

THE TINYBOX

User Manual v.1.7

Contents

Introduction: what is the TinyBox ?	4
1. An FCB1010 “booster”	4
2. A MIDI-USB interface.....	5
3. A MIDI filter/router	5
4. A wireless status display on iPad or tablet.....	5
5. A MIDI touch controller	5
6. A setlist manager, lyrics display, sheet music app	5
The TinyBox hardware	6
Getting started	8
1. Make sure the FCB1010 can connect to the TinyBox.....	8
2. Turn the FCB1010 into a TinyBox slave	8
3. Install TinyBox ControlCenter.....	8
4. Make all connections.....	8
5. Launch TinyBox ControlCenter	8
Using a computer with the TinyBox	9
The TinyBox ControlCenter software	10
Launching TinyBox ControlCenter	10
Displaying TinyBox status info on the laptop	13
Using the TinyBox editor	14
Managing setlists, song lyrics and music scores.....	15
Monitoring the TinyBox MIDI messages	19
Upgrading the TinyBox firmware	20
Programming the TinyBox	21
The TinyBox setup structure	22
Example 1 : structure of a typical TinyBox setup	23
Example 2 : sending MIDI messages	25
Example 3 : programming expression pedals.....	26
Example 4 : using variables	27
Example 5 : using conditional commands	28
Example 6 : a programmable MIDI filter	29
Example 7 : a programmable MIDI router.....	30
The TinyBox programming language	31
0. Comments	31
1. Defining preset, effect, trigger and sweep names	32

2.	Defining the bank structure	35
3.	Defining songs and setlist.....	39
4.	Defining preset contents	40
4.1.	Defining MIDI channels	41
4.2.	Defining data variables.....	42
4.3.	Defining the TinyBox initial state	43
4.4.	Defining song or bank initialization.....	43
4.5.	Defining the preset contents.....	44
4.6.	Defining the effect contents.....	44
4.7.	Defining the trigger contents	45
4.8.	Defining the sweep contents.....	46
5.	The command set.....	47
5.1.	Switch and pedal assignment commands	48
5.2.	Effect activation, relay activation and expressionpedal activation commands.....	50
5.3.	Navigation commands.....	51
5.4.	MIDI commands	52
5.5.	Continuous Control commands.....	53
5.6.	Delay command.....	54
5.7.	Filter/router commands	54
5.7.1.	Filtering certain MIDI message types	55
5.7.2.	Filtering certain MIDI channels	55
5.7.3.	Routing MIDI channels	56
5.7.4.	Filtering or routing MIDI notes.....	56
5.7.5.	Filtering or routing MIDI ControlChange messages	57
5.7.6.	Resetting the filter/router	57
5.8.	Variable commands.....	58
5.9.	Conditional commands.....	59
5.9.1.	The condition syntax	60
5.9.2.	if...then...else statements	61
5.9.3.	while statement.....	62
5.9.4.	switch statements	62
5.9.5.	conditions using predefined variables	63
5.10.	Remote Control	65
	APPENDIX : TinyBox programming language reference.....	67
	APPENDIX : the TinyBox MIDI routings – a detailed rundown.....	70
	The TinyBox as a MIDI-USB interface	71

The TinyBox as a MIDI mapper/filter	72
The TinyBox as advanced MIDI controller with (virtual) FCB1010 floorboard.....	73
Remotely controlling the TinyBox	75
Another remote control option : “UseKeyboardControl”	76
A special case : troubleshooting.....	77

Document versions

Version 1.7	01/01/2022	New functionality added in firmware v.1.7
Version 1.5	15/11/2020	Remark added about using the DOWN switch for NEXT song (p.18)
Version 1.4	20/08/2020	Support for extra SysCommon and SysRealtime messages added SwitchOn/SwitchOff commands no longer restricted to be used in presets only – effect activation infinite loop detection added
Version 1.3	01/07/2020	support for ‘while’ loops added
Version 1.2	05/06/2020	Appendix about latency measurements added
Version 1.1	01/06/2020	Getting started chapter added MIDI routing appendix added ‘UseKeyboardControl’ and ‘ModifyVelocity’ commands added
Version 1.0	18/05/2020	Initial release

Introduction: what is the TinyBox ?

As an ideal companion to the Behringer FCB1010, the TinyBox is many things in a tiny black box. It will take a comprehensive manual to explain all features in detail, but here's a short overview:

1. An FCB1010 "booster"

By combining your FCB1010 with the TinyBox, you turn it into a very powerful MIDI foot controller. The box eliminates all the restrictions of the original floorboard:

- a setup can contain up to 200 banks of 10 presets
- each bank can contain any mix of presets, effects (or stomp boxes) and triggers (or momentary effects)
- each preset can send a virtually unlimited number of MIDI commands on any combination of MIDI channels
- a preset can send SysEx messages of any complexity
- a preset can start a MIDIClock stream with programmable BPM, and send MIDIStart, MIDIStop or MIDIContinue messages
- a programmable delay can be added between MIDI messages
- an optional "Direct Bank" switch can give one-click access from all banks to an extra set of 9 commonly used presets
- each expression pedal can send one or multiple ControlChange sweeps, which can be different for each preset.
- expression pedals can use different sweep curves
- expression pedals can also send PitchBend or ChannelPressure messages
- the use of conditional commands and data variables allows to add complex "if...then...else..." logic to your setup content
- ...

All this comes with very intuitive editor software which makes creating a complex setup as easy as can be. You can describe your setup as plain text, using a text editor with auto-complete functionality. This makes sharing your setup with others as easy as copy-and-pasting a piece of text. The setup text is compiled into binary code prior to sending it to the TinyBox through USB. The software runs on Mac and Windows.

2. A MIDI-USB interface

The TinyBox includes a class-compliant MIDI-USB interface, which means it can be connected to a Windows or Mac computer without installing any drivers. The box can send MIDI messages both to hardware connected to its MIDI OUT port and to software running on a computer connected through USB.

The MIDI-USB interface is also used for reliable transfer of setups to the TinyBox. Gone are the days of buying and trying different MIDI interfaces to find one which is compatible with the Behringer FCB1010 patch dumps.

3. A MIDI filter/router

MIDI can be sent to the TinyBox and passed through a programmable filter/router. This filter can block certain MIDI channels, move or copy certain MIDI messages to a different MIDI channel, transpose MIDI Note messages up or down, adapt note velocity, block certain note ranges or ControlChange numbers, and so on. This allows you for instance to turn a simple MIDI keyboard into a full fledged master keyboard with different zones controlling multiple synths simultaneously.

4. A wireless status display on iPad or tablet

Connect the TinyBox to a computer running the included TinyBox ControlCenter, then wirelessly connect an iPad, Windows or Android tablet to the computer, and you get a wireless FCB1010 status display! TinyBox ControlCenter acts as a webserver for your tablet. Use the browser to view real time status of your FCB1010: current bank name, available preset names, stomp box states, and so on. What a huge difference compared to the small 7-segment display of the FCB1010 – you can now name your presets and stomp boxes and know what is going on at any time. ControlCenter itself also incorporates a local status display, so without WIFI you can view the status directly on your laptop too.

5. A MIDI touch controller

The same status display on your iPad can be used as a wireless MIDI controller! You can click any button on the “virtual” FCB1010 in your browser, and the TinyBox will react as if the footswitch of a “real” FCB1010 was pressed. This might appeal to keyboard players for instance, who could even leave the FCB1010 at home and use the virtual iPad version instead!

6. A setlist manager, lyrics display, sheet music app

As a fancy extra, TinyBox ControlCenter comes with a setlist manager which allows you to upload song texts and/or sheet music. You can scroll through the setlist using the FCB1010 up/down switches, and the preset bank for each song is activated on the FCB1010 while the corresponding song text or score is displayed on your laptop or iPad.

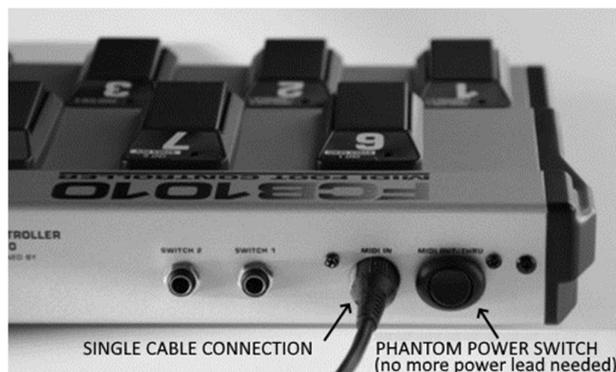
The TinyBox hardware



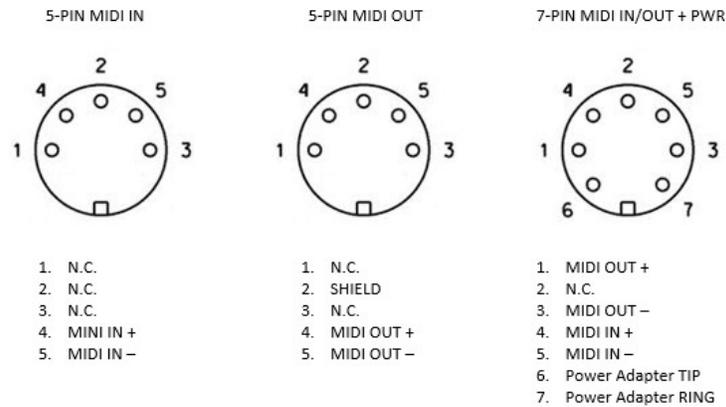
1. Power input: connect any power adapter delivering 9V AC or DC, 500mA or more
2. 7-pins MIDI connector for two-way connection with the Behringer FCB1010 ¹
3. USB connector for connection with a Windows or Mac computer
4. 5-pins MIDI IN connector
5. 5-pins MIDI OUT connector
6. Red power LED
7. Blue USB status LED

Along with the TinyBox comes a dedicated FCB1010 firmware chip, which turns the FCB1010 into a “TinyBox slave”. Programming the FCB1010 is no longer needed, all configuration is done in the TinyBox.

¹ The 7-pins cable transfers MIDI to and from the FCB1010, along with FCB1010 phantom power. For plug-and-play compatibility the FCB1010 needs to be equipped with a “Single Cable Kit” (available at <http://shop.tinybox.rocks>) which makes the FCB1010 phantom powered and gives it the same 7-pins MIDI connector for connection with the TinyBox.



The illustration below compares the 7-pins connector with regular MIDI IN and MIDI OUT connectors:



Although the TinyBox is designed as an add-on for the Behringer FCB1010, it is also possible to use all its features even without an FCB1010 connected:

- as mentioned in the introduction an iPad can serve as a “virtual FCB1010” : when clicking the buttons of the virtual FCB1010 the TinyBox will react in exactly the same way as if a real FCB1010 was used
- MIDI keyboard players can choose to use the lowest octave of their keyboard as TinyBox remote control instead of the FCB1010. 12 keys could replace the 12 footswitches of the FCB1010, and 2 joysticks or knobs could replace the expression pedals. Of course it is also possible to work with smaller preset banks in your setup and use less keys for remote control (3 keys could already be sufficient: “bank up” / “bank down” / “select preset”)

In order to use the TinyBox in this way, search for the “UseKeyboardControl” command further on in this manual.

Getting started

These are the first steps to take when you receive your TinyBox

1. Make sure the FCB1010 can connect to the TinyBox

As shown in the previous chapter the TinyBox communicates with the FCB1010 through a 7-pins MIDI connector. The easiest way to make the FCB1010 compatible with this connector is by installing the single cable kit (http://shop.tinybox.rocks/index.php?route=product/product&product_id=50)

Follow the instructions in the manual included with the kit.

2. Turn the FCB1010 into a TinyBox slave

Your TinyBox order includes a firmware chip for the FCB1010, which turns it into a TinyBox slave. With this firmware it is no longer necessary to do any programming on the FCB1010 itself. The floorboard just sends key presses to the TinyBox, and the TinyBox takes control over the FCB1010 display and LEDs. Instructions for replacing the FCB1010 firmware PROM can be found here :

https://www.fcb1010.eu/downloads/Upgrade%20Manual_FCB1010_Rev_A.pdf

3. Install TinyBox ControlCenter

Download the software from your account page in the TinyBox web shop. Installers both for Mac and for Windows are available

4. Make all connections

Connect the FCB1010 with the TinyBox using a 7-pins MIDI cable. Connect a power adapter (9V, 500mA or more, AC or DC) to the TinyBox power jack. Connect a USB cable between TinyBox and computer, connect a regular MIDI cable from TinyBox to controlled device.

5. Launch TinyBox ControlCenter

Launch TinyBox ControlCenter. Follow the instructions in the next chapters to verify the correct connection between TinyBox and computer.

Using a computer with the TinyBox

In order to program the TinyBox you need to connect it to a computer (Windows PC or Mac) using a USB cable. The computer runs the TinyBox ControlCenter software. You will find out all details about this software in the next chapter.

TinyBox ControlCenter not only allows you to program the TinyBox, but it can also communicate with the TinyBox during live use to request realtime status info: the current bank content, currently selected preset, activated effects, etc. The TinyBox ControlCenter can act as a lightweight web server, so any web browser can connect to it and display live status info of the FCB1010 and TinyBox. In the simplest scenario you can use ControlCenter itself to show the status view directly on your laptop, however you can as well use an iPad or other tablet as “wireless status display”. Of course, for this the device must have a WIFI connection with your computer. Most nowadays computers have WIFI built in and can easily be turned into a WIFI access point¹. An alternative is to connect through an external WIFI router, which can be purchased new for around 20 USD.



using laptop as WIFI access point



using an external WIFI router

¹ <https://www.imore.com/how-turn-your-macs-internet-connection-wifi-hotspot-internet-sharing>

¹ <https://www.windowscentral.com/how-turn-your-windows-10-pc-wireless-hotspot>

The TinyBox ControlCenter software

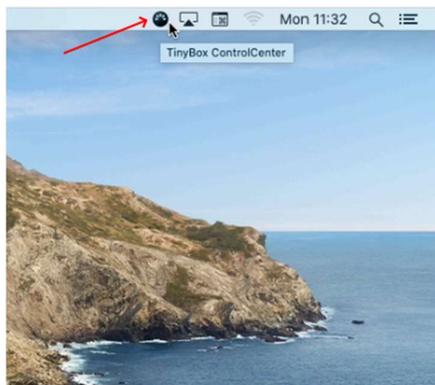
This software can be freely downloaded from our web shop. Once you have purchased a TinyBox a download link will appear on your account page. Installers are available for Windows and Mac.

Like the Tinybox itself also TinyBox ControlCenter is many things in one. It contains

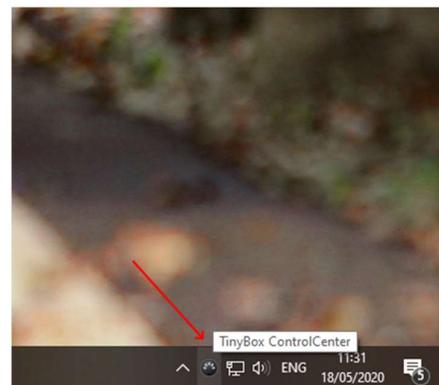
- an editor to create TinyBox setups and send them to the TinyBox through USB
- a setlist manager to upload song texts or scores to be displayed during live use
- a MIDI monitor app to check which MIDI messages the FCB1010 actually sends
- a status window showing currently selected song/bank/preset/activated effects etc...
- a web server hosting the same status window, so that the info can be displayed on a remote device connected through WIFI
- a TinyBox firmware upgrade tool

Launching TinyBox ControlCenter

After launching the software it shows up as a small MIDI connector icon in the Mac menu bar (in the upper right corner of the screen) or in the Windows tray area (in the lower right corner of the screen)



ControlCenter on the Mac menu bar

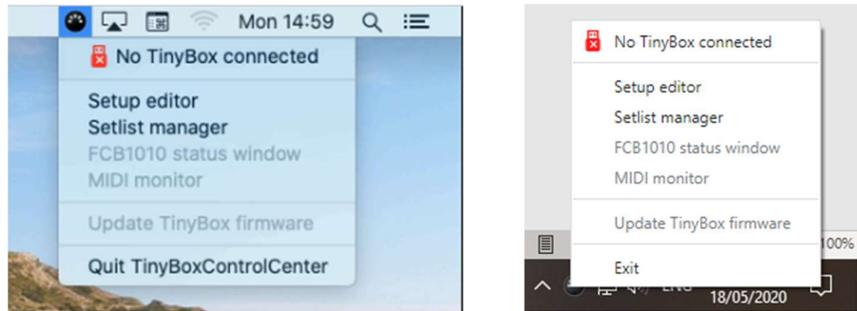


ControlCenter on the Windows status bar

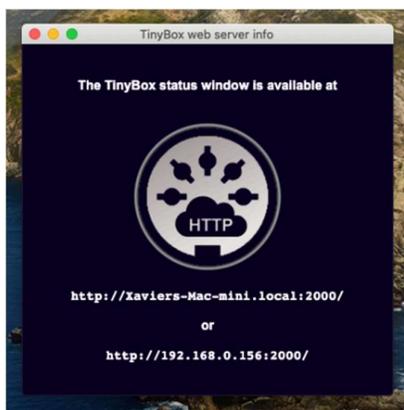
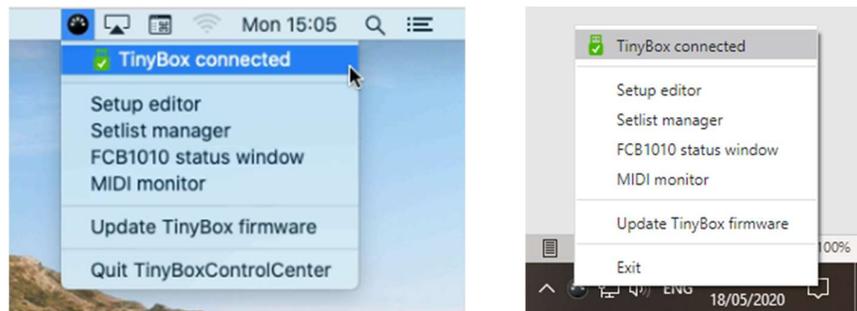
Click the icon to access all the different tools or to shut down the application. When you close all tools a notification will pop up to remind you that the ControlCenter application is still running. Indeed, the built-in web server will keep running until you explicitly quit the application.



