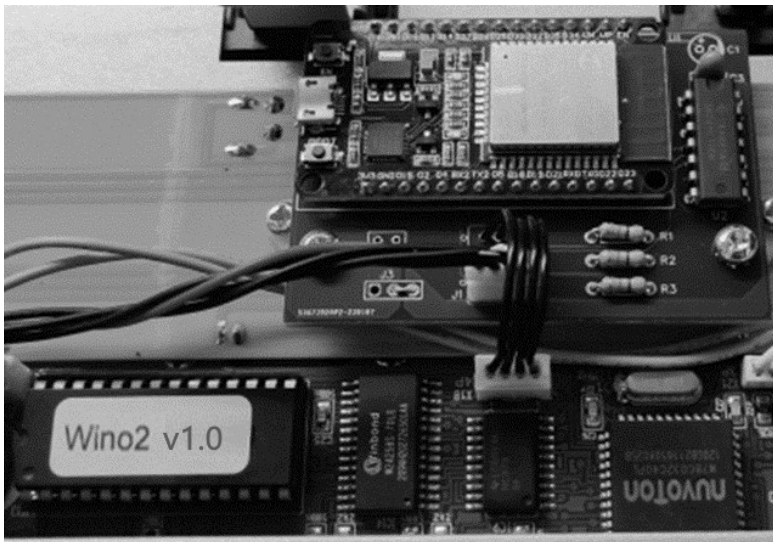


Wino2 user manual

Contents

Document versions

Version 1.0	31/07/2023	Initial release

Introduction: what is Wino2 ?

In short : *Wino2* is the acronym for "*WIFI-enabled UnO2*"

Over the past 15 years we have been releasing alternative firmware for the Behringer FCB1010. We named it "UnO firmware ", as in "UnOfficial firmware", since we are not affiliated with Behringer and therefore this firmware is not an official Behringer offering.

As many users will testify, the UnO firmware has brought several enhancements to the original FCB1010. Apart from some bugfixes (like correcting errors in the MIDI merge functionality) we added a few highly requested features, like a "real stompbox mode".

Even with these functional add-ons, the FCB1010 possibilities are still pretty limited, mainly because of the very limited setup storage capacity of the unit: each preset can send a maximum of 8 different MIDI messages (5xProgamChange, 2xControlChange, 1xNoteOn), while each of these 8 messages share the same MIDI channel for all presets.

In order to enhance the FCB1010 setup flexibility we created the UnO2 firmware, which is not an extension of the UnO firmware, but a completely new approach (actually a spin-off of the TinyBox hardware extension: https://www.tinybox.rocks/). Instead of the rigid 10x10 preset stucture, with up to 8 messages per preset, an UnO2 setup can contain any number of "presets", "stompboxes" or "triggers", organized in a fully flexible bank structure, and each preset can contain any number of MIDI messages, sent on any of the 16 MIDI channels.

Of course an UnO2 setup is still limited by the total available setup size of the FCB1010 hardware. This is why we initially believed the UnO2 firmware would never be as attractive as the TinyBox hardware extension, which contains about 100 times the FCB1010 memory size. The rising UnO2 popularity proved us wrong, and therefore we decided to bring some more TinyBox functionality to the UnO2 equipped FCB1010.

By installing the Wino2 module inside the FCB1010, you can make the floorboard WIFI enabled. This allows you to wirelessly connect your laptop or iPad to the FCB1010, and program your setup directly on the FCB1010, using the embedded text editor. You can read all about the "programming language" used to describe your setup in later chapters of this manual.

Making the FCB1010 WIFI enabled finally removes one of the big hurdles in programming the FCB1010: no more need for a MIDI-USB interface to connect the FCB1010 to your laptop. Many cheap interfaces unfortunately turn out to be incompatible with the FCB1010. So far the only solution was to purchase a more expensive MIDI-USB interface with dedicated drivers and large enough data buffers. The Wino2 module eliminates that requirement – instead of spending your money on a compatible MIDI-USB interface you can purchase a Wino2 module which offers some very interesting extra functionality.

The module contains a fullblown webserver, which not only hosts the necessary editor software to store and compile your UnO2 setup, but it also allows you to view live status info of your MIDI controller using the browser of your iPad or laptop: current bank, selected preset, activated effects, etc.

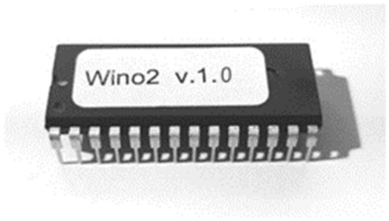
On top of that, the module also makes it possible to send or receive MIDI to or from your laptop or iPad over WIFI, using a protocol called "RTP MIDI" or "AppleMIDI".

Read on to learn all details about Wino2.

The Wino2 package

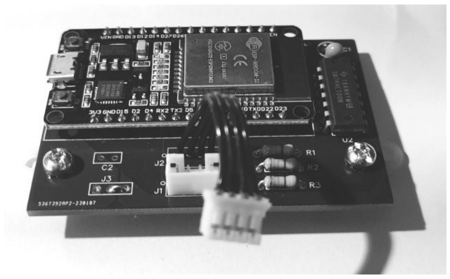
The Wino2 package contains 2 parts :

- A firmware chip.



