byte) of the last byte. The odd one is the first byte, as it only holds two instead of seven tracks, and those two are even shifted 5 bits to the left. But we can handle that…

To visually grasp the state of the tracks, it is better to express those hexadecimal values in binary, which gives us the following (note that the first bit always has to be 0):

60	=	**0 1 1 0**	**0 0 0 0**	tracks 1 & 2
tracks no.		x 2 1 x	x x x x	
09	=	**0 0 0 0**	**1 0 0 1**	tracks 3 & 6
tracks no.		x 9 8 7	6 5 4 3	
00	=	**0 0 0 0**	**0 0 0 0**	no tracks
tracks no.		x 16 15 14	13 12 11 10	
0D	=	**0 0 0 0**	**1 1 0 1**	tracks 17,19,20
tracks no.		x 23 22 21	20 19 18 17	
2A	=	**0 0 1 0**	**1 0 1 0**	tracks 25,27,29
tracks no.		x 30 29 28	27 26 25 24	
04	=	**0 0 0 0**	**0 1 0 0**	track 33
tracks no.		x 37 36 35	34 33 32 31	
00	=	**0 0 0 0**	**0 0 0 0**	no tracks
tracks no.		x 44 43 42	41 40 39 38	
02	=	**0 0 0 0**	**0 0 1 0**	track 46
tracks no.		x x x x	48 47 46 45	

3. Practical Implementation

Here's where the fun (or the pain, depending on nerd-level) begins!

The following are examples of the sheer endless possibilities a MIDI-spitting console allows us to explore. I'll demonstrate the procedures in REAPER (because EVERY command, basic or compound, can "learn" a MIDI input) and Bome MIDI Translator Pro (BMTP), for which I offer my custom programmed presets. Make sure to have the latest SWS Extensions installed for REAPER!

3.1. Bome MIDI Translator Pro

I use BMTP as a middle man between the my MIDI ports that are connected to the CS2000's MIDI ports and my DAW. In BMTP I select the MIDI inputs, and route them to the "Bome MIDI Translator Virtual Out". In REAPER, I select "Enable input for control messages", but leave "enable input" unchecked. I don't want to use the CS2000 as a track input…

In BMTP you can create different presets, than can either be persistent or switchable. I created 6 different presets that I switch with the 6 "special keys" on the DSC. They, of course, cancel each other out. Alongside I have several presets that I want to be active permanently, such as my translators for the record arm keys, or the preset that allows me to switch the other 6 presets in the first place.

I made a an entire BMTP Project containing all the presets I made for the console, which I'm gonna share on the Euphonix User Group Facebook page. Since it refers to my MIDI setup, it would be best if you copy all the presets from my file, create a new one with your MIDI routings, and then paste the presets into your new project.

Inside those presets are my translators. There can be as many "translators" as you want. A translator basically reacts to a given message (Keystroke, MIDI, timers from other translators or even RS232 messages) and either transforms it to something different, swallows it (the message doesn't reach the receiver), or evaluates it and sets variables accordingly and performs arithmetic operations with them. For our purposes, we'll need those variables a lot…

Please refer to the manual of BMTP for further guidance. Or just use my presets, you lazy ba…

3.2. Track Arming

I made a preset in BMTP that scans the incoming SysEx message on each track-specific bit and assigns its value to its own global variable. Depending on the state of that variable, the 48 MIDI messages that are sent contain either a unique Note with velocity 127 on MIDI channel 2 for "on" or on MIDI channel 3 for "off". I didn't want to pollute channel 1 with those messages, because they would mess up my other mappings.

Since REAPER doesn't have a command to specifically arm or disarm tracks, only to toggle between the two states (which inevitably will result in mismatches between lit keys on the DSC and actually armed tracks), I searched for a script that first checks the state of the track before toggling it:

```
track_num = 1 -- 01 to 99
rec_onoff = 1 -- 1 for arm, 0 for unarm
rec_toggle_comm_id = 8*track_num +17 -- calculate CommandID num for
track
toggle_state = reaper.GetToggleCommandStateEx(0, rec_toggle_comm_id) --
get toggle state for track
if toggle_state ~= rec_onoff then
reaper.Main_OnCommandEx(rec_toggle_comm_id, 0, 0) end -- if
toggle_state isn't desired rec_onoff state, toggle
-- reaper.Main_OnCommandEx(integer command, integer flag, ReaProject
proj)//
```

At the time of writing I haven't figured out yet how to elegantly get everything into one script, so here's the dirty version:

- In REAPER's action menu, hit "New action…" -> "New ReaScript…". Choose a location (best to create a folder called "Euphonix Record Arm" or similar) and save as "RecArm Track 01 ON.lua" or similar
- Paste the above code into the editor and save (Ctrl + S)
- In BMTP **deactivate all translators** named "out x" except the one you're setting up to bind (e.g. out 1). We don't want to send all notes at once when try to bind one MIDI note to a REAPER action. You can select all and then Ctrl-click the checkbox to disable all selected at once.
- On the DSC, make sure the small record button is lit and track 1 is **not** set to record arm
- In REAPER's action menu, select your newly created action and hit "Add…" in the "Shortcuts for selected action" field. A MIDI note message should be recognized.
- Repeat for "RecArm Track 02 OFF.lua" and adjust pasted code accordingly (rec_onoff = 0)
- Repeat for Tracks 2 – 48, adjust pasted code accordingly (track_num = 2/3/…/48)
- Enable all translators in preset in BMTP

3.3. Jog Dial

I'm using the jog dial on the DSC for various purposes:

- **Preset 1**: jog dial navigates through transients of selected media item / Talk button moves nearest grid line to cursor (very handy if you want to map your grid lines to your drummer)
- **Preset 2**: jog dial navigates through transients of selected media item / Talk button inserts stretch marker in selected item on cursor position (very handy, if you want to lock your bass player to the drummers grid)
- **Preset 3**: jog dial navigates through single frames / Talk button splits selected item(s) at cursor
- **Preset 4**: jog dial navigates through grid subdivions / Talk button splits selected item(s) at cursor
- **Preset 5**: jog dial navigates through items on selected track* / Talk button deletes selected item*
- **Preset 6**: empty (go crazy!)

* I need to figure out how to make it select multiple items…

In order to make this work, the Preset “[0] Switch Presets” must be activated, and the Presets [1] to [6] must be present. They are not set to “always active”, so they switch on preset change. The retrigger filter (200ms) is already implemented, but you can tweak it to taste by adjusting the “Outgoing” action of the “retrigger cancel” translator. You’ll have to adjust it in each preset though…

The whole preset occupies MIDI notes 00 – 11 (decimal) on MIDI channel 1. Those can be learned in REAPER and assigned to actions.

3.4. Shuttle

This preset sends continuous note messages on MIDI channel 4. Note number 00 gets sent on forward position and note number 01 on backwards position. The speed of that loop increases with shuttle position and stops completely when middle position is reached. You can assign those notes in REAPER to anything from next / previous frame, ms, pixel, subdivion, depending on your taste. Just search the acion menu for “move cursor next” (or previous) or “move cursor right” (or left), and you’ll get plenty of options.

Well, that should suffice for now. I hope I could help lifting the curtain a bit. I’ll be tinkering with those things myself from here on, and will update this document with new findings or more elegant ways to solve things (especially track arming). I’m sure you’ll come up with many creative ways to enhance your workflow with that mighty arsenal an possibilities.