As long as no TinyBox is connected to an USB port, some of the menu options will remain disabled.



As soon as a TinyBox is detected on the USB port (you don't need to install any drivers for that!) all menu options will be enabled, and you can click the topmost menu item to see the webserver address. An iPad or other remote device can surf to that address in order to display the current FCB1010 status, and optionally show song lyrics or scores :



Don't forget to add ":2000" to the URL as mentioned above. It indicates that the server communicates on port 2000. This way the default port 80 remains free for regular internet traffic.

Remark : a 'disclaimer'...

Next to the Windows and Mac versions of the software, we have also released a Linux version which is compatible with Raspberry Pi 3 (and higher). Being a very cost effective solution, the Raspberry Pi is gaining popularity. It can be a good alternative for bringing an expensive laptop on stage, an environment which is not always the safest for such gear. The low cost of a Raspberry Pi allows to always have second device available as backup.

Please be aware that the Linux version of the software is experimental, and not officially supported ! We do our best to keep the software source code as platform independent as possible, which allows us to release versions not only for the 2 main operating systems, but also for the Raspberry Pi. However, being a community driven project, the RasPi ecosystem is constantly moving forward and changing, and it is not feasible to guarantee constant compatibility of our software with this dynamic environment. We will do our best to provide a stable Raspberry Pi release, but its update frequency might be lower than for the 2 main operating systems.

Wireless TinyBox status display and remote control

When an iPad or other tablet is wirelessly connected to your laptop you can open a browser and surf to the address mentioned before. This will open following page :



This page gives you a representation of your FCB1010, showing detailed info about its current status:

- currently selected bank number and bank name
- currently selected preset
- active stomp boxes or effects
- available presets, stomp boxes and triggers in the current bank

Next to showing you the current status, this screen can also be used as an actual remote control for the TinyBox. You can use it as a “virtual FCB1010” which can even replace the real FCB1010. Keyboard players for instance might prefer using the iPad in front of them instead of a floorboard.

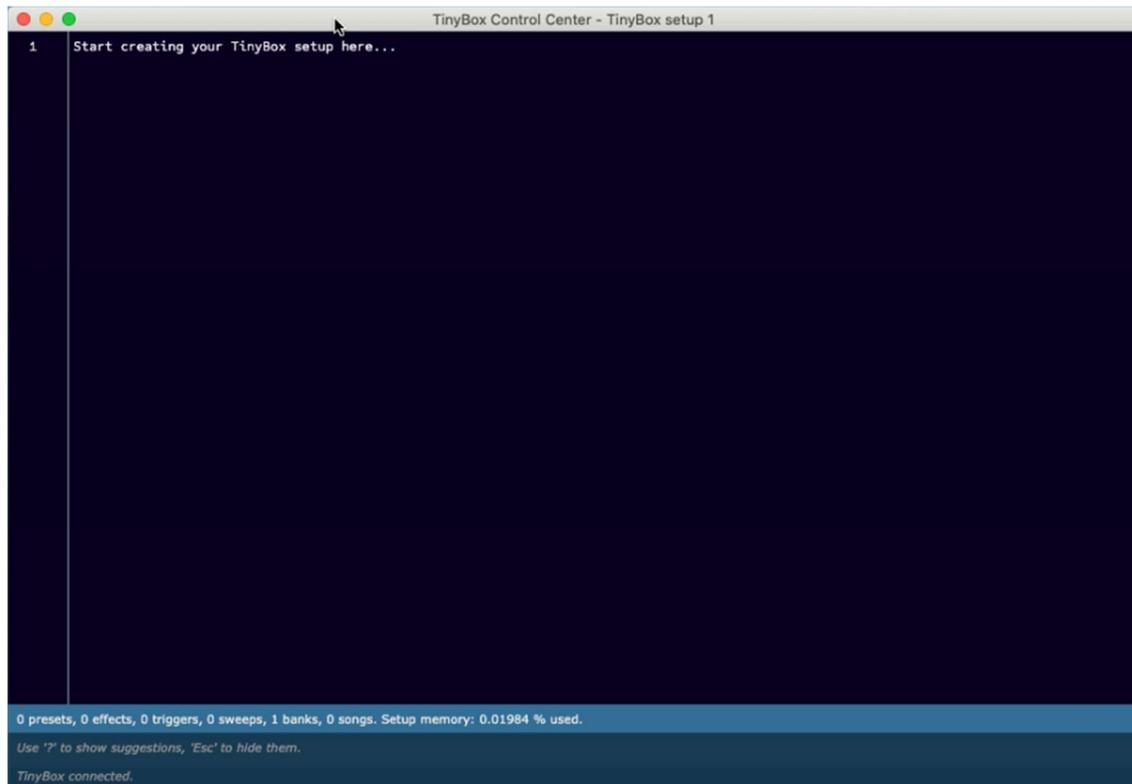
Whenever you touch the screen to select a preset, activate an effect, or scroll through banks, not only the TinyBox will react to these commands, but also a connected FCB1010 will adapt accordingly in order to remain in sync with the status screen at all time.

Displaying TinyBox status info on the laptop

Ticking the “FCB1010 status window” option in the ControlCenter context menu will open exactly the same status screen on your laptop. So if on stage you can directly look at your laptop screen, there is no need for a WIFI connection with another device, and you can monitor the status directly on your laptop using the ControlCenter status screen.

Using the TinyBox editor

In the ControlCenter menu tick the option “Setup editor”. This opens the editor window in which you can create, edit and manage your TinyBox setups.



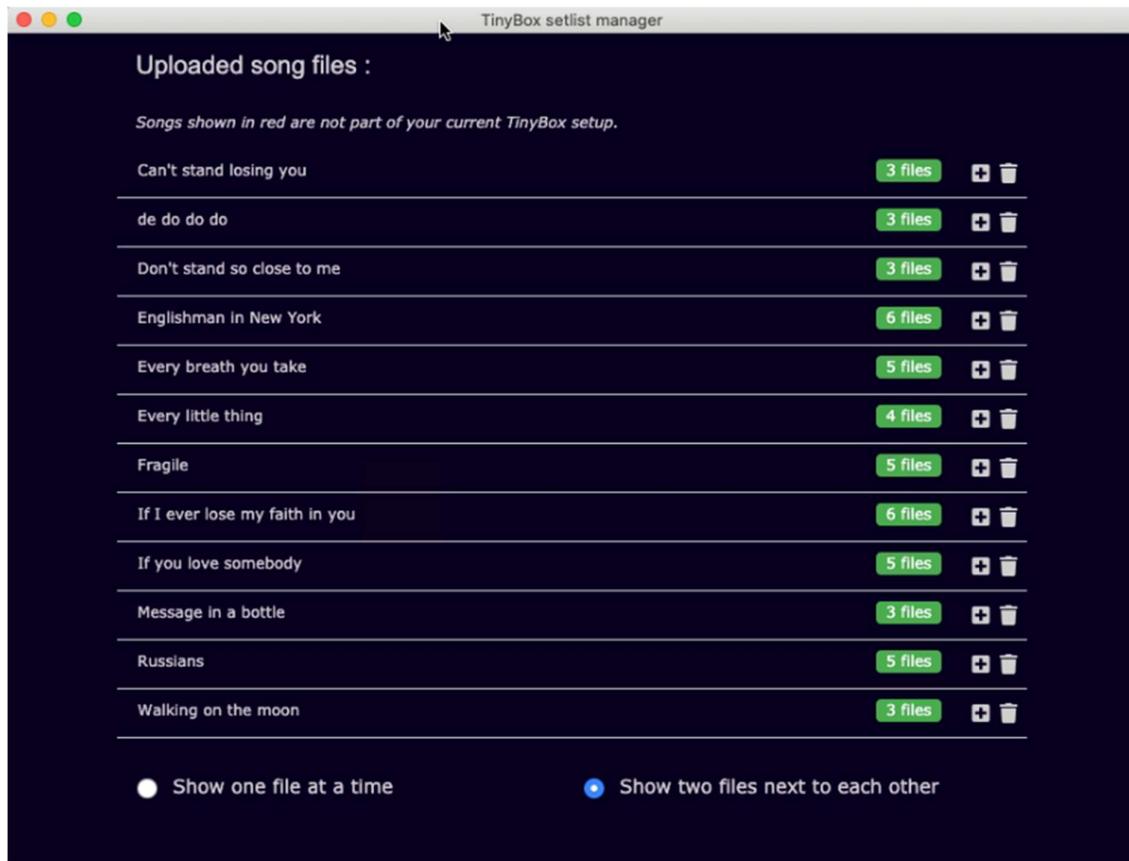
The menu options are self-explanatory. They allow you to create, rename and delete setups, save changes, and download a setup to your TinyBox.

One of the advantages of the TinyBox approach is that its setups are purely text based, so you can very easily copy-and-paste your setup into an external text file for backup on a stick, for sharing with others on a forum, etc...

Full details on the actual setup structure and programming syntax are given in a later chapter.

Managing setlists, song lyrics and music scores

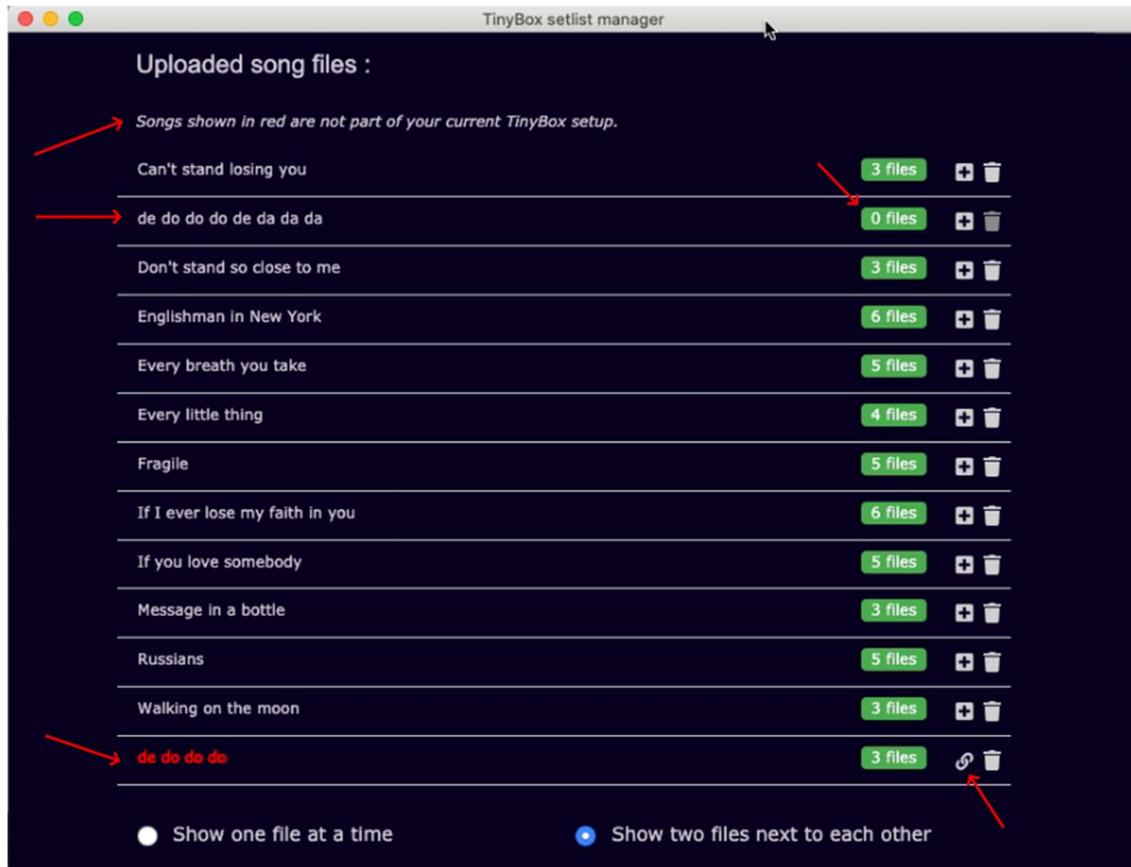
When ticking the “Setlist manager” option in the TinyBox ControlCenter menu, a tool pops up which lets you upload song lyrics, tabs or music scores to the built-in web server. These can be text files (.txt) or image files (.png or .jpg) which you may have created in any text or image editor.



Before opening this tool, make sure that you have specified all songs in your TinyBox setup (see further chapters for details on creating and editing a TinyBox setup). Then download this setup to your TinyBox. The setlist manager tool lists all songs which are defined in the setup currently stored in the TinyBox. For each song you are now able to upload one or multiple files. On the server the files will be renamed to match the song name, and they are sorted in alphabetical order of their original file name.

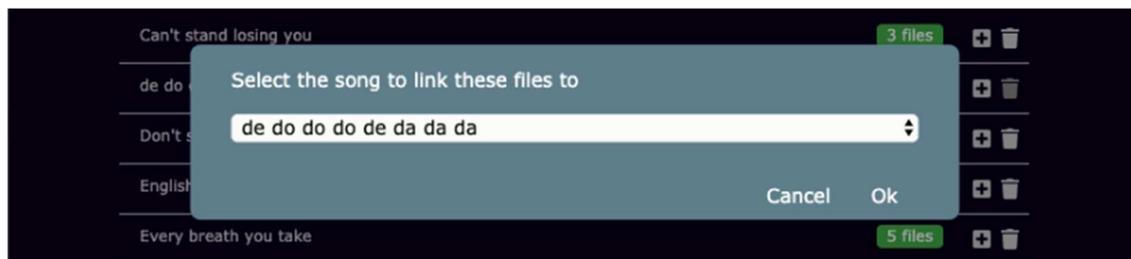
If at some point you would rename a song in your TinyBox setup, the setlist manager will no longer be able to link the uploaded song files to that song. This will be indicated in the setlist, and you will be able to manually re-link the uploaded files to the renamed song, or you can also remove the files and upload new ones.

For instance, suppose we rename the song “de do do do” in our TinyBox setup to “de do do do de da da da”. As a result, we will see the following when we open setlist manager :

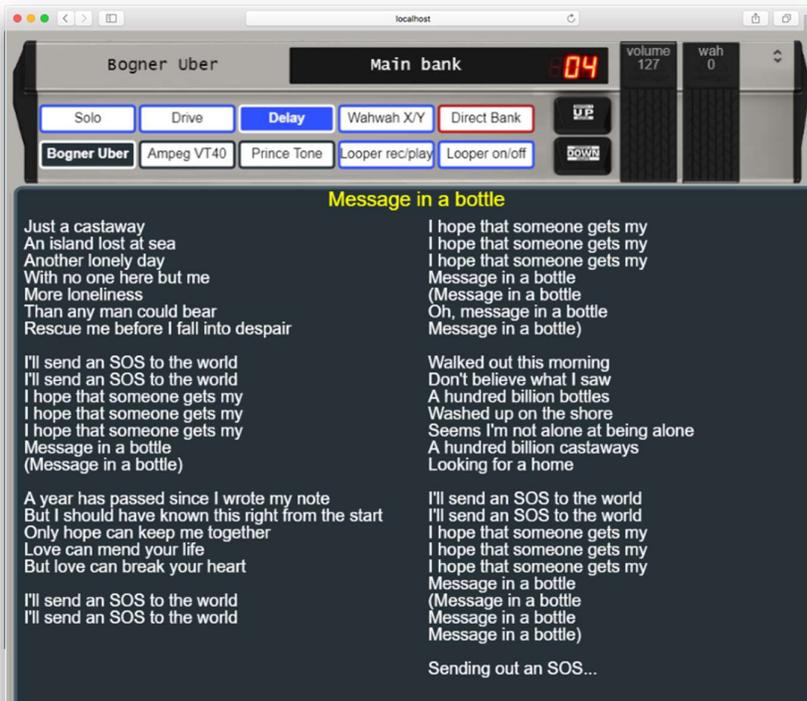
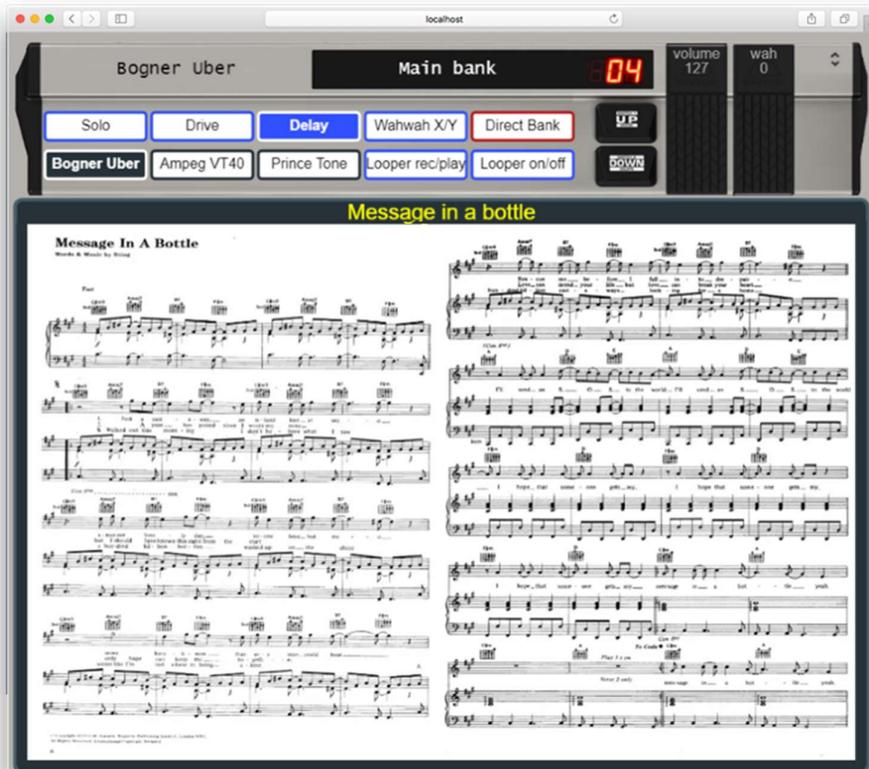


Songs for which you have uploaded song files to the server, but which are no longer part of your TinyBox setup, are shown in red at the bottom of the setlist. In our example that’s the case for the song “de do do do” which we just renamed. Indeed, we now see the renamed song “de do do do de da da da” in our setlist, with no song files uploaded yet.