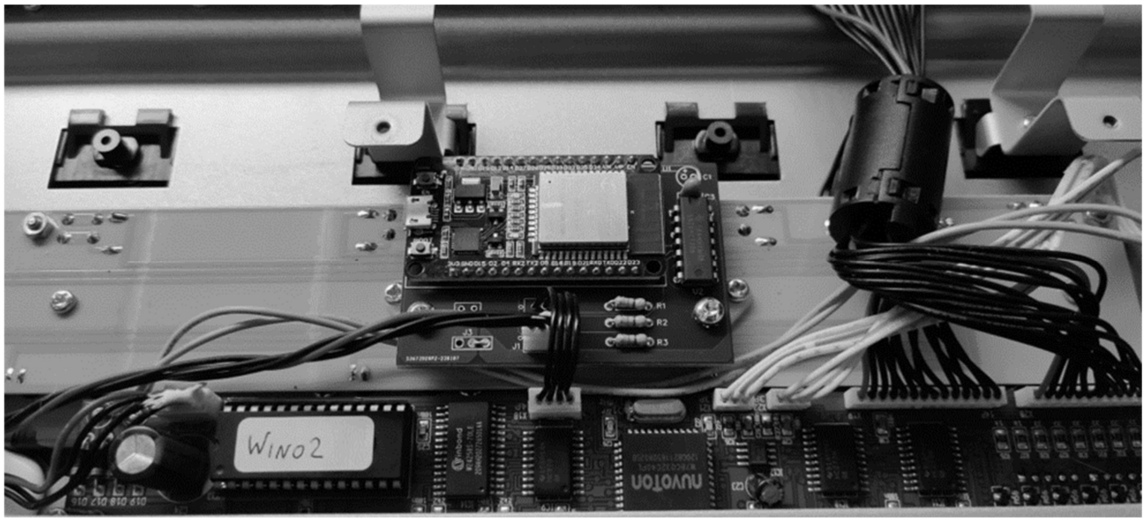
  This chip replaces the original Behringer firmware of the FCB1010, and offers exactly the same functionality as an UnO2 chip. Additionally it sends live status info to the WIFI module below. This allows you to view the current status of your FCB1010 on your laptop or iPad.

- A WIFI module.



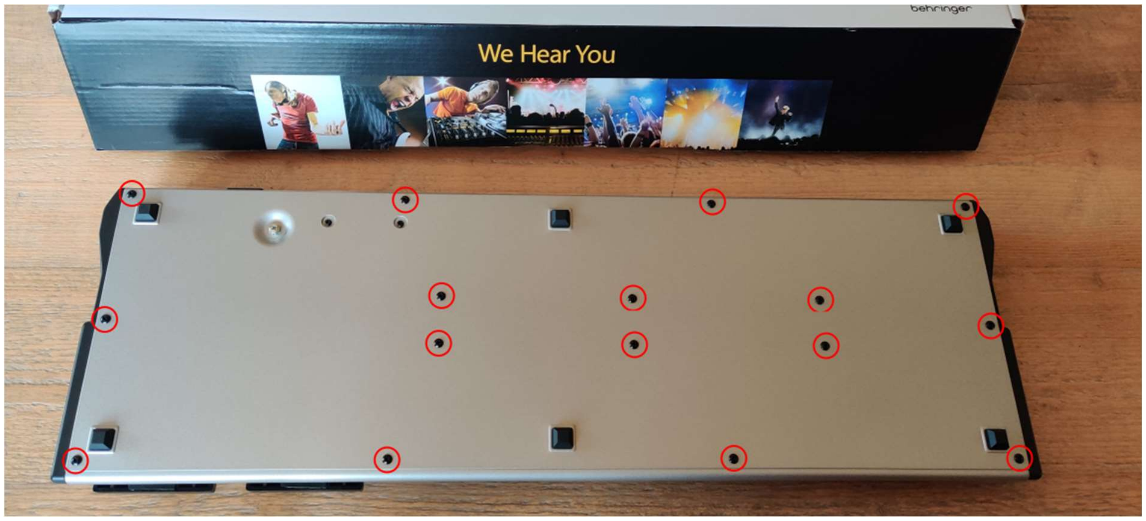
  The wireless connection provided by this module replaces the 2-way MIDI connection required so far to program the FCB1010. Moreover this module also contains editor software which runs in the browser and replaces the UnO2ControlCenter application. No more need for any software installation to program your floorboard.

  Installing the module is very simple. No soldering required, all you need is a screwdriver!

