MBT: Midi-Backup-Translator

Introduction

This little tool allow you:

- Backup and restore the settings of all your synths in one shot
- To convert any CC to SYSEX in realtime.

Special feature:

It is able to limit the bandwidth, so, no more MIDI buffer errors on synths like the **Yamaha TX81z**. If messages are coming too fast, they are simply dropped. Nevertheless, the last one is retained.

How to run

Windows

```
midi-translator.exe <command>
```

OSX

```
midi-translator <command>
```

JVM

```
java -jar midi-translator.jar <command>
```

How to configure

You have to define a `config.yml` in you current directory, typically the project folder your are currently working on with your DAW.

- We provide a `config-example.yml` as an example that you have to rename.
- Your configuration, specific to your current musical project, can use macros defined in the **device library**

You need to be comfortable with YAML syntax and Hexadecimal notation of numbers.

Usage

```
AVAILABLE COMMANDS

Built-In Commands
       help: Display help about available commands
       history: Display or save the history of previously run commands
       version: Show version info
       script: Read and execute commands from a file.

Midi Translator Shell
       backup: Backup devices with Sysex
       restore: Restore devices with Sysex
       list: List MIDI devices
       translate: Read MIDI input and send to another MIDI output limiting the
throughput
```

List devices

The first thing to do is to list the MIDI devices of your system:

```
>midi-translator.exe list
INPUT Device "FromDAW"
INPUT Device "MidiClock"
INPUT Device "ToDAW"
INPUT Device "loopMIDI Port"
INPUT Device "taurus"
OUTPUT Device "FromDAW"
OUTPUT Device "Microsoft GS Wavetable Synth"
OUTPUT Device "Microsoft MIDI Mapper"
OUTPUT Device "MidiClock"
OUTPUT Device "ToDAW"
OUTPUT Device "loopMIDI Port"
OUTPUT Device "taurus"
```

If a device using those ports is defined in your library you will get something like this:

```
MIDI INPUT Port "4- MIDISPORT Uno In" => bound to library device "DS-330"
MIDI INPUT Port "FromDAW"
MIDI INPUT Port "MidiClock"
MIDI INPUT Port "ToDAW"
MIDI INPUT Port "loopMIDI Port"
MIDI INPUT Port "taurus"
MIDI OUTPUT Port "4- MIDISPORT Uno Out" => bound to library device "DS-330"
MIDI OUTPUT Port "FromDAW"
MIDI OUTPUT Port "Microsoft GS Wavetable Synth"
MIDI OUTPUT Port "Microsoft MIDI Mapper"
MIDI OUTPUT Port "MidiClock"
MIDI OUTPUT Port "ToDAW"
MIDI OUTPUT Port "loopMIDI Port"
MIDI OUTPUT Port "taurus"
```

Backup

```
>midi-translator.exe backup
```

You can backup a specific device specifying a known macro:

```
>midi-translator.exe backup -d "DS-330" -m "Everything()"
```

The backup filename will be `BulkDump DS-330.syx`

Restore

```
>midi-translator.exe restore
```

You can backup a specific device specifying a known macro:

```
>midi-translator.exe restore -d "DS-330"
```

The backup filename has to be `BulkDump DS-330.syx`

The device Library

It is located in the distribution of the tool, in the folder `devices`

- It contains common settings and MIDI SYSEX for various synths in a file `<device-name>.yml`
- It also contains YOUR custom settings in `<device-name>-user.yml` file
- In this way you don't have to write the same settings on each project in the `config.yml`
- There is no constrains on the organization of sub folders, the tool scan everything.
- The file `<device-name>-user.yml` is loaded **after** the file `<device-name>.yml` in order to override settings of your choice. Typically MIDI IN/OUT ports
- On each release of the tool you can unzip it without taking the risk to loose your custom settings in `*-user.yml` files.

Example

Here a device file `Tr-rack.yml` for the Korg TR-Rack:

```
deviceName: "Tr-rack"
brand: "Korg"
outputMidiDevice: "M8U eX 11"
inputMidiDevice: "M8U eX 12"
macros:
  - "GLOBAL DATA()                    : 00543 : F042303B 0E 00 F7"
  - "MODE DATA()                      : 0000B : F042303B 12 F7"
  - "MULTI DATA()                     : 011E9 : F042303B 18 00 F7"
  - "CURRENT PROGRAM PARAMETER()      : 001F6 : F042303B 10 00 F7"
  - "CURRENT COMBINATION PARAMETER()  : 001C3 : F042303B 19 00 F7"
  - "ALL DRUMKITS DATA()              : 098D3 : F042303B 0D 00 00 F7"
  - "ALL PROGRAM PARAMETER()          : 3DDC1 : F042303B 1C 00 00 00 F7"
  - "ALL COMBINATIONS PARAMETER()     : 376E6 : F042303B 1D 00 00 00 F7"
  - "ALL DATA()                       : 80481 : F042303B 0F 00 F7"
```

On your system you can override the MIDI IN/OUT devices, creating the file `Tr-rack-user.yml`

```
deviceName: "Tr-rack"
brand: "Korg"
outputMidiDevice: "midi-port-2"
inputMidiDevice: "midi-port-3"
```

The following fields are provided:

Field	Description
`deviceName`	Name of the device that you will use in the `config.yml`
`brand`	Informative but not used
`outputMidiDevice`	MIDI port to send Bulk Requests
`inputMidiDevice`	MIDI port to receive Bulk data
`outputBandwidth`	Maximum bytes per sec used during translation: if you send too many CC, some of them will be dropped to prevent any "buffer overflow" on the device. Use that only for old devices.
`sysExPauseMs`	Pause between each MIDI requests during the backup and restore process. Default is 0.
`inactivityTimeoutMs`	When the response size is not specified, we use this parameter to know if something is wrong.
`macros`	A list of strings containing various macro definitions to build MIDI requests

Nibbles

You can generate half bytes, they will be merged to make a byte

```
- "User Perf Common(perf) : 39 : F043204B 70 perf 0 0 F7" #  "0 0" will becomes "00"
```

Macro definition

The overall definition is:

```
- "macro name(parameters) : reponse size in hexadecimal : payload in hexadecimal"
```

parameters: a list of names separated by `,` that can be used inside the payload. If there is no parameters `()` is still required.

response size: If you don't know the response size, just use a non-numerical string like `"-"` or `"---"`. Knowing the response size make faster backups. What you can do is first do a backup without, then observe the log to get the response size and put it in your macro for alter use.

payload: The payload is just a bunch of bytes in hexadecimal. You can pack them to make it more readable or separate them by spaces. `F042303B 1C 00 00 00 F7` is exactly the same than `F042303B1C000000F7`

Macro Call

Calling a macro looks like this:

```
macro name(parameters values)
```

parameters values: a list of values separated by `,`

Values format can be:

- Decimal: `64`
- Hexadecimal: `$12` or `0x12`.
  - You can force a n byte value with `0` like this: `$0012` or `0x0012`
  - You can force a nibble value (half byte) with: `$F` or `0xF`
- Decimal range: `[0-127]`
- Hexadecimal range of nibbles: `[$0-$F]` or `[0x0-0xF]`
- Hexadecimal range of bytes: `[$00-$7F]` or `[0x00-0x7F]`
- Hexadecimal range of word: `[$0000-$7F00]` or `[0x0000-0x7F00]`
- ...

So `Multi([0-15])` mean we call the macro 16 times with value 0 to 15. This will generate multiple MIDI Requests for various memory locations.

Recursion

You can call a macro inside a payload macro:

```
- "User Perf Common(perf) : 39 : F043204B 70 perf 00 F7"
- "User Perfs Common() : User Perf Common([00-127])"
```

This mean using `User Perfs Common()` in your `config.yml` will generate 128 MIDI requests for various memory locations.

Limitation

Only one range can be used in a payload.

Project Configuration

Example

Let say you work on a project using a `Yamaha TX-81z` and a `Yamaha TG-500`.

- Create a folder `/SysEx` at the root of your project folder. MIDI Dumps will be saved here.
- You will create a file `/SysEx/config.yml`:

```
devices:
  - name: "TX-81z"
    dumpRequests:
      - "VMEM()"
      - "PMEM()"
      - "SCED()"
      - "PCED()"
      - "System Data()"
      - "Program Change Table()"
      - "Effect Data()"
      - "Micro Tune Octave()"
      - "Micro Tune Full Keyboard()"

  - name: "TG-500"
    dumpRequests:
      - "System Setup()"
      - "AllMulti()"
      - "AllVoiceInternal1()"
      - "AllVoiceInternal2()"
      - "AllVoiceEditBuffer()"
      - "AllPerformances()"
translate:
  fromMidiDevice: "4- MIDISPORT Uno In"
  toDevice: "Tx81z"
# Translation rules:
# use vv to inject the control change value inside the SYSEX
# use [low,high] to rescale the input value
translations:
  - 7 => F0 43 12 12 32 vv F7 [-6,10] # convert CC Volume (7) in the range [0-
127] to some TX81z parameters in the range [-6,10]
```

You can now restore and backup your devices running the tool in the directory `/SysEx`.

Fields

Field	Description

Field	Description
`devices/name`	Name of the device that you will use. It should match one of the devices in the library
`devices/enabled`	Can be used if you don't want to backup/restore the device. Default is `true` of course.
`devices/outputMidiDevice`	Override the field declared in the library
`devices/inputMidiDevice`	Override the field declared in the library
`devices/outputBandwidth`	Override the field declared in the library
`devices/sysExPauseMs`	Override the field declared in the library
`devices/inactivityTimeoutMs`	Override the field declared in the library
`devices/dumpRequests`	A list of strings containing various macro calls or SYSEX payloads
`translate/fromMidiDevice`	Name of a input midi port
`translate/toDevice`	Name of a device. It should match one of the devices in the library
`translations`	A list of strings containing various translation rules

Translation

The syntax looks like this:

```
"CC" => "payload" "output range"
7 => F0 43 12 12 32 vv F7 [-6,10]
```

This convert CC Volume (7) in the range [0-127] to some TX81z parameters in the range [-6,10]

⚠ Pay attention on the representation of numbers:

- The CC on the left is in decimal
- The payload is in hexadecimal
- The target range is in decimal

mbt-config

You can change the default path of the `config.yml` with `-Dmbt-config=/somewhere/config.yml`