We can delete the files for “de do do do”, after which the red entry in the list will disappear, and we can upload new files for “de do do do de da da da” using the “+” icon next to that song name. But as the song files probably didn’t change it is easier to just link the uploaded files to the renamed song, by clicking the “link” icon next to the red song name :



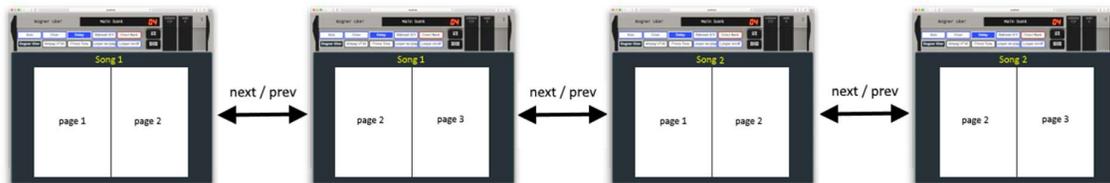
After uploading song files in the setlist manager, the FCB1010 representation on the status page will be shrunk in order to take less space while still showing the same status info, and the song files (lyrics or music scores) will be displayed under the FCB1010 image :



The setlist manager lets you choose to display two images or text files simultaneously next to each other, or to display one single image or text file at a time. Image files are scaled to take all the available space, while for text files the used font size will be adapted to fill the available space. If readability of song lyrics is a problem you can increase the font size by splitting up the text in multiple separate text files.

When clicking the up or down switches of the FCB1010, you will scroll through all song files for each song one by one: if you have multiple files for a song, the up/down switches will now scroll to the next or previous score or lyrics file within the current song, rather than immediately jumping to the next or previous song.

In “dual file” display, the score files will be scrolled through as follows, if you have 3 files per song:



Remark :

Be aware that the DOWN switch will proceed to the next song, and the UP switch will go back to the previous song. This is different compared to bank scrolling, where you proceed to the next bank by clicking the UP switch. While on first sight this might seem confusing to some, it actually is more intuitive to scroll through a setlist this way: when you go through a paper setlist, you also go DOWN to see the next song title, don't you?

On the FCB1010, the small green LEDs next to the songnumber display show how many lyrics pages are still to be shown for the current song. If no LEDs are on, you know that a click on the “Down” button will proceed to the next song. If one or more LEDs are on, you know that a click on the “Down” button will not proceed to the next song but will instead proceed to the next lyrics page on the FCB1010 status display.



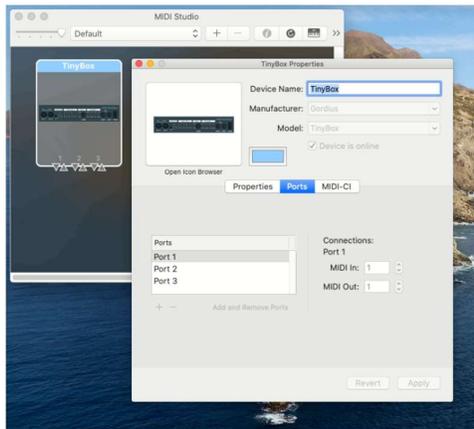
The menu LEDs above indicate there are still 4 lyrics pages to go for the current song

When you want to scroll quickly down or up your setlist, you can keep the FCB1010 up or down switch pressed. When doing so, after a small delay the FCB1010 will start auto-scrolling through the songs of the setlist without scrolling through all individual songfiles for each of the songs. Fast scrolling will continue until you release the footswitch, or until you reach the last or first song in your setlist.

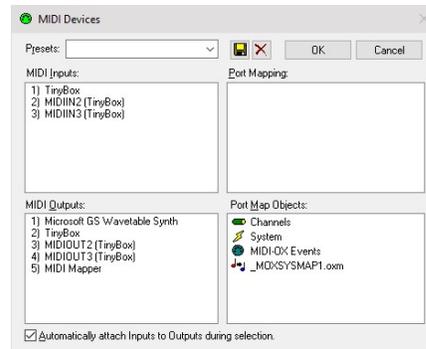
Monitoring the TinyBox MIDI messages

All MIDI messages programmed in the TinyBox are sent both to the MIDI OUT connector of the TinyBox and to port 1 of the built-in MIDI-USB interface.

Indeed, when connecting a USB cable between TinyBox and your computer, you will notice that a total of 3 MIDI In and 3 MIDI Out ports are detected :



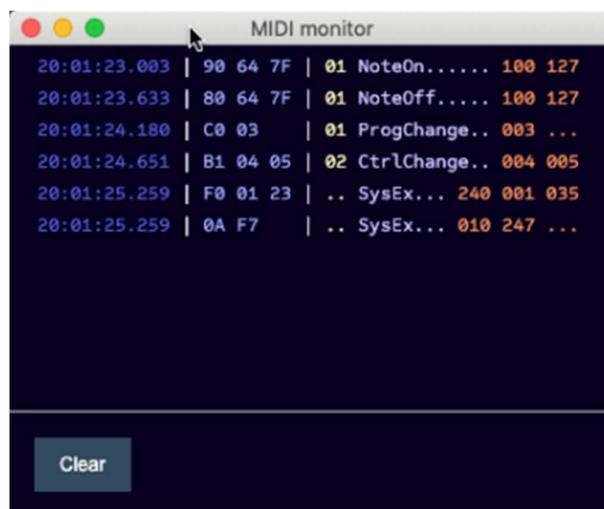
Midi Studio on Mac



MIDI-OX on Windows

Port 3 is used for transferring TinyBox setups, port 2 is used for getting TinyBox status info and for remotely controlling the TinyBox from your iPad or laptop. Port 1 is available for your setup, through this port you can control any music software application with your FCB1010.

Incoming MIDI traffic on port 1 can be monitored with the MIDI monitor tool included in TinyBox ControlCenter. Click the "MIDI monitor" option in the ControlCenter context menu to open it. Just make sure you have closed down the other music applications prior to launching the monitor tool, as the MIDI ports can only be used by 1 application at the same time.



Upgrading the TinyBox firmware

TinyBox firmware upgrades will be made available when necessary under the form of a binary .tbf file which can be downloaded from the tinybox.rocks website. This file can be sent to the TinyBox using the “Upgrade TinyBox firmware” option in the ControlCenter context menu :



After clicking this option, select the firmware file. It will be automatically downloaded to the TinyBox and installed. You will understand that under no circumstance should you power off or disconnect the TinyBox during this upgrade process. It would be difficult to do so, since the full procedure only takes a few seconds. You will be notified about the successful firmware download, after which you have to power cycle the TinyBox for the new firmware to be applied. It's definitely a good idea to also quit and (if needed) relaunch TinyBox ControlCenter after a TinyBox power cycle.



The TinyBox setup structure

A TinyBox setup can contain :

- up to 199 banks, through which you can scroll using the FCB1010 up/down switches, plus one optional “direct” bank, accessible with 1 foot click from within any bank.
- up to 1000 presets. Each preset can contain a virtually unlimited number of MIDI commands to be sent on any of the 16 MIDI channels.
- up to 1000 effects. When a footswitch is linked to an effect, it behaves as a toggle switch with 2 states (ON/OFF), and it sends a different set of MIDI commands for each of the 2 states.
- up to 500 triggers. When a footswitch is linked to a trigger, it behaves as a momentary switch which can send 2 different sets of MIDI commands: one on switch press and another one on switch release.
- up to 250 different “sweeps”. Sweeps are behaviors for the FCB1010 expression pedals. They not only describe which continuous MIDI commands need to be sent by the expression pedal (ControlChange, PitchBend, ChannelPressure) but also which value range, and even which type of sweep curve (linear, fast rising, slow rising) they need to follow.
- a setlist of up to 250 songs. Each song can activate one of the available preset banks, and a song text or score can be shown on your laptop or wireless device (iPad, Windows or Android tablet, ...)

Each TinyBox preset, effect or trigger can contain any combination of different types of commands:

- MIDI commands like *ProgChange*, *CtrlChange*, *NoteOn*, *NoteOff*, *MIDIStart*, *MIDIContinue*, *MIDIStop*, *MIDIClock*, *SysEx*
- a *Wait* command, which halts MIDI transmission for a programmable number of milliseconds or seconds
- commands to (de)activate effects or triggers, to modify the behavior of the 2 FCB1010 expression pedals, and to control the 2 FCB1010 jack outputs
- commands to manage the embedded MIDI filter/router. With these commands you can block specific MIDI channels or command types, copy ranges of MIDI commands from one MIDI channel to another, etc.
- variable commands, which set or modify the content of the different types of data variables
- conditional commands, which act upon the current content of those variables

A TinyBox setup can use up to 128 *numeric* variables, up to 256 *boolean* variables, and up to 256 *string* variables.

- a numeric variable can contain any number in the range 0-127. It can be used in any MIDI command instead of specifying a hardcoded value.
An integer variable can be incremented, decremented, added or subtracted, and compared to other integer variables to make decisions.
- a boolean variable can be true or false. Different messages can be sent depending on the current value of a boolean variable.
- a string variable can be used in comparisons for making decisions. (a *string* in the context of programming languages is a short piece of text, a word). Using string values instead of integers can help to make your setup more readable.

Example 1 : structure of a typical TinyBox setup

```
/* Below you see the general structure of a TinyBox setup.
   You can add as much text comment to a setup as you like.
   A single-line comment starts with 2 slashes, as shown below.
   A multi-line comment is embedded between the symbols you see here */

// Start with listing all presets, effects, triggers and sweeps in your setup:

PRESETS =
{
  Vox Ac-30
  65_Bassman
  HiWatt Bright
  Soldano SLO1
  // etc...
}

EFFECTS =
{
  Chorus
  Compressor
  Delay
  // etc...
}

TRIGGERS =
{
  Sustain
  Looper rec/play
  Looper dub
  // etc...
}

SWEEPS =
{
  volume
  wah
  whammy
  // etc...
}

// Then organize all presets, effects and triggers in a bank layout

BANKS =
{
  Looper          : Looper on/off | Looper rec/play | Looper dub | Looper reverse ...
  Preset bank 1  : Vox Ac-30      | 65_Bassman   | HiWatt Bright | Soldano         ...
  Preset bank 2  : RectoOrangeV   | Friedman HBE | Trainwreck X  | Cameron High   ...
  Preset bank 3  : Engl Powerb    | Fry D60 Mor  | FAS Modern    | Diezel VH4     ...
  // etc...
}

// Optionally you can also define songs, link each of them to a bank,
// and organize them in a setlist

SONGS =
{
  Another Brick      : Preset bank 10
  Beds Are Burning  : Standard bank
  Ca Plane Pour Moi : Looper
  Chasing Cars      : Preset bank 2
  // etc...
}

SETLIST =
{
  White Wedding
  Fortunate Son
  Twilight Zone
  Gloria
  Oh Darkness
  Kryptonite
  // etc...
}
```

```

// Now we are ready to define which MIDI messages each preset needs to send

// First thing to do is define the MIDI channels to be used in your setup.
// Giving each channel a meaningful name will help a lot to make your setup readable

CHANNEL Profiler = 1
CHANNEL Helicon = 2
CHANNEL Strymon = 3
CHANNEL Octaver = 10

// Before defining the presets, you might want to specify some global initialization
// which is done when the TinyBox is powered :

INIT_TINYBOX =
{
    Pedal 1 = volume
    Pedal 2 = wah
    BlockChannels [11-16]
}

// It's also possible to define MIDI commands to be sent when a certain song or bank
// is activated :

INIT_SONG SongForTheDead =
{
    SendMidi Profiler ProgChange 125
    SendMidi Helicon ProgChange 3
    SwitchOn Chorus
    SendRealtime MIDISTart
    SendRealtime MIDIClock 107 BPM
    Pedal 2 = whammy
}

// A preset content can be as simple as a single MIDI command ... :

PRESET Vox Ac-30 = SendMidi Profiler ProgChange 2

// ... or multiple lines surrounded by curly braces :

PRESET 65_Bassman =
{
    SendMidi Profiler ProgChange 5
    SendMidi Helicon ProgChange 17
    SendMidi Octaver CtrlChange 13 127
}

// it is also possible for presets to send MIDI messages on switch release :

PRESET_RELEASE 65_Bassman = SendMidi Profiler CtrlChange 11 127

// a stompbox effect will typically send at least 2 different MIDI messages :

EFFECT_ON Chorus = SendMidi Strymon CtrlChange 3 127
EFFECT_OFF Chorus = SendMidi Strymon CtrlChange 3 0

// a trigger can send a single message ... :

TRIGGER_CLICK LooperOn/Off = SendMidi Helicon NoteOn 69 127

// ... or a message on switch press and another message on switch release :

TRIGGER_CLICK OctaveUp = SendMidi Octaver CtrlChange 13 127
TRIGGER_RELEASE OctaveUp = SendMidi Octaver CtrlChange 13 0

// a sweep defines which MIDI commands an expression pedal can send :

SWEEP volume =
{
    SendMidi Profiler CtrlChange 7 0-127 SlowRising
    SendMidi Strymon CtrlChange 7 40-100 SlowRising
}

SWEEP whammy = SendMidi DX7 PitchBend 0-127

```

Example 2 : sending MIDI messages

```
// The example below gives an overview of all supported MIDI messages
// A preset can send one single message or multiple messages on different channels

CHANNEL MyGear = 10
CHANNEL MySynth = 3

// Instead of using fixed values you can also use "variables" in MIDI commands,
// as will be shown in more detail later on

VAR $cc = 10
VAR $delay = 20
VAR $mix = 100

PRESET SimplePreset = SendMidi MyGear ProgChange 1

PRESET ComplexPreset =
{
  SendMidi MyGear ProgChange 125
  SendMidi MyGear CtrlChange 13 127
  SendMidi MySynth NoteOn 72 120
  SendMidi MySynth NoteOn 76 120
  Wait 20
  SendMidi MySynth NoteOff 76 0
  SendMidi MySynth NoteOff 72 0

  // The Wait command above inserts a delay between 2 MIDI messages.
  // It is expressed in 0.1s units, so 'Wait 20' introduces a 2 second delay.

  SendRealtime MIDIClock 120 BPM
  SendRealtime MIDIStart
  SendRealtime MIDIContinue
  SendRealtime MIDIStop
  SendRealtime SystemReset

  // MIDIClock is a special case : the command doesn't send one single MIDI message, but it
  // starts a continuous stream of MIDIClock messages using the given tempo in beats per minute

  SendSysCommon SongSelect 100
  SendSysCommon SongPointer 16000
  SendSysCommon TuneRequest

  SendSysEx F0 00 20 33 02 7F 01 00 32 59 00 40 F7

  // Integer variables can be used for all the values in the MIDI messages above.
  // Even a SysEx message can contain variables:

  SendMidi MyGear CtrlChange $cc 127
  Wait $delay
  SendSysEx F0 7F 01 $mix F7
}
```

Example 3 : programming expression pedals

```
CHANNEL MyGear = 10

// You can modify the MIDI commands, range and sweep type for each of the expression pedals.
// Sweep type is linear by default, but can also be set to SlowRising or FastRising

// First you define all available continuous controls or "sweeps" :

SWEEP volume = SendMidi MyGear CtrlChange 07 0-127 SlowRising
SWEEP whammy = SendMidi MyGear CtrlChange 19 0-127
...

// Then you can specify in each preset or effect which sweep
// needs to be active on each of the 2 expression pedals.
// For instance, a stomp switch could change the wah pedal into a whammy effect :

EFFECT_ON ActivateWhammy = Pedal 2 = whammy
EFFECT_OFF ActivateWhammy = Pedal 2 = wah

// you can even define a virtual "tipswitch" or "heelswitch" for each expression pedal :

Tipswitch 2 = ActivateWhammy

// If you link the effect above to the "tipswitch" of pedal 2, the pedal will toggle between
// both functions when pressing the pedal tip all the way down.
// It's a good idea to stick a small piece of foam under the pedal. This way you have
// the full adjustment range by moving the pedal, and the virtual switch will only be activated
// when putting some extra force on the pedal tip.

// As is the case for presets, also sweeps can contain multiple commands
// surrounded by curly braces:

SWEEP volume =
{
  SendMidi MySynth CtrlChange 07 0-127
  SendMidi MyAmp CtrlChange 07 0-127 SlowRising
}

// This way one pedal can send multiple parallel streams of MIDI commands
// simultaneously on different MIDI channels
```