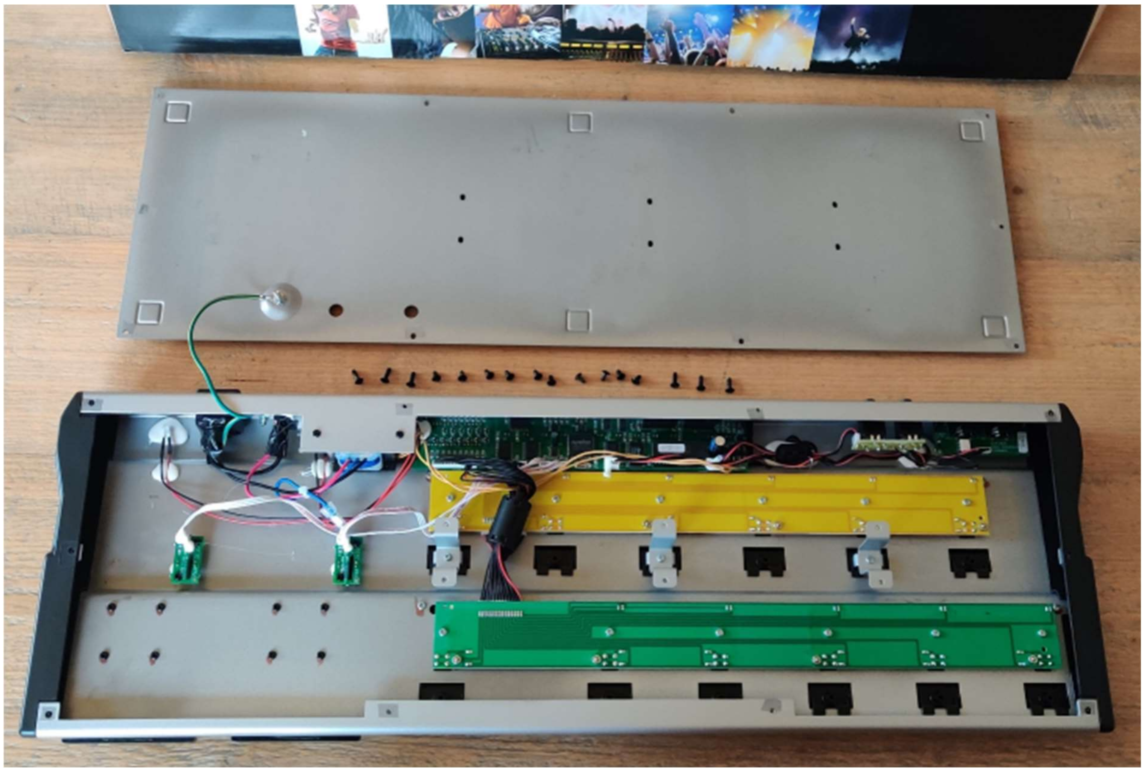


Getting started

1. Open up the FCB1010

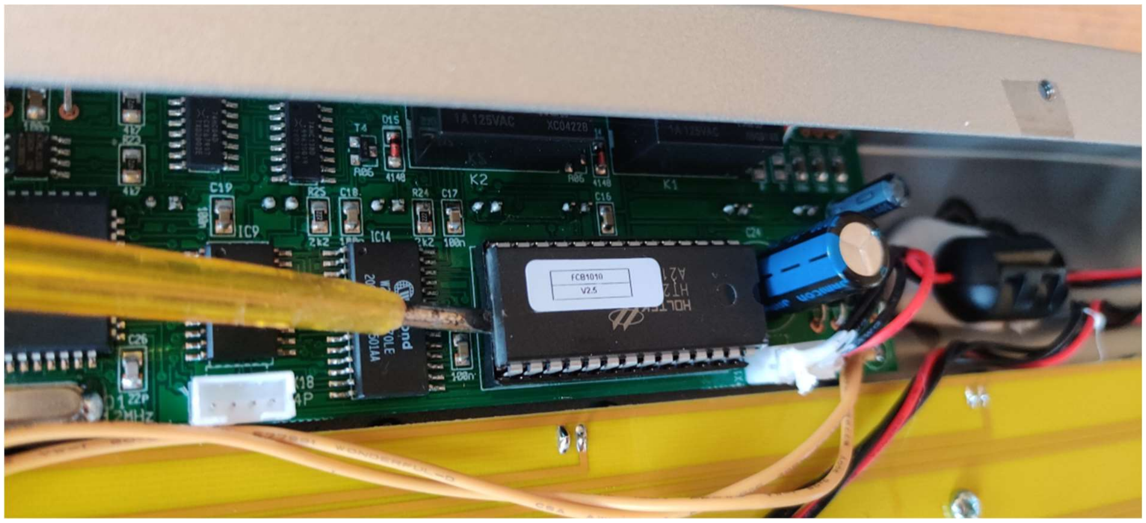


Turn the FCB1010 upside down, remove 16 screws to open the housing, lay the bottom plate next to the housing (leave the ground wire connected)



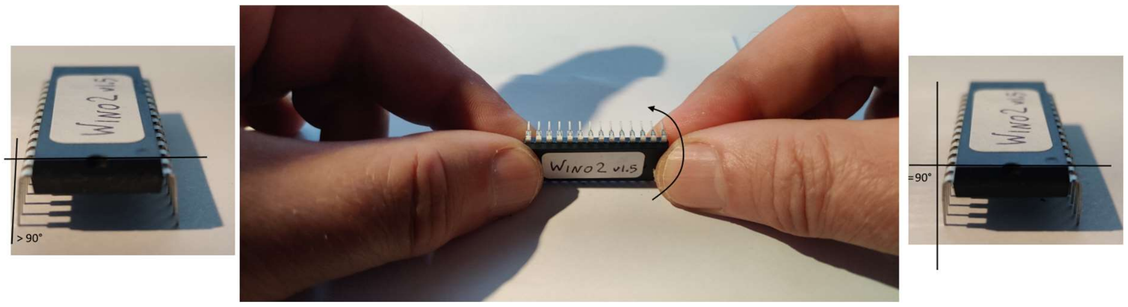
2. Install the Wino2 chip

Remove the original firmware chip from the socket on the main board. If necessary remove all hot melt adhesive which might be applied to the chip. You can use a small screwdriver to lift the chip out of the socket without bending the chip legs.



Store the original chip in a safe place, you might want to have it available if you would ever sell your FCB1010.

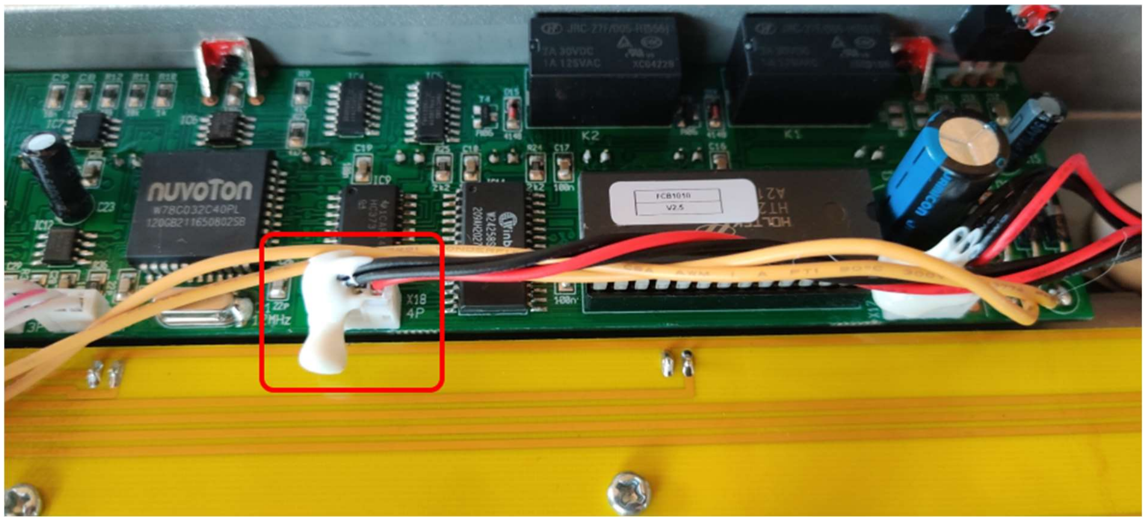
Before inserting the Wino2 chip, make sure the chip legs are perfectly perpendicular to the chip body. You can do that by slightly rotating the chip on a flat surface as follows :