Example 4 : using variables

```
// The use of 'variables' is something very common in programming languages.
// It adds huge possibilities to the TinyBox.

// There are 3 types of variables:
// - integer ( = numeric )
// - boolean ( = true or false )
// - string ( = text )
// You can recognize a variable by its leading '$' :

VAR $currentBank = 1                // these are integer variables
VAR $currentPreset = 35
VAR $delay = false                  // this is a boolean variable
VAR $currentSong = "Go with the flow" // this is a string variable

// You can set the value of each variable whenever a certain preset is triggered :

PRESET no_one =
{
    $currentSong = "No one knows"
    $delay = true
    // ...
}

// You can work with integer or boolean values in different ways :

PRESET examples =
{
    $currentBank++                // increment
    $currentBank--                // decrement
    $currentBank += 10            // add a constant value
    $currentPreset = $currentBank + 5 // calculate
    $delay = !$delay              // invert a boolean value
}

// There is one pre-defined integer variable, named $MidiChannel
// You can use that variable to have a dynamically changing MIDI channel in certain commands :

EFFECT_ON Boost = SendMidi $MidiChannel CtrlChange 7 127
EFFECT_OFF Boost = SendMidi $MidiChannel CtrlChange 7 64

PRESET Control_Device1 =
{
    $MidiChannel = 1
}

PRESET Control_Device2 =
{
    $MidiChannel = 2
}

// The true strength of variables will become clear in the next example,
// covering conditional commands
```

Example 5 : using conditional commands

```
// 'Conditional commands' are another common concept in traditional programming languages:
// 'if...then...else...' statements allow an application to make decisions.
// The examples below show how your setup can behave differently depending on the value
// of any variable.

PRESET Sample =
{
  // here $solo is a boolean variable, it can be true or false :

  if($solo)
  {
    // send a set of MIDI messages...
  }
  else if($currentBank > 1)
  {
    // send another set of messages...
  }

  // a 'switch' statement checks the value of a variable
  // and specifies the code to be executed for each value :

  switch($currentSong)
  {
    case "No one knows":
      SendMidi MyGear CtrlChange 112 127
      break
    case "Go with the flow":
      SendMidi MyGear CtrlChange 12 0
      SendMidi MyGear CtrlChange 113 127
      break
    // and so on...
    default:
      SendMidi MyGear CtrlChange 12 127
      break
  }

  // By using a 'while' statement it is even possible to create loops, for instance
  // in order to do an effect fade-in/fade-out

  $fadeout = 100
  $fadein = 0
  while($fadeout > 0)
  {
    SendMidi MyGear CtrlChange 07 $fadeout
    SendMidi MyOtherGear CtrlChange 07 $fadein
    Wait 1
    $fadeout -= 5
    $fadein += 5
  }
}
```

Example 6 : a programmable MIDI filter

```
// The TinyBox can act as a fully programmable and dynamic MIDI filter.
// You can configure a fixed MIDI filter for all incoming MIDI,
// or you program presets to adapt that filter at any time.

// Several levels of filtering are available:
// - block certain message types (ActiveSense, SystemCommon, SystemRealtime)
// - block certain MIDI channels or channel ranges
// - block certain ControlChange numbers
// - block a range of Note messages

// Some examples of the commands available for programming the MIDI filter :

INIT_TINYBOX =
{
  BlockMidi ActiveSense
  BlockMidi SystemCommon
  BlockMidi SystemRealtime
  BlockChannels [8-16]
  BlockChannel LittleGiant
  BlockCC ElevenRack 7
  BlockNotes Hammond C0-C2

  AllowMidi ActiveSense
  AllowMidi SystemCommon
  AllowMidi SystemRealtime
  ResetChannelRouting LittleGiant
  ResetCCRouting ElevenRack 7
  ResetNoteRouting Hammond
  ResetMidiFilter
}
```

Example 7 : a programmable MIDI router

```
// The TinyBox can act as a fully programmable and dynamic MIDI router.
// This means that you can move or copy MIDI messages from one MIDI channel
// to another, from one CC number to another, or from one Note range to another.

// This allows you for instance to turn a simple MIDI keyboard into a full blown
// MIDI master keyboard with different zones controlling different synths,
// or you could transpose the keyboard to the key of each song.

// Some examples of the commands available for programming the MIDI router :

INIT_TINYBOX =
{
  CopyChannel LittleGiant to ElevenRack
  MoveChannel Yamaha to Hammond
  CopyCC LittleGiant 07 to Yamaha
  MoveCC LittleGiant 07 to ElevenRack 04
  CopyNotes Yamaha C1-C8 to Hammond C2
  MoveNotes Yamaha C0-C1 to Rhodes

  ResetChannelRouting LittleGiant
  ResetCCRouting LittleGiant 07
  ResetNoteRouting Yamaha
}
```

The TinyBox programming language

The following pages describe in detail the syntax to be used for creating a TinyBox setup.

It is important to preserve the correct order of the different parts in your setup :

1. Define preset, effect, trigger and sweep names
2. Define the bank structure of the setup
3. Optionally define songs and setlist
4. Optionally define commands to be sent when powering the TinyBox
5. Optionally define commands to be sent when selecting each song or bank
6. Define all contents for the presets, effects, triggers and sweeps (in any order)

0. Comments

Multiline comments can be used at any place in the setup in order to document the different parts of the setup

```
/*  
  [ any multiline  
    text here... ]  
*/
```

Inline comments can be used at the start of any line or at the end of any command

```
// [any inline text here...]  
SendMidi KPA ProgChange 10 // [any comment at the end of the line]
```

1. Defining preset, effect, trigger and sweep names

At the start of the setup you have to list all setup element names (presets, effects, triggers and sweeps). These names can then be used when specifying the setup layout and the commands for each preset.

While not all 4 element types are mandatory, you will at least need to specify some presets before you can continue defining the other parts of your setup.

INFO

- a **preset** is the main element in your setup. It is typically activated by clicking one of the 10 footswitches of the FCB1010. A (virtually unlimited) number of MIDI commands is sent when activating the preset, and the footswitch LED turns on to indicate that this preset is currently active. Only one preset can be active at the same time, so selecting a different preset will automatically switch off the LED of the previous preset. Although probably little used, if needed a preset can also send commands on switch release.
- an **effect** can also be assigned to one of the 10 footswitches of the FCB1010. The main difference with a preset is that an effect typically has 2 states : ON or OFF. Clicking the effect footswitch toggles between those 2 states, and the switch LED turns on or off along with the effect. As opposed to presets, it is perfectly possible to have multiple effects activated at the same time. You will need to define which MIDI commands to send both on effect *activation* and on effect *deactivation*.
- a **trigger** is very similar to an effect, except that it doesn't toggle between 2 states on each click. Instead it goes ON when pressing the footswitch, and OFF when releasing the footswitch. Therefore you could also call it a **momentary effect**. A sustain pedal is a typical example for such a momentary effect: it is only activated while you keep the footswitch pressed. Just like with an effect, you need to specify at least 2 MIDI commands: one for activating, one for deactivating the effect.

The reason why we call this a **trigger** is because this same setup element can also be used to *trigger* a certain action. A looper command is a good example: a "REC/PLAY" or "UNDO/REDO" command is just a one-shot command sent to the looper. In this case you will only specify a MIDI command for the footswitch press, and probably none for the footswitch release. As opposed to a preset or effect the footswitch LED of a trigger doesn't stay on after releasing the trigger switch.

- a **sweep** is a very specific type of preset. You cannot link it to any footswitch, instead you link it to one of the two expression pedals of the FCB1010. The sweep content will specify which continuous MIDI commands each of the pedals will send when moving them – it will typically be ControlChange commands for modifying volume, expression, or continuous effects like wah or whammy. The fact that a TinyBox setup provides "sweep" elements allows you to easily modify the expression pedal behavior depending on the currently active preset for instance.

SYNTAX :

```
PRESETS =
{
    [preset name]
    ...
}

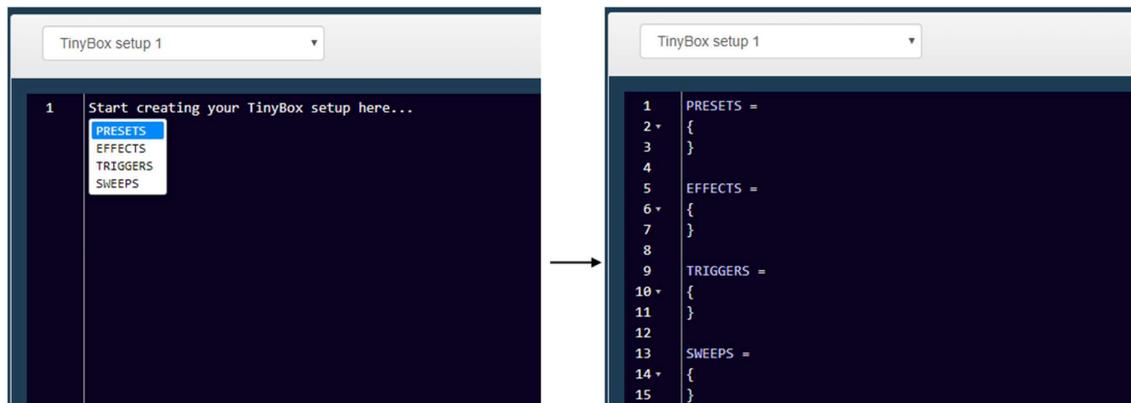
EFFECTS =
{
    [effect name]
    ...
}

TRIGGERS =
{
    [trigger name]
    ...
}

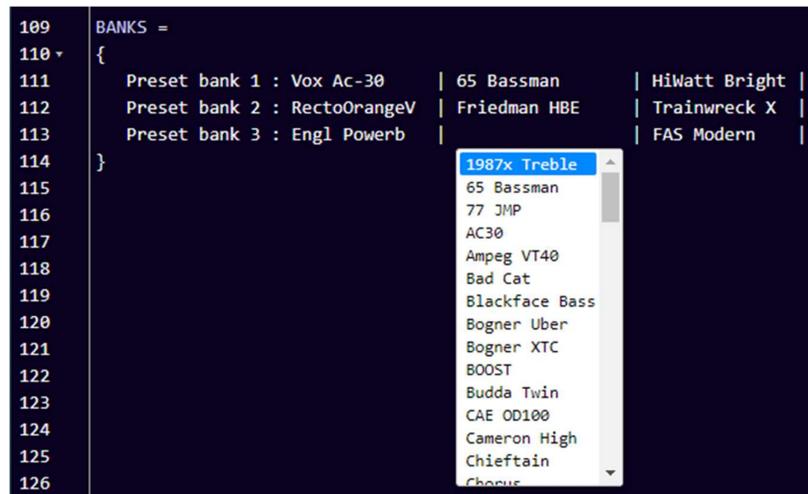
SWEEPS =
{
    [continuous control name]
    ...
}
```

In each of the 4 lists, specify one name per line. A few reserved characters like “ (or) cannot be used in preset names.

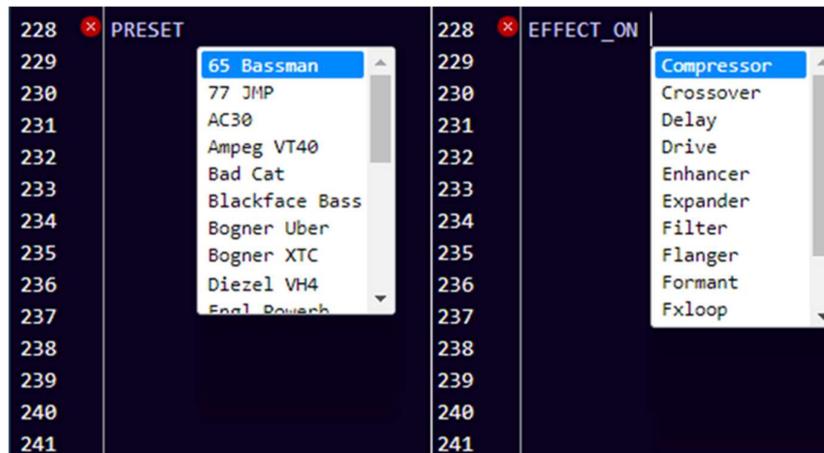
The autocomplete or “intellisense” functionality of the TinyBox editor helps you setting up this initial setup structure : after creating a new setup, type ‘?’ to get a dropdownlist of available commands. Clicking <ENTER> 4 times on the proposed command list will give you the 4 empty lists which you can now start filling with preset names :



The TinyBox setup editor uses these name lists in order to help you when specifying further parts of the setup. For instance when defining the bank layout (as explained later on in this chapter), the editor will automatically propose a list of available presets to choose from :



Also when defining the preset contents further down the setup, a dropdown list will appear to select each of the available presets, effects, triggers or sweeps :



As you should define the content of each preset only once in your setup, the editor gradually adapts the dropdown lists and shows only those presets which have no content defined yet.

2. Defining the bank structure

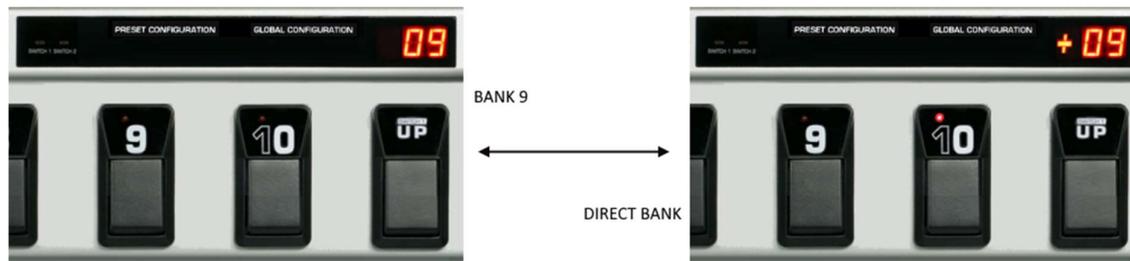
Once you have listed all presets, effects and triggers to be used in your setup, you can start organizing them in banks.

By default a TinyBox setup is structured in banks of 10 presets, corresponding with the 10 preset switches on the FCB1010 floorboard. The Up and Down switches allow you to browse through all banks. The possible size of your setup is huge, compared to the 10 banks available in a regular FCB1010: you can define up to 199 banks! Why 199? Because that's the highest number which can be shown on the "2.5 digit" display of the FCB1010 :



Direct Bank

Next to this default layout of 199 "sequential" banks, accessible using the Up/Down switches, you can also choose to add a "Direct Bank". This Direct Bank typically contains a number of presets or effects which you want to access easily at any time. Or it can contain a specific set of commands (for looper control for instance), while the regular banks contain different sound presets. When using a Direct Bank, this bank can be activated with footswitch 10 of the FCB1010, no matter which bank you are currently in. One click activates the Direct Bank, another click on the same switch 10 returns to the previous bank. A "+" sign on the display indicates that the Direct Bank is currently active :



Since in this mode footswitch 10 is a dedicated Direct Bank toggle switch, each bank can now contain 9 presets instead of 10.

SYNTAX :

```
GLOBALSWITCH [1...10] = [presetname]

BANKS =
{
    [bank name] : [preset name] | [preset name] | ... | [preset name]
    [bank name] : [preset name] | [preset name] | ... | [preset name]
    ...
}

// or if you want to use the Direct Bank functionality :

GLOBALSWITCH [1...10] = [presetname]

USE_DIRECT_BANK

BANKS =
{
    Direct Bank : [preset name] | [preset name] | ... | [preset name]
    [bank name] : [preset name] | [preset name] | ... | [preset name]
    ...
}
```

When you add the instruction “USE_DIRECT_BANK” prior to the BANKS specification, the first line in the bank list will define the Direct Bank contents. You will notice that the editor will automatically change the first bank name to “Direct Bank”, this cannot be modified.

Global switches

Even without using the “Direct Bank” structure you can have one or a few effects available in all banks, simply by linking the same preset or effect to the same switch in each bank. In order to make that easy for you, you can add one or several “GLOBALSWITCH” definitions prior to the BANKS specification. When doing so, the editor will automatically fill in the given preset name(s) on the given switch position(s) in each new bank.

For instance, after specifying switch 5 to be a global switch for TapTempo, you can simply click <ENTER> in the BANKS section (between the 2 curly braces), and the editor will create a new bank template for you, with the switch 5 position already containing the TapTempo preset :



After that you can just start typing a preset name for each switch position, and a dropdown list will appear to auto-complete what you are typing with valid preset, effect, or trigger names :



A “pipe” character (vertical line) is automatically added between each switch assignment, and the smart editor will automatically adapt the spacing between assigned presets in order to obtain a neatly formatted grid layout:



As explained above you can specify one “Direct Bank” before creating the list of regular banks. In that case all bank definitions will contain 9 presets instead of 10.

If you want to keep a switch “empty”, just leave the switch assignment empty without removing any of the vertical separator lines in the bank definition.