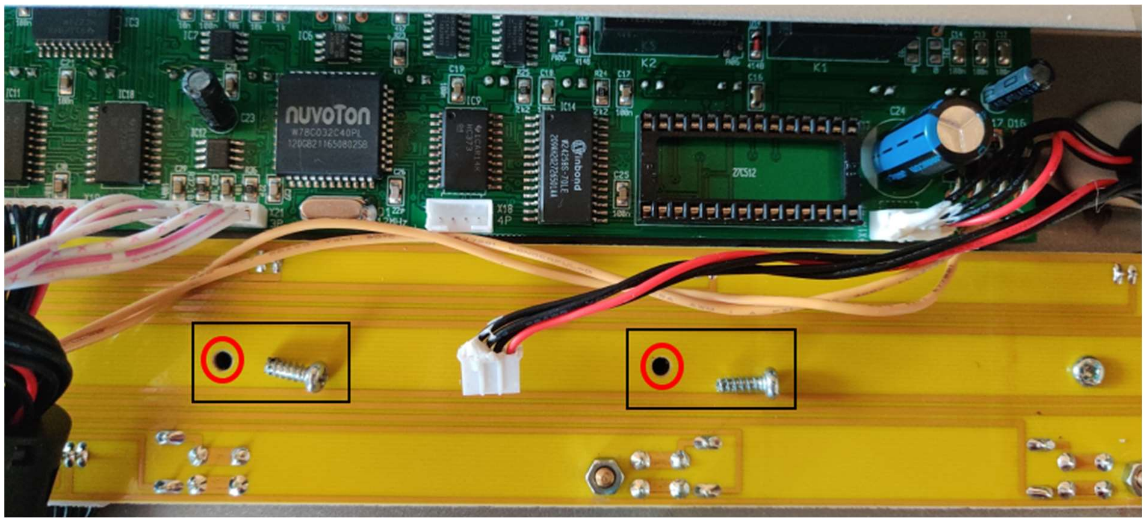
Place the Wino2 chip in the socket using the same orientation: the notch on one side of the EPROM must match the notch on the EPROM socket. Be careful not to press too hard, and pay attention not to bend any of the EPROM pins during insertion.

3. Install the Wino2 WIFI module

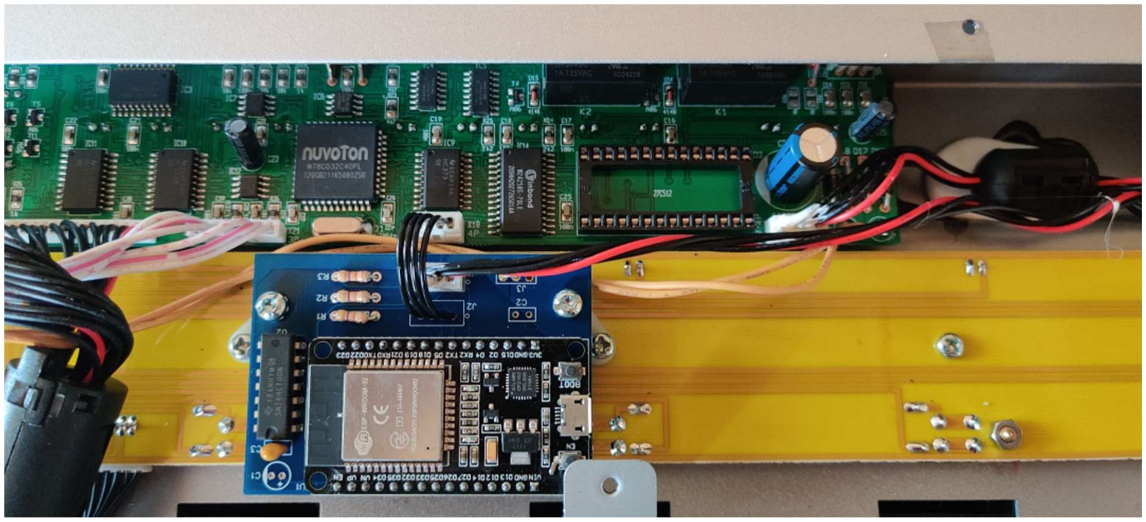
On the FCB1010 main board, unplug the red/black wire unit with 4 leads, which is running from the small MIDI connector board to the main board. Before you can do so it may be necessary to (carefully!) remove applied hot melt adhesive. Never pull the wires, you might pull them out of the connector housing. Instead remove all hot glue and wiggle the connector itself until it loosens from the main board.



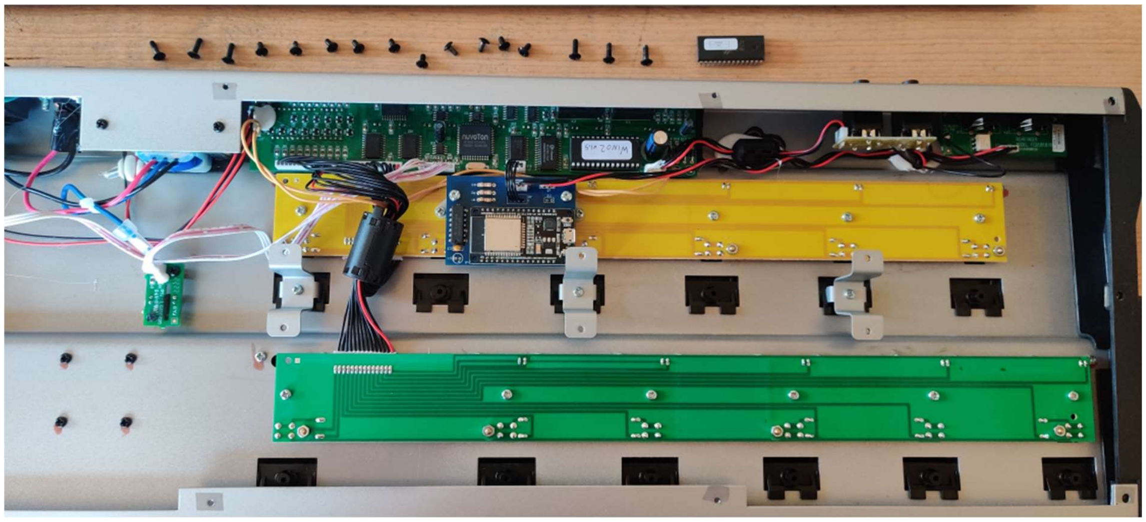
Remove following 2 screws which hold the FCB1010 switch board :



Reuse the 2 screws to mount the WIFI module. The module orientation is so that the 4-pins connector is close to the now empty 4-pins connector on the FCB1010 main board. Plug the large red/black wire into the connector of the WIFI module, and plug the short wire unit of the WIFI module into the 4-pins connector of the main board :

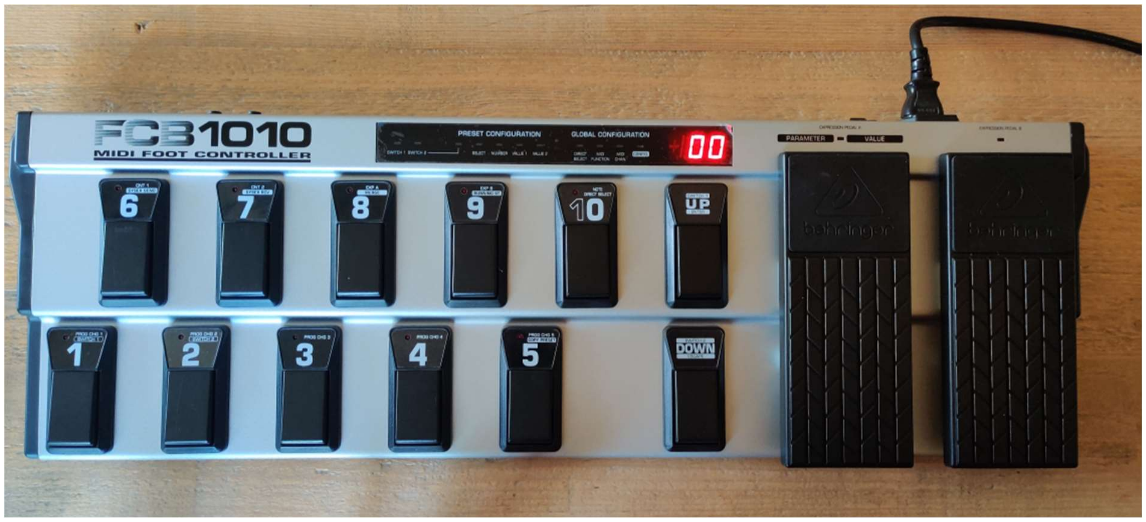


The FCB1010 inside now looks like this :



4. Test the FCB1010 and close the housing

Before closing the FCB1010 housing using the 16 screws, you can check if the modification was done correctly. Power up the FCB1010, the display should show “---” for a short time, followed by “00” :