Remark :

It is good to understand exactly how the GLOBALSWITCH definitions, which can be added prior to the bank list, work. Actually these are just a convenience feature of the editor to avoid having to select the same preset for that global switch over and over again in each bank. The editor fills the switch position automatically for you. Other than that, this global switch definition is not sent to the TinyBox, so in fact the switch is not “forced” to contain this one preset only – you can still modify the content of a global switch in any of the banks afterwards. You could for instance add or modify a GLOBALSWITCH definition after already having defined a large number of banks, and this setting will be used from then on for all new banks added to the setup, without modifying anything in the switch assignment of already defined banks.

Special case : using all 12 switches as preset switch

SYNTAX :

```
NO_UPDOWN_SWITCHES

BANKS =
{
    [bank name] : [preset 1] | [preset 2] | ... | [preset 12]
}

// or if you want to use the Direct Bank functionality :

NO_UPDOWN_SWITCHES
USE_DIRECT_BANK

BANKS =
{
    Direct Bank : [preset 12] | [preset 13] | ... | [preset 22]
    [bank name] : [preset 1] | [preset 2] | ... | [preset 11]
}
```

In case you don't need multiple banks in your setup, you can choose to use the Up/Down switches of the FCB1010 as 2 additional preset switches. Add the line **NO_UPDOWN_SWITCHES** to your setup before the bank definition, and you will be able to specify 12 presets in one bank instead of 10.

Since the Up/Down switches of the FCB1010 don't contain an LED, the state of those 2 preset switches is shown on the small "SWITCH1" and "SWITCH2" LEDs instead: the SWITCH1 LED is on when the preset of the "Up" switch is active, the SWITCH2 LED is on when the preset of the "Down" switch is active.

This functionality can also be combined with **USE_DIRECT_BANK**. This results in a setup with 2 banks containing 11 presets each. In this case, the "Down" footswitch is used instead of footswitch 10 to toggle between the 2 banks.

3. Defining songs and setlist

SYNTAX :

```
SONGS =
{
    [songname] : [bankname]
    ...
}

SETLIST =
{
    [songname]
    ...
}
```

This part of the setup is optional. You can perfectly use the FCB1010 without defining any songs. In that case you use the Up and Down switches to scroll through the different banks defined before. However you can also choose to define songs in your setup, and organize these songs in a setlist. When you do so, the Up and Down switches will scroll through these songs instead. You link each song to a bank defined before, and that bank will automatically be activated when you scroll to the song.

The order in which you scroll through the songs is defined in a “setlist”. The advantage of working with a setlist is obvious : it allows you to define all songs in your repertoire, link them to the required FCB1010 bank, and then specify which sublist of the full repertoire will be used in the next gig. Remember that you can comment out any line in the setup by prepending “//”. This way you can easily skip one or a few songs from a setlist. Alternatively you can use a multiline comment “/* ... */” to have a number of different setlists added to your setup file, and before each gig you simply uncomment the setlist you want to use that night.

What are the advantages of using a setlist and songs ?

- It allows you to reuse the same bank layout for multiple songs. Suppose you play 40 songs during the gig, which use 8 different banks. Instead of requiring multiple up/down clicks between each song to search for the correct bank, the setlist setup covers that for you and a single “Up” click advances to the next song.
- When you make use of the wireless status display for your FCB1010 (as explained in an earlier chapter) the screen shows you which song is up next, so no more need for paper setlists in that case.
- On top of that, the wireless status display can also display song lyrics or the music score for each song. Super easy!

Remark :

When using the song lyrics or score functionality described in an earlier chapter, remember that renaming a song in your TinyBox setup will also require re-linking the uploaded lyrics or score files to the renamed song. See the earlier chapter about managing lyrics and score files.

4. Defining preset contents

SYNTAX :

```
CHANNEL channelname = [1...16]
VAR $intvarname      = [0...127]      // up to 128 int vars
VAR $boolvarname    = [true/false] // up to 256 bool vars
VAR $stringvarname = "any string" // up to 256 string vars

INIT_TINYBOX          = ... // single command,
                       // or list of commands between curly braces
INIT_SONG song       = ...
INIT_BANK bank       = ...

PRESET preset        = ...
PRESET_RELEASE preset = ...

EFFECT_ON effect     = ...
EFFECT_OFF effect    = ...

TRIGGER_CLICK trigger = ...
TRIGGER_RELEASE trigger = ...

SWEEP sweep          = ... // continuous control command(s)
```

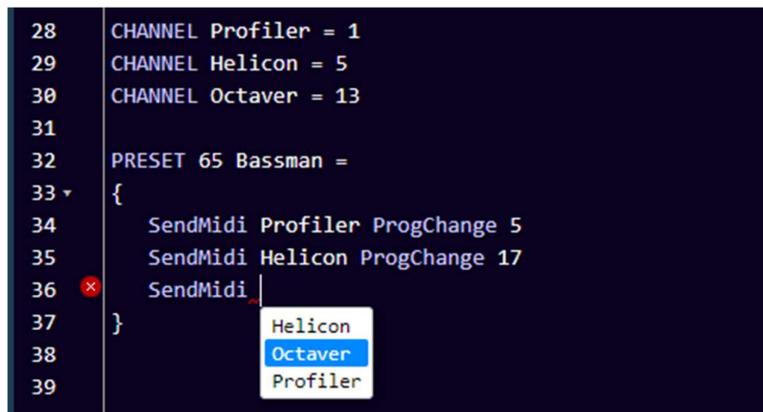
Now starts the main part of the setup, containing all details of which MIDI commands each preset will send. The next sub-topics cover each of the setup parts listed above. The actual format of the commands which will be contained in each of the preset definitions is described in later chapters.

4.1. Defining MIDI channels

MIDI commands can be sent on 16 different channels. Typically each device in the MIDI chain will listen to its specific channel, allowing you to control multiple devices simultaneously. In order to ease the setup, and also to make the setup more readable, each used MIDI channel in a TinyBox setup is given a name. Those MIDI channel definitions have to be the first instructions in this part of the setup.

Once you have specified a name for all MIDI channels you intend to use, the setup editor will show a dropdown box with those names each time a MIDI channel needs to be specified :

```
28 CHANNEL Profiler = 1
29 CHANNEL Helicon = 5
30 CHANNEL Octaver = 13
31
32 PRESET 65 Bassman =
33 {
34   SendMidi Profiler ProgChange 5
35   SendMidi Helicon ProgChange 17
36   SendMidi
37 }
38
39
```

A screenshot of a code editor with a dark background. The code defines MIDI channels and a preset. Lines 28-30 define channels for 'Profiler', 'Helicon', and 'Octaver'. Line 32 starts a preset named 'Bassman'. Lines 33-37 show a block of code for the preset, including 'SendMidi' commands for 'Profiler' and 'Helicon', and an incomplete 'SendMidi' command on line 36. A dropdown menu is open below line 36, showing the names 'Helicon', 'Octaver', and 'Profiler' as options.

A big advantage of this approach is also that you can very easily modify the MIDI channel on which a certain device is listening. Just adapt the MIDI channel number in the channel definition, and the new channel will be used in all MIDI commands for that device throughout your setup.

Attention :

Defining MIDI channels this way at the start of your setup is not only useful, it is also required. You will not be able to specify the MIDI commands to be sent by each preset as long as you haven't named the MIDI channels to be used.

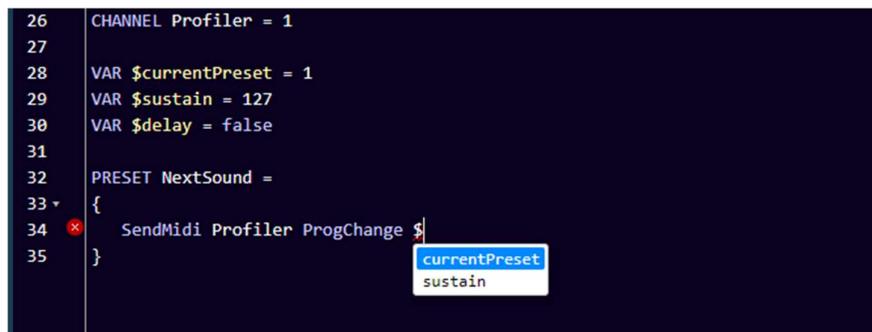
4.2. Defining data variables

The use of data variables in your setup is definitely an optional advanced feature. As long as you don't intend to use this you can safely skip this part of the documentation.

Right after specifying the used MIDI channels as described in the previous topic, you now (optionally) specify each of the data variables you intend to use. A variable name always starts with '\$'. Give the variable an initial value, this way the TinyBox setup compiler can detect which data format the variable will contain: a numeric value (between 0 and 127), a boolean value (true or false), or a string value (any text between double quotes) :

```
VAR $currentPreset = 56
VAR $delay = false
VAR $currentSong = "Go with the flow"
```

A numeric variable can be widely used in the setup: as it can contain any value between 0 and 127, it can directly replace a "hardcoded" value as part of any MIDI message. Whenever you would normally type a value between 0 and 127, you can type '\$' and the editor will propose a dropdown list with all available numeric data variables :



```
26 CHANNEL Profiler = 1
27
28 VAR $currentPreset = 1
29 VAR $sustain = 127
30 VAR $delay = false
31
32 PRESET NextSound =
33 {
34 SendMidi Profiler ProgChange $
35 }
```

We will go into more detail about the possibilities when covering the "variable commands" and "conditional commands" later on in this manual.

There is one predefined numeric variable, with the name **\$MidiChannel**

You can use this variable in any command which requires a MIDI channel, for instance :

```
SendMidi $MidiChannel ProgChange 13
```

By modifying the value of the MidiChannel variable you can target different devices with the same command set. In one bank you could for instance program 8 switches to select a preset, and 2 switches to specify on which sound module you want to activate that preset, resulting in 16 different preset selections with only 10 switches.

4.3. Defining the TinyBox initial state

You might want to initialize some global settings right after powering up the TinyBox. A typical example for this could be for instance the default behavior of the FCB1010 expression pedals, the initial settings for the built-in MIDI filter/router, or the `UseKeyboardControl` command to use a keyboard instead of FCB1010 as TinyBox remote control. These topics will be discussed in more detail further on when explaining the corresponding commands. For now it is sufficient to know that the commands to run during TinyBox initialization can be specified using following syntax :

```
INIT_TINYBOX =
{
    // any command to set an initial state
    // ...
}
```

It is of course also possible to specify MIDI commands here, to be sent for initializing the different devices connected to the TinyBox. It will be clear that in that case it is necessary to power the TinyBox as last component in your rig, or else to do a quick power cycle of the TinyBox once all other gear is connected and up and running.

4.4. Defining song or bank initialization

It is possible to send a set of MIDI messages each time a certain bank or song is being selected, that is when a click on the FCB1010 Up or Down switch selects the new bank or song.

If for instance you would have a backing track or clicktrack running for each song, the song initialization messages could contain a command to specify the tempo of the MIDI Clock which the TinyBox can send. You can definitely think of many other examples. Again, the actual commands which can be used are described in more detail in the next chapters, here we just indicate the format for specifying song or bank initialization commands :

```
INIT_SONG Go with the flow =
{
    // any song initialization command
    // ...
}

INIT_BANK looper =
{
    // any bank initialization command
    // ...
}
```

4.5. Defining the preset contents

In many cases a preset can be very simple: a single MIDI ProgChange command can be used to select a certain patch or sound in an effects module for instance. In this case the preset content can be described on a single line in the setup :

```
PRESET Vox Ac-30 = SendMidi Profiler ProgChange 2
```

In other cases a preset can be more complex, sending multiple MIDI commands to multiple devices. In that case those commands are specified on multiple lines, surrounded with curly braces:

```
PRESET 65_Bassman =  
{  
    SendMidi Profiler ProgChange 5  
    SendMidi Helicon ProgChange 17  
    SendMidi Octaver CtrlChange 13 127  
}
```

Do the same for each preset in the preset list. If you don't specify any content for a preset, the setup compiler will not complain, but obviously nothing will happen when selecting that preset using the FCB1010 preset switches.

Optionally a preset can also send MIDI messages on switch release, although this behavior is more appropriate for "triggers", which are designed for exactly that (see below). If you want a preset to send messages on switch release, use following syntax :

```
PRESET_RELEASE Vox Ac-30 = SendMidi Profiler CtrlChange 23 100
```

4.6. Defining the effect contents

As explained in an earlier chapter, the main difference between effects and presets is that effects can toggle between 2 states: ON and OFF. For each of those 2 states you will need to specify the MIDI command(s) to be sent. Again, in many cases sending one single MIDI command will be sufficient to (de)activate an effect, but you can go as complex as you desire, using following syntax :

```
EFFECT_ON Chorus = SendMidi Strymon CtrlChange 3 127  
  
EFFECT_OFF Chorus = SendMidi Strymon CtrlChange 3 0  
  
EFFECT_ON Delay =  
{  
    // any combination of multiple commands  
}  
  
EFFECT_OFF Delay =  
{  
    // any combination of multiple commands  
}
```

4.7. Defining the trigger contents

A trigger can be used to send one or multiple MIDI commands on the click of a footswitch. The syntax again supports both a one-line or a multiline content definition :

```
TRIGGER_CLICK Overdub = SendMidi Looper NoteOn 15 127

TRIGGER_CLICK HalfSpeed =
{
  // any combination of multiple MIDI commands
}
```

As explained in an earlier chapter a “trigger” can also be used to define a momentary effect, which sends a different message on switch press and on switch release. Both messages (or message groups) can be specified as follows :

```
TRIGGER_CLICK OctaveUp = SendMidi Octaver CtrlChange 15 127

TRIGGER_RELEASE OctaveUp = SendMidi Octaver CtrlChange 15 0

TRIGGER_CLICK PlayAmin7Chord =
{
  SendMidi Synth NoteOn 57 100
  SendMidi Synth NoteOn 60 100
  SendMidi Synth NoteOn 64 100
  SendMidi Synth NoteOn 67 100
  SendMidi Synth NoteOn 69 110
}

TRIGGER_RELEASE PlayAmin7Chord =
{
  SendMidi Synth NoteOff 57 127
  SendMidi Synth NoteOff 60 127
  SendMidi Synth NoteOff 64 127
  SendMidi Synth NoteOff 67 127
  SendMidi Synth NoteOff 69 127
}
```

4.8. Defining the sweep contents

A “sweep” might be a less intuitive concept in the TinyBox setup architecture. It’s a way to describe the functionality of an expression pedal. You first list the different required functionalities (volume, wah, expression, ...) as the possible “sweeps” in your setup. In the sweep definition (see syntax below) you specify the MIDI commands necessary for each of those functions. Once you have specified this, a simple command can link any of the sweeps to one of the two FCB1010 expression pedals. Such a pedal assignment command (see chapter 5.1) can be added to any preset or effect. This allows for a very flexible use of the expression pedals: they can serve a different function depending on the current context.

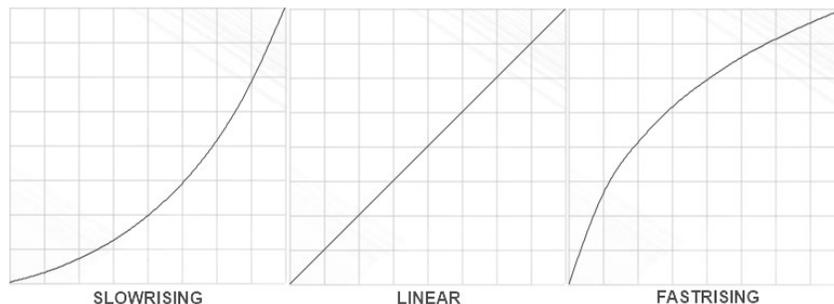
The syntax for defining sweep contents will look familiar by now :

```
SWEEP whammy = SendMidi DX7 PitchBend 0-127

SWEEP volume =
{
    SendMidi Profiler CtrlChange 7 0-127 SlowRising
    SendMidi Strymon CtrlChange 7 40-100 SlowRising
}
```

The sweep definition can only contain “continuous control commands”: these are the MIDI commands used for continuous parameter adjustment: “CtrlChange”, “PitchBend” or “ChannelPressure”.