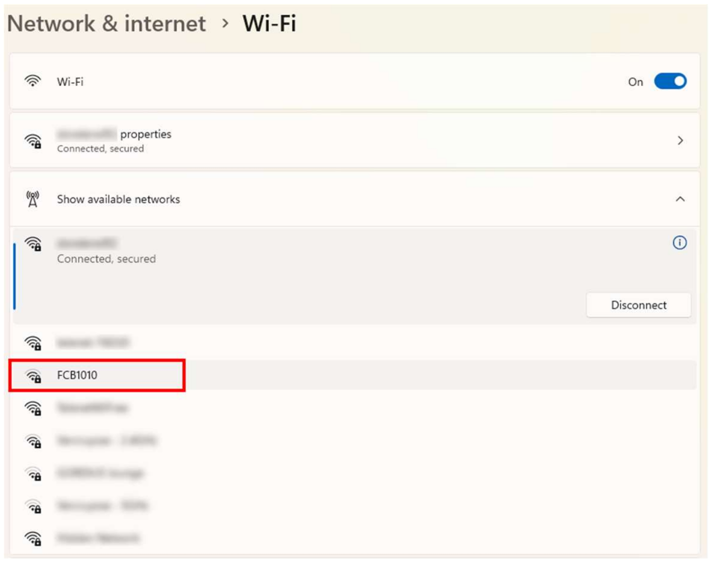


5. Calibrate the expression pedals

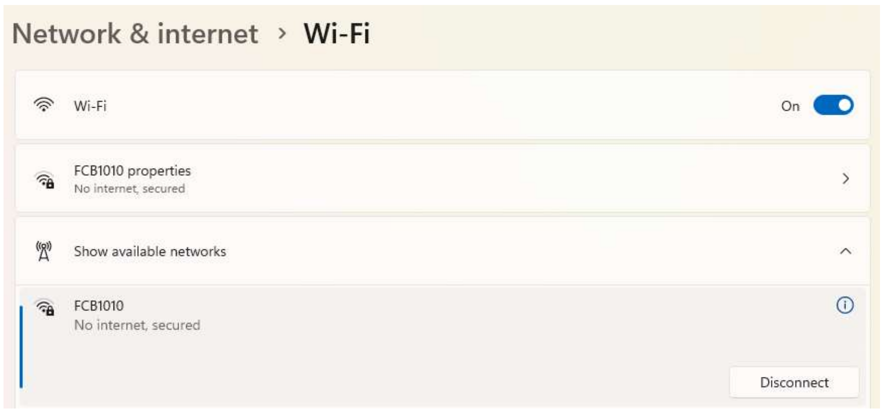
If not done already, don’t forget to calibrate the FCB1010 expression pedals. Failing to do so is the most common cause for non-working expression pedals. Calibration instructions can be found in the Behringer manual or by googling “FCB1010 calibration”. Keeping the 1+3 switches pressed during power-up initiates the calibration procedure. Keeping the 1+5 switches pressed during power-up initiates a full test procedure of all FCB1010 hardware components, followed by the pedal calibration.

6. Connect to the FCB1010 through WIFI

Use your laptop or iPad to connect to the WIFI accesspoint which is now embedded in the FCB1010. The SSID "FCB1010" should appear in your list of available WIFI networks (sample screenshots taken from a Windows computer) :



Select the access point with SSID **FCB1010** , and fill in the password, which is **FCB_Wino**

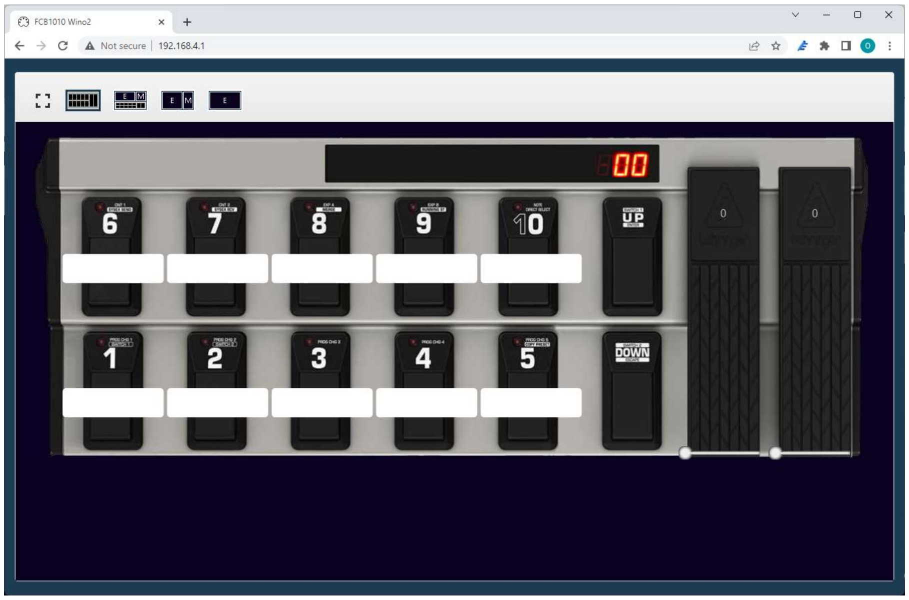


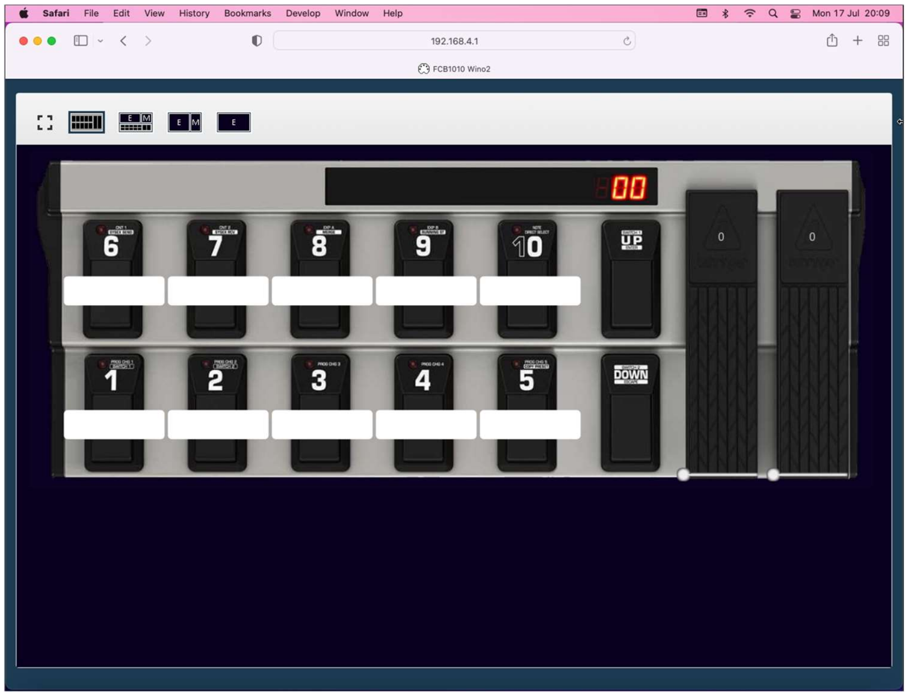
Remark :

Further on in this manual you will learn an alternative way of connecting to the FCB1010, using your home router instead of the embedded access point. That has the advantage of retaining your internet connection while being connected to the FCB1010. As you can see in the screenshot above no internet is available while using the embedded Wino2 access point.

7. Open the embedded web application

With your laptop wirelessly connected to the FCB1010 access point, you can open the embedded web application by surfing to IP address **192.168.4.1**. Preferably use Google Chrome (or a recent MS Edge browser on Windows), although Safari should also work correctly on MacOS.



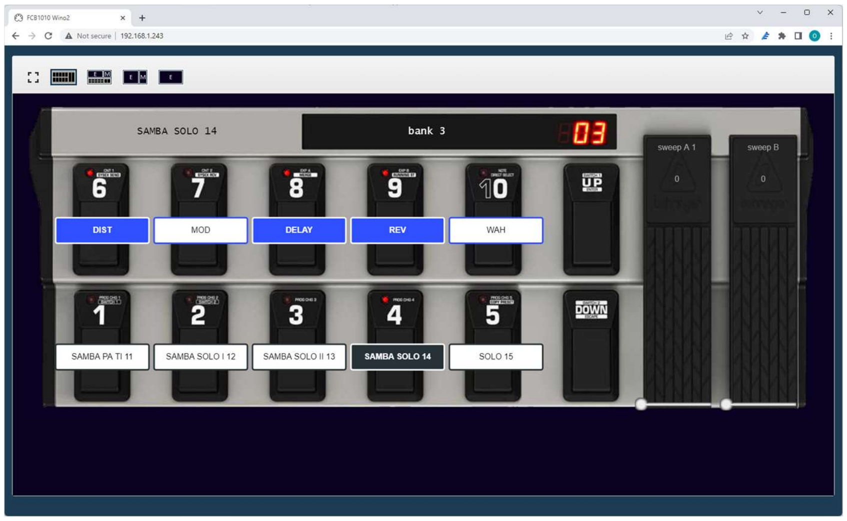


That's it, you are ready to rock and roll! Detailed info about the embedded web application and the "programming language" used for creating your FCB1010 setup can be found in further chapters of this manual.

The Wino2 web application

The web page shown when browsing to the FCB1010 IP address has 4 large icons in its header bar. These serve to choose the preferred combination of application views :

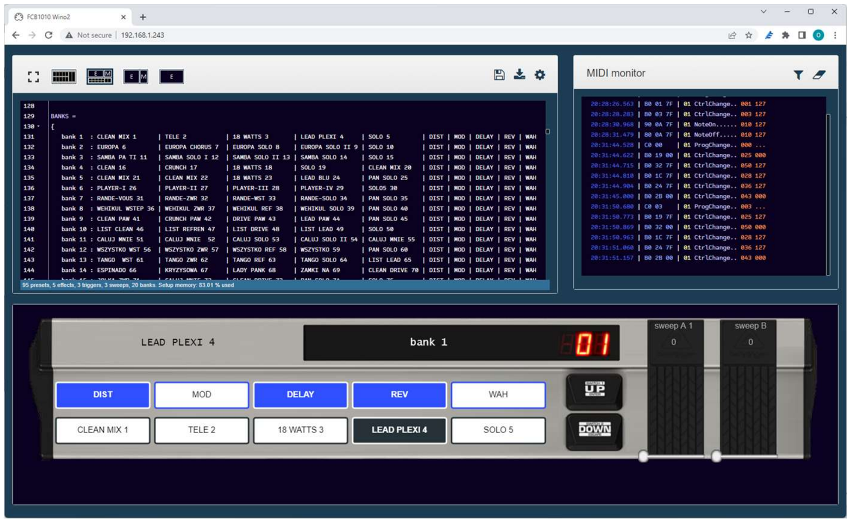
The first one, opened by default, is the FCB1010 status page. It shows an image of the FCB1010, with info on the currently selected bank, which presets are assigned to which switch, which effects are activated, etc. This is also a remote control for your FCB1010!



The next one shows a combination of all available windows on 1 screen :

- a shrinked status page
- an editor window containing your setup code
- a MIDI monitor window which shows all MIDI sent by the FCB1010

This screen can be useful while testing out setup changes



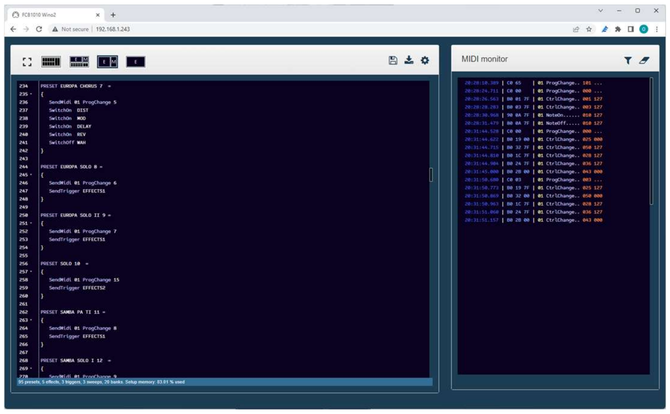
```

BANKS =
{
   bank 1  : CLEAN MIX 1      | TELE 2           | 18 WATTS 3         | LEAD PLEXI 4       | SOLO 5          | DIST | MOD | DELAY | REV | WAH
   bank 2  : EUROPA 6         | EUROPA CHORUS 7  | EUROPA SOLO 8      | EUROPA SOLO II 9   | SOLO 10         | DIST | MOD | DELAY | REV | WAH
   bank 3  : SAMBA PA TI 11   | SAMBA SOLO I 12  | SAMBA SOLO II 13   | SAMBA SOLO 14      | SOLO 15         | DIST | MOD | DELAY | REV | WAH
   bank 4  : CLEAN 16         | CRUNCH 17        | 18 WATTS 18        | SOLO 19            | CLEAN MIX 20    | DIST | MOD | DELAY | REV | WAH
   bank 5  : CLEAN MIX 21     | CLEAN MIX 22     | 18 WATTS 23        | LEAD BLU 24        | PAN SOLO 25     | DIST | MOD | DELAY | REV | WAH
   bank 6  : PLAYER-I 26      | PLAYER-II 27     | PLAYER-III 28      | PLAYER-IV 29       | SOLO5 30        | DIST | MOD | DELAY | REV | WAH
   bank 7  : RANDE-VOUS 31    | RANDE-ZWR 32     | RANDE-WST 33       | RANDE-SOLO 34      | PAN SOLO 35     | DIST | MOD | DELAY | REV | WAH
   bank 8  : WEHIKUL WSTEP 36 | WEHIKUL ZWR 37   | WEHIKUL REF 38     | WEHIKUL SOLO 39    | PAN SOLO 40     | DIST | MOD | DELAY | REV | WAH
   bank 9  : CLEAN PAW 41     | CRUNCH PAW 42    | DRIVE PAW 43       | LEAD PAW 44        | PAN SOLO 45     | DIST | MOD | DELAY | REV | WAH
   bank 10 : LIST CLEAN 46    | LIST REFREN 47   | LIST DRIVE 48      | LIST LEAD 49       | SOLO 50         | DIST | MOD | DELAY | REV | WAH
   bank 11 : CALUJ MNIE 51    | CALUJ MNIE  52   | CALUJ SOLO 53      | CALUJ SOLO II 54   | CALUJ MNIE 55   | DIST | MOD | DELAY | REV | WAH
   bank 12 : WSZYSTKO WST 56  | WSZYSTKO ZWR 57  | WSZYSTKO REF 58    | WSZYSTKO 59        | PAN SOLO 60     | DIST | MOD | DELAY | REV | WAH
   bank 13 : TANGO  WST 61    | TANGO ZWR 62     | TANGO REF 63       | TANGO SOLO 64      | LIST LEAD 65    | DIST | MOD | DELAY | REV | WAH
   bank 14 : ESPINADO 66      | KRYZYSOWA 67     | LADY PANK 68       | ZAMKI NA 69        | CLEAN DRIVE 70  | DIST | MOD | DELAY | REV | WAH
```