By default the sweep curve when moving an expression pedal will be linear. However you can also specify that a sweep needs to be “SlowRising” or “FastRising”. An analog volume pedal for instance typically uses a “log taper” (sometimes called an “audio taper”). This taper increases the volume more slowly at the beginning of the pedal movement, and more steeply at the end. You can mimic this behavior with your digital FCB1010 expression pedal by specifying a “SlowRising” behavior for this sweep:



5. The command set

In the previous chapter we described the syntax for specifying the content of a preset, effect, sweep, etc. Now we will go in more detail about the actual commands which can be used in those preset contents. Obviously the MIDI command will be the most used command type, probably more than 90% of your setup will consist of MIDI commands. However the TinyBox offers much more than just the basic MIDI functionality of a regular FCB1010. The next sub-chapters handle each of the different command types supported in a TinyBox setup :

- Switch and pedal assignment commands
- Effect and relay activation commands
- MIDI commands
- Delay command
- Continuous control commands
- Filter/router commands
- Variable commands
- Conditional commands
- UseKeyboardControl

5.1. Switch and pedal assignment commands

SYNTAX :

```
Footswitch [1...10] = preset/effect/trigger-name/"nothing"  
Tipswitch  [1...2]  = preset/effect/trigger-name/"nothing"  
Heelswitch [1...2]  = preset/effect/trigger-name/"nothing"  
Pedal      [1...2]  = sweep-name/"nothing"
```

In general presets are assigned to FCB1010 footswitches through the bank setup, which was discussed in a previous chapter. For each bank you specify which preset (or effect or trigger) is assigned to each of the 10 footswitches. However it is possible to temporarily modify this predefined bank layout, by adding switch assignment commands to a preset content. This way you can create a very “dynamic” bank layout, with for instance a bottom row containing 5 presets and a top row containing up to 5 effects which can be different depending on the selected preset :

```
PRESET Chieftain =  
{  
    Footswitch 6 = BOOST  
    Footswitch 7 = Compressor  
    Footswitch 8 = Fxloop  
    // MIDI commands ...  
}  
  
PRESET Matchless =  
{  
    Footswitch 6 = Delay  
    Footswitch 7 = Flanger  
    Footswitch 8 = Chorus  
    // MIDI commands ...  
}
```

Pedal assignments are necessary in order to define what the function is for each of the 2 expression pedals. If you want to have a fixed pedal setup (for instance : left pedal = volume, right pedal = wah) you can specify the pedal assignment in the TinyBox initialization mentioned before :

```
INIT_TINYBOX =  
{  
    Pedal 1 = Volume  
    Pedal 2 = Wah  
}
```

“Volume” and “Wah” in this example are sweeps defined earlier in the setup. In order to disable pedal2 simply specify `Pedal 2 = nothing`

Of course, just like with switch assignments, also pedal assignment commands can be part of any preset content. This way the pedal behavior can be very dynamic, changing along with the currently selected preset.

Next to the 10 footswitches, you can also assign a preset, effect or trigger to 2 virtual “tip switches” and “heel switches”, one for each expression pedal. An optional but very powerful feature. You might have seen professional expression pedals which have a footswitch mounted underneath. The pedals behave like regular volume or expression pedals, but when pushing all the way down you can also engage the underlying switch in order to trigger some change. Those pedals typically require 2 separate jack cables running to your gear. With the TinyBox you can obtain this same behavior with the regular FCB1010 expression pedals!

Once you assign a preset or effect to “virtual tip switch 1”, you can trigger that preset by pushing expression pedal 1 all the way down. You could for instance link an effect to the tip switch which toggles the expression pedal behavior between 2 functions. No need to sacrifice a separate footswitch to change pedal behavior, you can do it with the pedal itself!

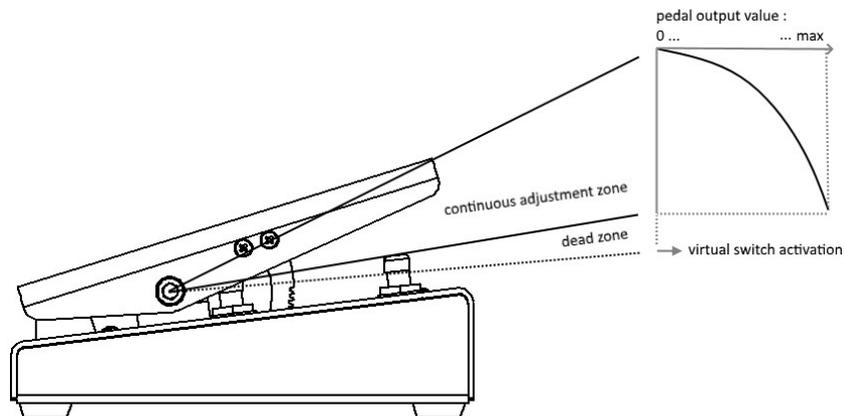
Similarly a “virtual heel switch” can be defined. This one is triggered by moving the expression pedal all the way up (heel-down). Functionality which probably doesn’t even exist with “real” footswitches. A great example of how to make use of this feature is the following setup fragment, which automatically activates the tuner as soon as you turn the volume of your guitar down to 0 :

```
INIT_TINYBOX =
{
    Pedal 1 = volume
    Heelswitch 1 = Tuner
}

TRIGGER_CLICK Tuner = SendMidi Eventide CtrlChange 99 127
TRIGGER_RELEASE Tuner = SendMidi Eventide CtrlChange 99 0
```

Remark :

When specifying a tip or heel switch, the TinyBox will automatically introduce a small “dead zone” in the pedal range, so that you can use the full adjustment range of the pedal without automatically triggering the tip (or heel) switch. In this case it is very helpful to add some resistance to the pedal at a point between full adjustment range and switch engagement. This can be done by sticking a piece of foam underneath the pedal, or even by mounting a real (unconnected) footswitch.



5.2. Effect activation, relay activation and expressionpedal activation commands

SYNTAX :

```
SwitchOn effectname
SwitchOff effectname
SendTrigger triggername
Close Relay1
Open Relay1
Close Relay2
Open Relay2
SendPedal 1
...
SendPedal 4
```

Some straightforward yet powerful commands : when selecting a certain preset you can activate a number of effects along with it. If you have a footswitch assigned to an effect, activating the effect with this command will also switch the footswitch LED on in order to show the current effect state.

Next to these effect activation commands there is also a trigger activation command which can be used to send all messages of a specific trigger whenever you select a certain preset.

The “Open Relay” and “Close Relay” commands are self explanatory. The FCB1010 contains 2 jack outputs which are controlled by a relay. These can be used for instance to control an amp which does not have any MIDI capability.

A “SendPedal” command can be used to resend the command linked to the current position of an expression pedal. For instance on some devices it might be necessary to set the current volume after changing the active preset. Sending this command avoids that you need to move the volume pedal a little in order to restore the previously active volume setting.

Remark :

When using the SwitchOn, SwitchOff or SendTrigger commands in an effect or trigger content, you risk to create an “infinite loop”. The simplest example is when you would specify :

```
EFFECT_ON effect 1 = SwitchOn effect 1
```

It is obvious that this line of code would cause the TinyBox to lock up, with the same command being executed over and over again. This kind of loop is not always as easy to detect as in the example above. It can be caused by a much longer “chain” of effect activation commands, finally ending up in an effect “activating itself” again through the activation of other effects or triggers. Luckily enough the TinyBox editor analyzes your setup while you are typing the effect activation commands, and the code compiler will give an error message when an infinite loop is being detected.

5.3. Navigation commands

SYNTAX :

```
GotoBank bankname  
GotoSong songname
```

Use these commands to directly jump to a specific bank or song, instead of using the up/down switches to scroll through the bank list or setlist.

When working with setlists and songs, you should always use the GotoSong command, while the GotoBank command can be used when you don't specify any setlist.

5.4. MIDI commands

SYNTAX :

```
SendMidi channelname ProgChange value
SendMidi channelname CtrlChange value value
SendMidi channelname NoteOn      value value
SendMidi channelname NoteOff     value value
SendSysEx      F0 ... F7
SendSysCommon SongSelect value
SendSysCommon SongPosition 14-bit-value
SendSysCommon TuneRequest
SendRealtime  MIDIStart
SendRealtime  MIDIContinue
SendRealtime  MIDIStop
SendRealtime  MIDIClock [10...250] BPM
SendRealtime  SystemReset
```

As the TinyBox is in the first place a MIDI controller, “MIDI command” will without doubt be the most used command type in a TinyBox setup. One of the TinyBox strengths is the fact that you can combine any number of MIDI messages, sent on different MIDI channels, into one single preset. All these messages will be sent simultaneously with one single foot click.

The syntax above is self explanatory: *channelname* is any of the MIDI channel names specified at the start of the setup (see chapter 4.1), *value* is any numeric value between 0 and 127. As mentioned before you can also use a numeric variable instead of specifying a value in any MIDI message (even in the content of a SysEx message!)

In principle the length of a SysEx message is unlimited, however in a realistic use case this command will be used to send rather short SysEx messages for effect or preset parameter tweaking, not entire patchdumps. The values are written in hexadecimal notation, starting with F0, ending with F7, and with values 00-7F in between the start and stop bytes.

The command **SendRealtime MIDIClock nnn BPM** does not send one single command. Instead it starts a continuous stream of MIDIClock messages at the given tempo (in beats per minute). The MIDIClock stream is stopped by calling the command **SendRealtime MIDIClock 0 BPM**

5.5. Continuous Control commands

SYNTAX :

```
SendMidi channel CtrlChange      value from-to [SlowRising/FastRising]
SendMidi channel PitchBend      value from-to [SlowRising/FastRising]
SendMidi channel ChannelPressure value from-to [SlowRising/FastRising]
SendSysEx F0 ... value ... F7
```

“Continuous control” commands are used for specifying a sweep content. They describe which messages will be sent when moving one of the FCB1010 expression pedals.

With most (if not all) MIDI controllers expression pedals can send specific ControlChange messages. The most commonly used values for continuous control are :

- CC 07 = volume
- CC 04 = foot controller
- CC 01 = modulation wheel
- CC 11 = expression controller

The transmitted value range for those MIDI messages is 0 (heel down) to 127 (tip down).

With the TinyBox you got some powerful extra options which you will probably not find in any other MIDI controller :

- next to CtrlChange messages it is also possible to send PitchBend or ChannelPressure messages, and even SysEx messages!
- the sweep range is fully customizable
- you can specify the sweep curve which the pedal should follow : linear, slow rising or fast rising (see chapter 4.8 for more details)
- it is possible for 1 pedal to send multiple different MIDI messages on different channels. This is done by adding multiple continuous control commands to the sweep content.

When using the SendSysEx command, you can specify any length of SysEx message, while inserting the word “value” at any place within the SysEx. The current position of the expression pedal (value between 0 and 127) will be inserted in the SysEx message at the position of the word “value”.

5.6. Delay command

SYNTAX :

```
Wait value
```

This command can be added in between MIDI commands in order to add a small pause between the commands. Some gear has proven to react unreliably if multiple MIDI commands are sent to it at full speed. Another use case could be to play short MIDI samples when clicking a footswitch, by specifying a series of NoteOn/NoteOff messages, separated by the necessary delay commands to hold each chord for the required duration. Although we must admit that would require some tedious programming...

The delay value can be anything between 1 and 127, and is expressed in 100ms units. This means that a `Wait 10` command will introduce a 1 second delay, and the maximum possible delay is 12.7 seconds. Be aware that such delay is “blocking”: the TinyBox cannot process any switch press or pedal movement or do anything else while it is executing the `Wait` command.

5.7. Filter/router commands

SYNTAX :

```
[BlockMidi/AllowMidi] ActiveSense
[BlockMidi/AllowMidi] [SysEx/SystemRealtime/SystemCommon]
[BlockMidi/AllowMidi] All

BlockChannels      channelrange
BlockChannel       channelname
MoveChannel        channelname to channelname
CopyChannel        channelname to channelname
ResetChannelRouting channelname

BlockNotes         channelname from-till
MoveNotes          channelname from-till to channelname [from]
CopyNotes          channelname from-till to channelname [from]
ResetNoteRouting   channelname

BlockCC            channelname ccnr
CopyCC             channelname ccnr to channelname [ccnr]
MoveCC            channelname ccnr to channelname [ccnr]
ResetCCRouting    channelname ccnr

ResetMidiFilter

ModifyVelocity from in_from-in_till to out_from-out_till
```

Apart from being a very powerful MIDI controller (and many other things, as described in the introduction of this manual), the TinyBox also contains a MIDI filter + MIDI router. This filter is placed between the MIDI IN and MIDI OUT connectors, so all MIDI messages coming in through the MIDI IN connector run through the MIDI filter/router before being forwarded to the MIDI OUT connector.

This is very specific functionality, only relevant when you have the TinyBox wired in the middle of a larger MIDI chain, with “upstream” devices connected to the TinyBox MIDI IN. Therefore many of you will not intend to use the filter capabilities and can safely skip this chapter.

5.7.1. Filtering certain MIDI message types

You can filter out certain MIDI message types which can cause quite some traffic on the MIDI chain, like the MIDI ActiveSense messages sent every 250ms by some devices. The commands for this are

```
BlockMidi ActiveSense // blocks FE MIDI messages
BlockMidi SystemRealtime // blocks F8,FA,FB,FC MIDI messages
BlockMidi SystemCommon // blocks F1,F2,F3,F6 MIDI messages
BlockMidi SysEx // blocks [F0...F7] MIDI SysEx messages
BlockMidi All // blocks ALL MIDI messages (*)
```

(*) “BlockMidi All” stops all MIDI forwarding from MIDI IN to MIDI OUT port. This can be necessary in some specific MIDI routing configurations, where forwarding the MIDI from MIDI IN to MIDI OUT might cause a MIDI loop.

The blocking can be undone by sending the corresponding `AllowMidi` command.

5.7.2. Filtering certain MIDI channels

A specific MIDI channel can be filtered out with the command

```
BlockChannel channelname
```

With this command you can make sure that the device listening to that specific MIDI channel will only be controlled by the TinyBox, and not by any other MIDI device connected to the TinyBox MIDI IN.

You can also block a whole range of MIDI channels, using the channel numbers 1-16 instead of channel names. A range is specified between square brackets, and can combine several number sequences like this :

```
BlockChannels [11-16] // blocks all channels from 11 till 16
BlockChannels [11,16] // only blocks channels 11 and 16
BlockChannels [1-4,10,13-16] // a combination of ranges
```

5.7.3. Routing MIDI channels

The MIDI processing block between MIDI IN and MIDI OUT connectors is called a “filter/router” : it not only *filters out* certain MIDI message types or channels, but it can also *route* one MIDI channel to another. If for instance you have one master keyboard which you want to use to play several synth modules, you can copy the MIDI messages sent on one channel to a second channel in order to play both synths at the same time. If instead you want to switch from one synth to another you can do that by moving the incoming messages of the master keyboard to a different channel. The messages to setup such channel routing are :

```
MoveChannel channelname1 to channelname2  
CopyChannel channelname1 to channelname2
```

This channel routing (or the channel filtering of previous chapter) can be undone with the command

```
ResetChannelRouting
```

5.7.4. Filtering or routing MIDI notes

This functionality can be of great use for keyboard players. It can turn a very simple MIDI keyboard into a powerful master keyboard with several zones controlling several synth modules, adding transpose functionality, and so on.

```
BlockNotes channelname from-till
```

This command blocks a certain zone of the keyboard. The range is not expressed using numeric values but using notes on the keyboard, ranging from C-1 to G9

```
MoveNotes channelname from-till to channelname [from]  
CopyNotes channelname from-till to channelname [from]
```

These commands are without doubt more interesting. They allow to move a certain zone of the keyboard to a different channel, thus playing one synth module with one hand, and another synth module with the other hand. On top of that you can specify a new “from” value for the moved or copied range, which results in transposing the played notes.

A sample syntax is : `MoveNotes synth1 C4-G9 to synth2 C3`

The MIDI note filtering or routing can be undone with the command

```
ResetNoteRouting
```

Another MIDI note related command allows you to modify the touch sensitivity of your keyboard. As you will know each MIDI NoteOn/NoteOff command contains a velocity value. Some velocity sensitive keyboards don't have the optimal sensitivity for certain sounds. In that case the ModifyVelocity command allows you to extend or compress the velocity range.

A sample syntax is : **ModifyVelocity from 20-100 to 40-127**

In the example above incoming velocity values lower than 20 will translate to velocity 40, and incoming values above 100 will translate to the maximum velocity of 127. Between those values the velocity range will be expanded linearly from 20-100 to 40-127.

Velocity mapping can be undone by specifying **ModifyVelocity from 0-127 to 0-127**

Remark: no matter which ranges are specified, an incoming velocity value of 0 will never be changed to a non-zero value. This is important, because many keyboards send a NoteOn message with velocity 0 to actually indicate a Note Off event. Modifying this value would result in hanging notes.

5.7.5. Filtering or routing MIDI ControlChange messages

Very similar to the commands discussed in the previous topics, a number of filter commands allow to block specific ControlChange numbers, or copy or move them to another MIDI channel. It is even possible to let the MIDI router change the ControlChange number. As a result any MIDI CC message can be turned into any other CC message. This allows you for instance to extend the FCB1010 with another very simple MIDI footcontroller, connected to the FCB1010 MIDI IN. Even if that second controller is limited to sending a few fixed CC messages on a fixed MIDI channel, the MIDI router can route those messages to anything else. Since the router could be reconfigured easily using one of the commands below, you can turn the limited controller into a very flexible one.

The ControlChange router commands are :

```
BlockCC channelname ccnr  
MoveCC channelname ccnr1 to channelname [ccnr2]  
CopyCC channelname ccnr1 to channelname [ccnr2]
```

The ControlChange routing can be undone with the command

```
ResetCCRouting
```

5.7.6. Resetting the filter/router

All of the router settings mentioned above can be reset with the command

```
ResetMidiFilter
```

This resets all MIDI blocking, channel routings, ControlChange routings and Note routings.

5.8. Variable commands

We have mentioned in a previous chapter how you can define numeric, boolean or string variables in your setup. Of course the use of variables only makes sense when you have the possibility to change their values when selecting a certain preset or effect, and then to react upon those changed values. You can do that with the variable commands of this chapter and the conditional commands of the next chapter.

SYNTAX :

```
$intvarname = value  
$intvarname = $intvarname2  
$intvarname = $intvarname2 [+ -] value  
$intvarname [++ --]  
$intvarname [+= -=] value  
$boolvarname = [true/false]  
$boolvarname = $boolvarname2  
$boolvarname = !$boolvarname2  
$stringvarname = "any string"
```

A numeric variable can be set to any value between 0 and 127. It can be incremented (++) or decremented (--), a fixed value can be added (+=) or subtracted (-=), or the value of another numeric variable can be taken and modified (= \$intvarname2 +/- value)

A boolean variable can be set to true or false, can take over the value of another boolean variable, or its value can be inverted from true to false and vice versa (using the ! symbol)

A string variable can simply be given any text value. The variable content can then be checked using one of the conditional commands of the next chapter.

5.9. Conditional commands

SYNTAX :

```
if (condition*) {
    ...
}
else if (condition*) {
    ...
}
else {
    ...
}

while(condition*) {
    ...
}

* condition : $intvarname      [ > >= == != <= < ] [0...127]
               $intvarname      [ > >= == != <= < ] $intvarname2
               $stringvarname    [ == != ]          "any string"
               $boolvarname      [ == != ]          $boolvarname2
               $boolvarname
               !$boolvarname

switch(&intvarname) {
    case[0...127]:
        ...
        break
    ...
    default:
        ...
        break
}

switch(&stringvarname) {
    case "any string":
        ...
        break
    ...
    default:
        ...
        break
}
```

You can make your setup very “dynamic” by having the same preset select a different sound or activate a different effect, depending on some condition. Data variables are used to set those conditions, and the conditional commands above are used to make the necessary decisions.

The ‘while’ statement can be used to create loops, for instance in order to do a fade-in/fade-out effect (see “Example 5” in a previous chapter for an example of this)

Of course the while statement needs to be used with care, making sure that you don’t create an infinite loop in your setup. This would make the unit unresponsive.

5.9.1. The condition syntax

All conditional commands check the value of a data variable to see if a certain *condition* is true. The possible checks are :

- for numeric variables :

```
$var > nn      // is bigger than
$var >= nn     // is bigger than or equal to
$var == nn    // is equal to
$var != nn    // is not equal to
$var <= nn    // is less than or equal to
$var < nn     // is less than
              // nn being a value between 0 and 127,
              // or another numeric variable
```

- for string variables :

```
$var == "any string"
$var != "any string"
```

- for boolean variables :

```
$var1 == $var2 // $var1 and $var2 are both true or false
$var1 != $var2 // $var1 is different from $var2
$var          // $var is true
!$var         // $var is false
```

If needed multiple conditions can be combined:

(`&&` means "and", `||` means "or")

```
// all of the following conditions need to be true :
((condition1) && (condition2) && ... )

// at least one of the following conditions needs to be true :
((condition1) || (condition2) || ... )
```

5.9.2. if...then...else statements

The syntax for an if...then...else... type of check is

```
if (condition) {
    // any number of commands...
}
else if (another condition) {
    // any number of commands...
}
else if (yet another condition) {
    // any number of commands...
}
else {
    // if none of the conditions apply
    // execute the commands in this segment
}
```

You can even have nested conditional statements (although we don't think a TinyBox setup will require that amount of complexity...):

```
if (condition) {
    if (subcondition) {
        ...
    }
    else {
        ...
    }
}
else {
    ...
}
```

If you prefer you can also put all curly braces on a new line for clarity :

```
if (condition)
{
    // any number of commands...
}
```

On the other hand the curly braces are not strictly needed if they are surrounding only 1 command :

```
if ($delay)
    SendMidi MyGear CtrlChange 112 127
else
    SendMidi MyGear CtrlChange 112 0
```

5.9.3. while statement

The while statement has an identical syntax as the if statement :

```
while (condition) {  
    // any number of commands..  
}
```

5.9.4. switch statements

A switch statement is a shortcut for a long series of if statements. It is used to check a variable against a larger series of different possible values.

- for a numeric variable :

```
switch($delayValue) {  
    case 2:  
        // any number of commands  
        break  
    case 15:  
        // any number of commands  
        break  
    case 123:  
        // any number of commands  
        break  
    default:  
        // if none of the above values match  
        // execute the commands in this segment  
        break  
}
```

- for a string variable :

```
switch($songName) {  
    case "Go with the flow":  
        // any number of commands  
        break  
    case "No one knows":  
        // any number of commands  
        break  
    case "Do it again":  
        // any number of commands  
        break  
    default:  
        // if none of the above values match  
        // execute the commands in this segment  
        break  
}
```

5.9.5. conditions using predefined variables

The TinyBox programming language provides a few predefined variables, which can be used in conditional commands. Those predefined variables are :

```
#CURRENT_BANK
#CURRENT_SONG
#CURRENT_PRESET
EFFECT_ON effectname
EFFECT_OFF effectname
```

In the previous example about the switch statement for instance you could typically use those variables, instead of creating an own \$songName variable which you would have to fill with the currently selected song name yourself :

```
switch(#CURRENT_SONG) {
    ...
}

if(EFFECT_ON Delay) {
    ...
}
```

A small example of the conditional logic in action: with the few lines of code below you can program 2 footswitches to browse through all sounds of your synth or modeler :

```
119 CHANNEL Profiler = 1
120
121 VAR $currentPreset = 1
122
123 PRESET Next sound =
124 {
125   if($currentPreset < 100) {
126     $currentPreset++
127   }
128   else {
129     $currentPreset = 1
130   }
131   SendMidi Profiler ProgChange $currentPreset
132 }
133
134 PRESET Previous sound =
135 {
136   if($currentPreset > 1) {
137     $currentPreset--
138   }
139   else {
140     $currentPreset = 100
141   }
142   SendMidi Profiler ProgChange $currentPreset
143 }
```

5.10. Remote Control

Let's end with a few very specific commands which don't belong in any of the preceding command categories. Most of you will be using the FCB1010 along with the TinyBox and will not have any interest in using these commands.

SYNTAX :

```
REMOTECONROLCHANNEL = nn
```

When you add this command prior to the other MIDI channel definitions, you can remotely control the TinyBox from another MIDI device (through MIDI IN), or from a laptop connected through USB.

Following ControlChange and ProgramChange commands can be sent on the specified remote control MIDI channel to simulate a connected FCB1010 footcontroller :

```
ProgramChange 00-09 = click/release footswitch 1-10  
ProgramChange 14-15 = click/release Down/Up footswitch  
ControlChange 04/07 value = move left/right expr.pedal
```

SYNTAX :

```
UseKeyboardControl
```

This command provides an alternative way for remote control, which might be more convenient for keyboard players. When you add this command to the `INIT_TINYBOX` section of your setup, you can use a MIDI keyboard, connected to the TinyBox MIDI IN connector, as remote control for the TinyBox, instead of using the FCB1010 floorboard.

The FCB1010 with its specific "TinyBox slave" firmware sends a NoteOn/NoteOff command for each keypress, and a series of CC04 or CC07 commands when moving the 2 expression pedals. When adding the "UseKeyboardControl" command to your setup the TinyBox will also listen for such remote control messages on its MIDI IN input. More specifically, in this case the FCB1010 remote control commands are :

```
Channel 13 NoteOn/NoteOff 00-09 127 = click/release footswitch 1-10  
Channel 13 NoteOn/NoteOff 14/15 127 = click/release Down/Up footswitch  
Channel 13 ControlChange 04/07 value = move left/right expr.pedal
```

The appendix at the very end of this manual gives more info about the internal signal routing in the TinyBox, with specific topics about these remote control use cases. There you will see that when using the UseKeyboardControl command the incoming MIDI messages first run through the TinyBox “MIDI mapper” before being used as remote control trigger. This gives you the possibility to use the note routing commands explained in chapter 5.7.4 to convert the lowest octave of your keyboard to the required note range and MIDI channel for remote control. Any incoming note message which doesn’t serve as remote control command is just forwarded to the MIDI OUT connector, so of course all other octaves of the keyboard can be used for playing, while the lowest keys can be used for selecting presets or activating effects.

Here's an example note mapping configuration :

```
CHANNEL MySynth = 1
CHANNEL RemoteControl = 13

INIT_TINYBOX =
{
  UseKeyboardControl

  // NoteNumber 0-9 :
  MoveNotes MySynth C0-A0 to RemoteControl C-1

  // NoteNumber 14-15 :
  MoveNotes MySynth Bb0-B0 to RemoteControl D0
}
```

Although the FCB1010 has 10 preset switches next to the 2 bank up/down switches, it is not necessary to reserve 12 keys on the MIDI keyboard for remote control. If you wish you can limit it to as little as 3 keys for “bank up”, “bank down” and “select preset”. In that case you would create a setup with many banks but with only 1 preset in each bank.



Done ! Now it's time to have fun !



APPENDIX : TinyBox programming language reference

Comments :

```
// single-line comment : any text...

/*
  multi-line comment :
  any text...
*/
```

Defining presets, effects, triggers, sweeps, bank layout, songs and setlist :

```
PRESETS =
{
  [preset name]
  ...
}

EFFECTS =
{
  [effect name]
  ...
}

TRIGGERS =
{
  [trigger name]
  ...
}

SWEEPS =
{
  [continuous control name]
  ...
}

NO_UPDOWN_SWITCHES
USE_DIRECT_BANK
GLOBALSWITCH [1...10] = [presetname]

BANKS =
{
  [bank name] : [preset name] | [preset name] | ... | [preset name]
  [bank name] : [preset name] | [preset name] | ... | [preset name]
  ...
}

SONGS =
{
  [songname] : [bankname]
  ...
}

SETLIST =
{
  [songname]
  ...
}
```

Defining preset content :

```
CHANNEL channelname = [1...16]

VAR $intvarname      = [0...127]
VAR $boolvarname     = [true/false]
VAR $stringvarname   = "any string"

INIT_TINYBOX          = ... (single command, or list of commands between curly braces)
INIT_SONG song        = ... "
INIT_BANK bank        = ... "

PRESET preset         = ... "
EFFECT_ON effect      = ... "
EFFECT_OFF effect     = ... "
TRIGGER_CLICK trigger = ... "
TRIGGER_RELEASE trigger = ... "
SWEEP sweep           = ... " (contains only continuous control commands)
```

Commands :

Dynamic switch and pedal assignment commands :

```
Footswitch [1...10] = preset/effect/trigger-name/"nothing"
Tipswitch  [1...4]  = preset/effect/trigger-name/"nothing"
Heelswitch [1...4]  = preset/effect/trigger-name/"nothing"
Pedal      [1...4]  = sweep-name/"nothing"
```

Effect activation commands

```
SwitchOn   effectname
SwitchOff  effectname
SendTrigger triggername
SendPedal  [1...4]
```

MIDI Commands :

```
SendMidi channelname ProgChange value
SendMidi channelname CtrlChange value value
SendMidi channelname NoteOn   value value
SendMidi channelname NoteOff  value value
SendSysEx F0 ... F7
SendRealtime MIDISTart
SendRealtime MIDIContinue
SendRealtime MIDISTop
SendRealtime MIDIClock [10...250] BPM
```

Continuous control commands :

```
SendMidi channelname CtrlChange value [from-till] [FastRising/SlowRising]
SendMidi channelname PitchBend      [from-till] [FastRising/SlowRising]
SendMidi channelname ChannelPressure [from-till] [FastRising/SlowRising]
SendSysEx F0 ... value ... F7
```

Delay command :

```
Wait value (expressed in 100ms units)
```

Filter/Router commands :

```
[BlockMidi/AllowMidi] ActiveSense      -> FE
[BlockMidi/AllowMidi] SystemRealtime   -> F8/FA/FB/FC
[BlockMidi/AllowMidi] SystemCommon     -> F1/F2/F3/F6

BlockChannels      channelrange      (e.g.[1-16], [1-4,5,10-16], ...)
BlockChannel       channelname
MoveChannel        channelname to channelname
CopyChannel        channelname to channelname
ResetChannelRouting channelname

BlockNotes         channelname from-till  (from,till = C-1 -> G9)
MoveNotes          channelname from-till to channelname [from]
CopyNotes          channelname from-till to channelname [from]
ResetNoteRouting  channelname

BlockCC            channelname ccnr
CopyCC             channelname ccnr to channelname [ccnr]
MoveCC            channelname ccnr to channelname [ccnr]
ResetCCRouting    channelname ccnr

ResetMidiFilter

ModifyVelocity from from-till to from-till  (from,till = 0-127)
```

Variable Commands :

```
$intvarname = value
$intvarname = $intvarname2
$intvarname = $intvarname2 [+ -] value
$intvarname [++ --]
$intvarname [+ = - =] value
$boolvarname = [true/false]
$boolvarname = $boolvarname2
$boolvarname = !$boolvarname2
$stringvarname = "any string"
```

Conditional Commands :

```
if (condition*) { | switch(&intvarname) { | switch(&stringvarname) {
  ... | case[0..127]: | case "any string":
} | ... | ...
else if (condition*) { | break | break
  ... | ... | ...
} | default: | default:
else { | ... | ...
  ... | break | break
} | } | }

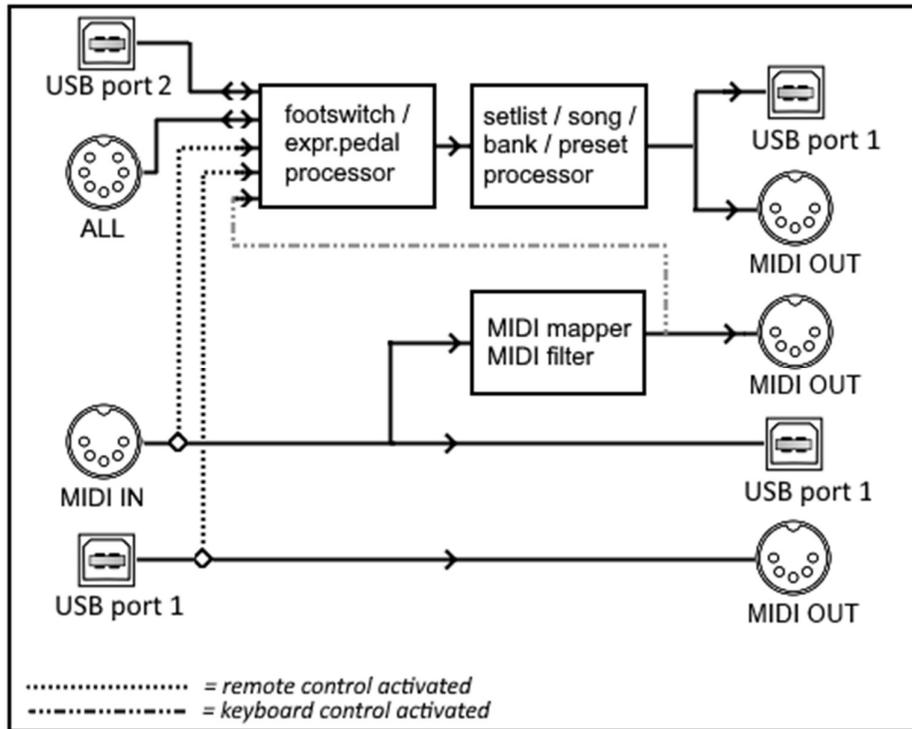
while (condition*) {
  ...
}
```

```
* condition : $intvarname [ > >= == != <= < ] [0..127]
              $intvarname [ > >= == != <= < ] $intvarname2
              $stringvarname [ == != ] "any string"
              $boolvarname [ == != ] $boolvarname2
              $boolvarname
              !$boolvarname
              #CURRENT_SONG [ == != ] songname
              #CURRENT_BANK [ == != ] bankname
              #CURRENT_preset [ == != ] presetname
              EFFECT_ON effectname
              EFFECT_OFF effectname
```

Global configuration commands :

```
UseKeyboardControl
```

APPENDIX : the TinyBox MIDI routings – a detailed rundown

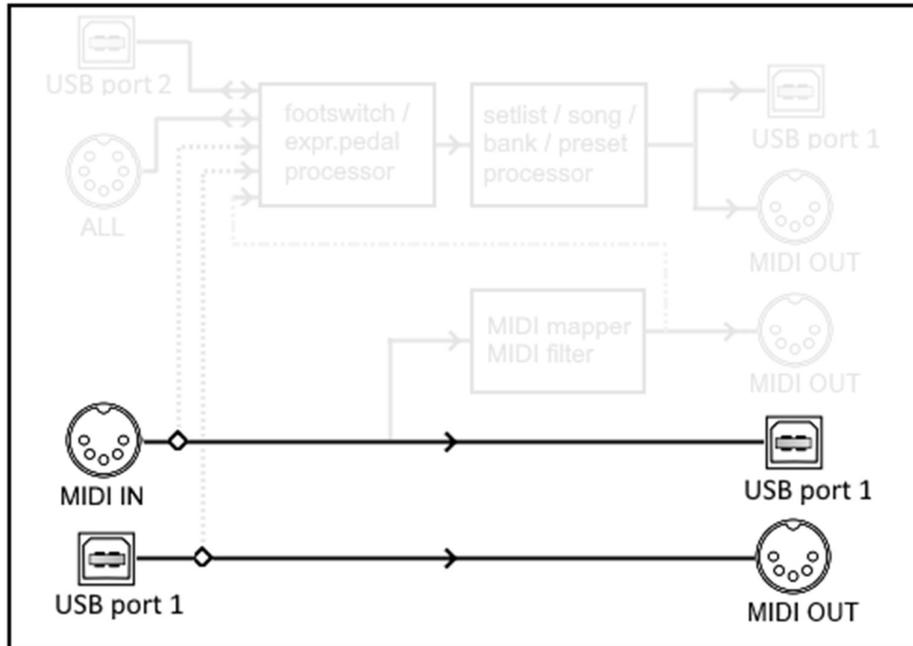


The TinyBox internal routing

If you look at the scheme above, the MIDI routing inside the TinyBox may seem quite complex. That is because the TinyBox is actually many things in 1 box. It will all become clear when we break down the scheme in the different pieces of TinyBox functionality, which we will do on the next pages.

Remark : in the schematic representations you will see several USB connectors labeled “USB port 1” or “USB port 2”. In reality these 2 MIDI-USB ports share the same physical USB connector. Actually, the TinyBox contains 3 MIDI-USB ports in total. Port 3 is a dedicated port for patchdumps and firmware updates, and is therefore not shown in the schematic representations of the TinyBox MIDI routing. Similarly you notice several MIDI OUT connectors on the pictures. Again this is actually 1 physical connector, repeated a few times in the schematic just to depict more clearly the different internal MIDI paths.

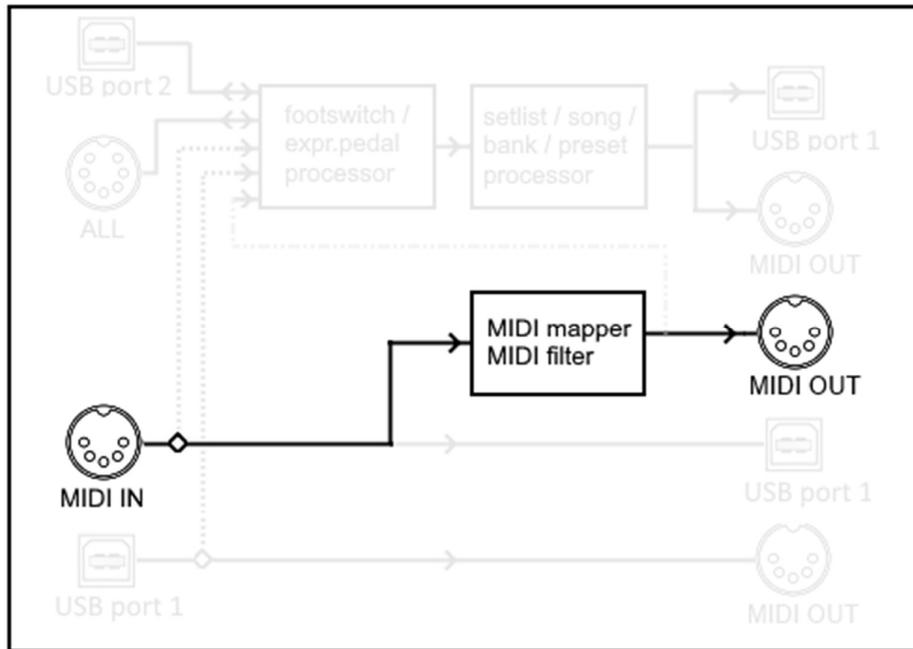
The TinyBox as a MIDI-USB interface



MIDI-USB interface

The most simple MIDI routing : all MIDI entering the TinyBox through the MIDI IN connector is forwarded to a connected computer through the MIDI-USB port labeled “TinyBox port 1”. All MIDI sent by the computer to that same MIDI-USB port is forwarded by the TinyBox to its MIDI OUT port.

The TinyBox as a MIDI mapper/filter

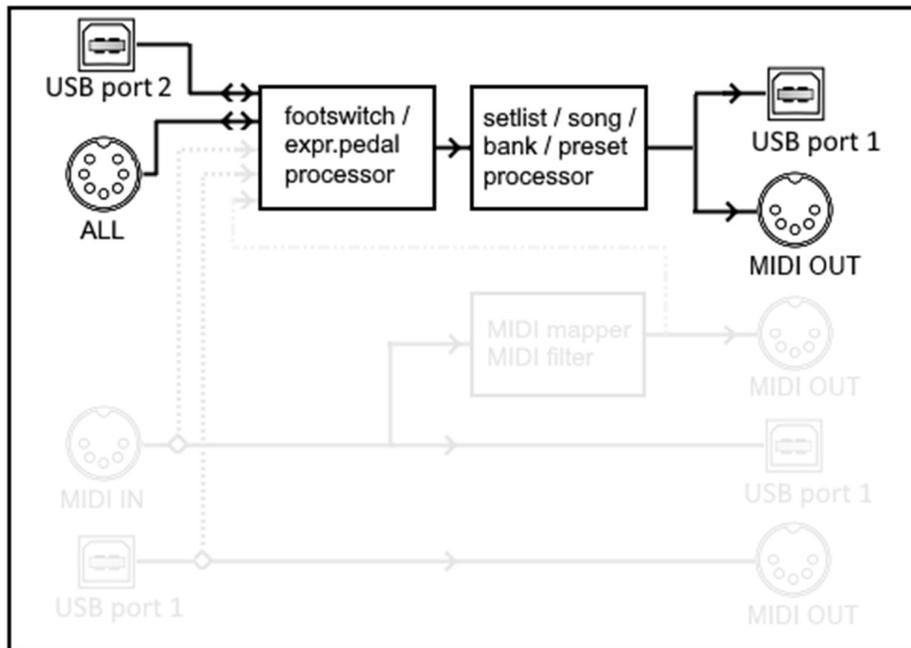


MIDI mapper/filter

All MIDI entering the TinyBox through the MIDI IN connector is forwarded to the MIDI OUT connector, but first runs through a programmable mapper/filter. You can transform the MIDI in many different ways: move the messages to a different MIDI channel, copy them to multiple channels, filter out certain MIDI message types or certain MIDI channels, transpose MIDI notes, etc.

The different commands which offer this functionality are discussed in detail in chapter 5.6 about filter/router commands.

The TinyBox as advanced MIDI controller with (virtual) FCB1010 floorboard



MIDI controller

When you connect a Behringer FCB1010 to the connector labeled “ALL”, you establish 2-way MIDI communication between the floorboard and the TinyBox (*). A click on the FCB1010 can trigger any complex preset, the up/down buttons let you scroll through a large set of banks or through a predefined setlist, at the same time the expression pedals can control many different parameters.

The MIDI messages sent by your presets are being sent both to the MIDI OUT connector and to MIDI-USB Port 1. This means that you can simultaneously control both hardware devices and software applications with a single click on the floorboard.

In this MIDI controller scenario the FCB1010 is turned into a dummy slave controller for the TinyBox by installing a specific firmware chip, included in the TinyBox purchase. This firmware communicates with the TinyBox using specific MIDI commands, which are listed in detail on the next page.

Additionally, the TinyBox server software gives you the option of displaying a “virtual” FCB1010 on an iPad, which shows the current status of your setup. This virtual FCB1010 can also be used as a wireless remote control for your TinyBox. To do so, the TinyBox server software, running on a connected laptop, communicates with the TinyBox through its dedicated MIDI-USB port 2.

(*) *Remark:* connecting the Behringer FCB1010 to the 7-pins MIDI connector of the TinyBox requires a “single cable kit” which adds an identical 7-pins connector to the FCB1010. This kit can be ordered along with the TinyBox.

FCB1010 to TinyBox :

MIDI Channel 13

NoteOn/NoteOff 0x00-0x09 0x7f = switch 1-10 press/release
NoteOn/NoteOff 0x0e/0x0f 0x7f = switch DOWN/UP press/release
ControlChange 0x04/0x07 value = expr.pedal A/B position

TinyBox to FCB1010 :

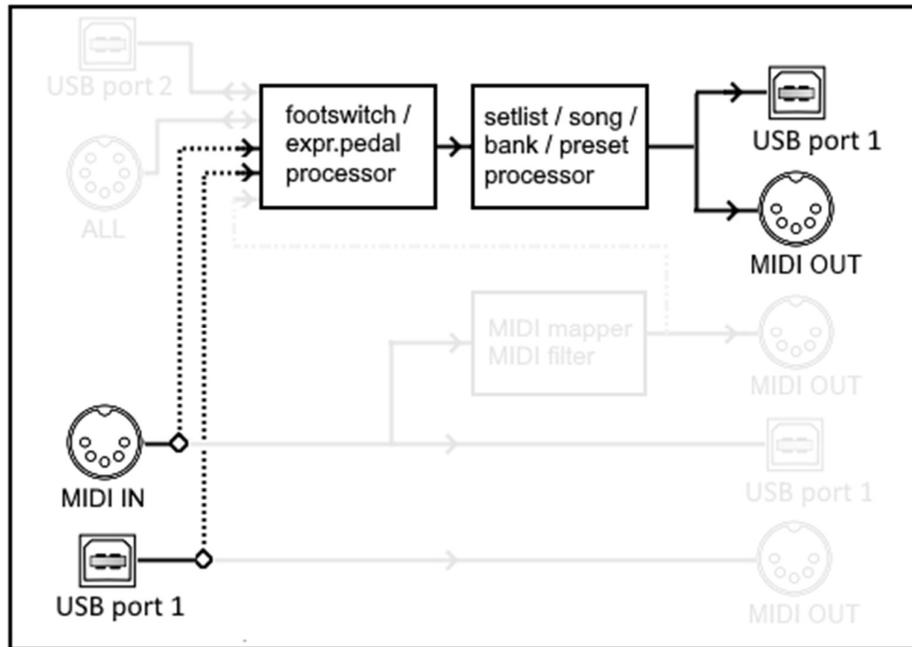
MIDI Channel 13

ControlChange 0x0c value 0x00-0x09 = LED ON for switch 1-10
value 0x0a-0x0b = LED ON for expr.pedal A/B
value 0x0c-0x16 = LED ON for fcb "menu" LEDs
value 0x18-0x19 = RELAY1/2 ON
value 0x20-0x29 = LED OFF for switch 1-10
value 0x2a-0x2b = LED OFF for expr.pedal A/B
value 0x2c-0x36 = LED OFF for fcb "menu" LEDs
value 0x38-0x39 = RELAY1/2 OFF
ControlChange 0x0d value 0x00-0x7f = DIGIT001 7-seg (*)
ControlChange 0x0e value 0x00-0x7f = DIGIT010 7-seg (*)
ControlChange 0x0f value 0x00-0x7f = DIGIT100 4-seg + points (**)
ControlChange 0x10 value 0x00-0x7f = show value 000-127 on display
ControlChange 0x11 value 0x00-0x47 = show value 128-199 on display
ControlChange 0x12 value 0x00-0x7f = as 0x10 but with '+' added
ControlChange 0x13 value 0x00-0x47 = as 0x11 but with '+' added

```
g
--
b| |f          (*) bit6..0 = g f e d c b a
-- --> a      (**) bit6..0 = dp3 dp2 dp1 '+' f e a
c| |e
--
d
```

Communication protocol FCB1010 - TinyBox

Remotely controlling the TinyBox



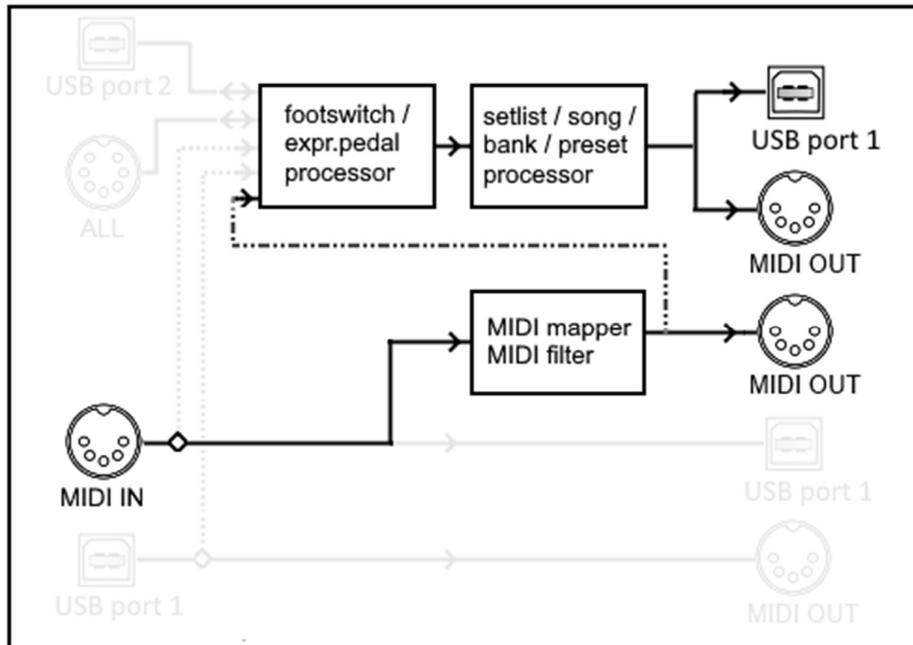
TinyBox remote control

Apart from (or next to) using the FCB1010 as foot controller, you can also remotely control the TinyBox from any MIDI hardware device (through MIDI IN) or software (through MIDI-USB port “TinyBox port 1”)

This is shown with a dashed line on the schematic above, because it is an option which is not activated by default. In order to activate this option, you need to specify which MIDI channel is used for this remote control, by adding the line `REMOTE_CONTROL_CHANNEL = nn` to your setup.

Once activated, you can remotely simulate a press on an FCB1010 footswitch by sending ProgChange 00-09 for switches 1-10 and ProgChange 14-15 for the Down/Up switches. The expression pedals can be simulated by sending a ControlChange message stream with CC number 04 (left) or 07 (right). You can directly jump to a certain bank by sending ControlChange 00 *nn* to go to bank *nn*.

Another remote control option : “UseKeyboardControl”



TinyBox keyboard control

This option is very similar to the previous one, except that it can be more suited for keyboard players.

By adding the command `UseKeyboardControl` to the init section of your setup you can remotely control the TinyBox with the use of MIDI Note messages instead of ProgChange messages. The Note message values used for remote control are specified a few pages ago, in the topic about the FCB1010 as footcontroller.

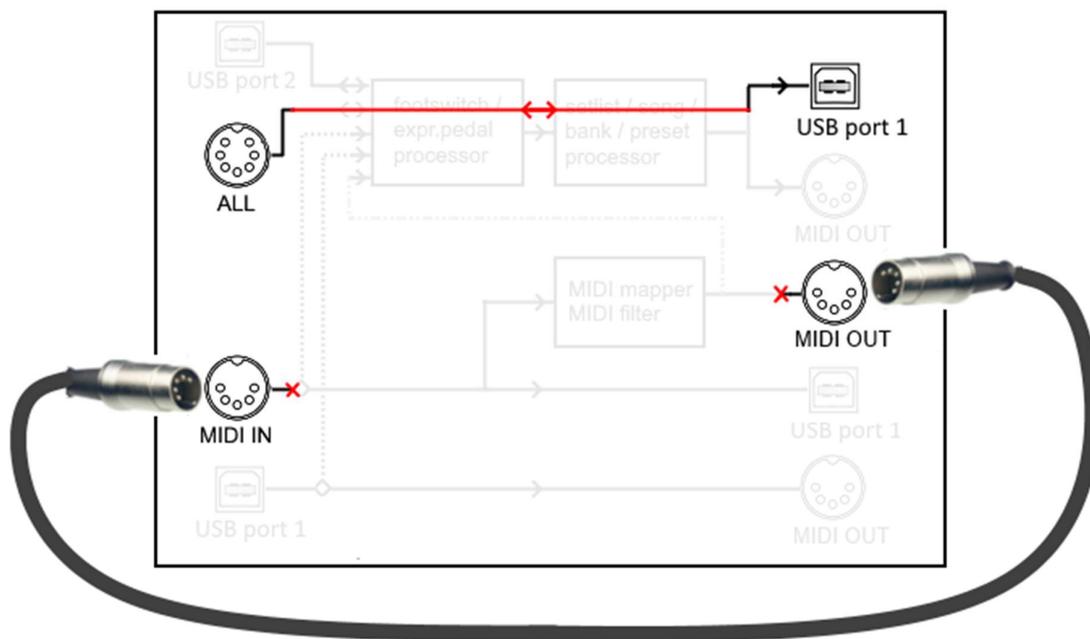
As you can see above the incoming Note messages first pass through the MIDI mapper before being routed to the footswitch processor. This means that you can map the lower octave of a keyboard to the required MIDI channel and Note range for remote control. All note messages which don't fall within the remote control range are forwarded to the MIDI OUT connector. This way you can use a master keyboard to play any connected synth module, while at the same time controlling the TinyBox presets using the lower octave of that keyboard.

Here's a sample MIDI mapper configuration:

```
CHANNEL MySynth = 1
CHANNEL RemoteControl = 13

INIT_TINYBOX =
{
  UseKeyboardControl
  MoveNotes MySynth C0-A0 to RemoteControl C-1 // this is NoteNumber 0-9
  MoveNotes MySynth Bb0-B0 to RemoteControl D0 // this is NoteNumber 14-15
}
```

A special case : troubleshooting



Factory reset

You can force the TinyBox in “troubleshooting” mode by powering it up with a MIDI cable connected directly from MIDI OUT to MIDI IN connector. In this mode the full TinyBox functionality is omitted, all MIDI messages coming from the FCB1010 are directly forwarded to MIDI-USB port 1, and all messages coming from MIDI-USB port 1 are directly forwarded to the FCB1010.

This mode helps troubleshooting the FCB1010 connection. If you suspect the FCB1010 has problems sending or receiving MIDI messages you can use a MIDI monitor application and check if a footswitch press or expression pedal movement results in the expected messages being sent. In the opposite direction a MIDI monitor application can send LED control messages to the FCB1010 and you can check if the floorboard reacts to these messages.

Apart from this specific routing, the troubleshooting mode also clears the setup stored in the TinyBox. If for some reason a corrupted setup would make the TinyBox unstable, you can always revert it to factory setting with no setup loaded by connecting a MIDI cable directly between MIDI IN and MIDI OUT during power-up.