95 presets, 5 effects, 3 triggers, 3 sweeps, 20 banks. Setup memory: 83.01 % used

```
20:28:26.563 | B0 01 7F | 01 CtrlChange.. 001 127
20:28:28.283 | B0 03 7F | 01 CtrlChange.. 003 127
20:28:30.968 | 90 0A 7F | 01 NoteOn...... 010 127
20:28:31.479 | 80 0A 7F | 01 NoteOff..... 010 127
20:31:44.528 | C0 00    | 01 ProgChange.. 000 ...
20:31:44.622 | B0 19 00 | 01 CtrlChange.. 025 000
20:31:44.715 | B0 32 7F | 01 CtrlChange.. 050 127
20:31:44.810 | B0 1C 7F | 01 CtrlChange.. 028 127
20:31:44.904 | B0 24 7F | 01 CtrlChange.. 036 127
20:31:45.000 | B0 2B 00 | 01 CtrlChange.. 043 000
20:31:50.680 | C0 03    | 01 ProgChange.. 003 ...
20:31:50.773 | B0 19 7F | 01 CtrlChange.. 025 127
20:31:50.869 | B0 32 00 | 01 CtrlChange.. 050 000
20:31:50.963 | B0 1C 7F | 01 CtrlChange.. 028 127
20:31:51.060 | B0 24 7F | 01 CtrlChange.. 036 127
20:31:51.157 | B0 2B 00 | 01 CtrlChange.. 043 000
```

E M

This icon selects a combination of editor window and MIDI monitor, again useful while testing setup changes. Collapsing the status page leaves more space for the editor and monitor windows



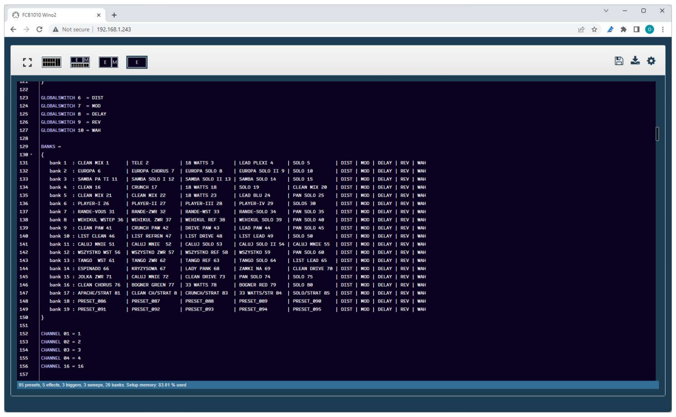
```
PRESET EUROPA CHORUS 7  =
{
   SendMidi 01 ProgChange 5
   SwitchOn  DIST
   SwitchOn  MOD
   SwitchOn  DELAY
   SwitchOn  REV
   SwitchOff WAH
}

PRESET EUROPA SOLO 8 =
{
   SendMidi 01 ProgChange 6
   SendTrigger EFFECTS1
}

PRESET EUROPA SOLO II 9 =
{
   SendMidi 01 ProgChange 7
   SendTrigger EFFECTS1
}

PRESET SOLO 10  =
{
   SendMidi 01 ProgChange 15
   SendTrigger EFFECTS2
}

PRESET SAMBA PA TI 11 =
{
   SendMidi 01 ProgChange 8
   SendTrigger EFFECTS1
}

PRESET SAMBA SOLO I 12  =
{
   SendMidi 01 ProgChange 9
```

95 presets, 5 effects, 3 triggers, 3 sweeps, 20 banks. Setup memory: 83.01 % used

```
20:28:10.389 | C0 65    | 01 ProgChange.. 101 ...
20:28:24.711 | C0 00    | 01 ProgChange.. 000 ...
20:28:26.563 | B0 01 7F | 01 CtrlChange.. 001 127
20:28:28.283 | B0 03 7F | 01 CtrlChange.. 003 127
20:28:30.968 | 90 0A 7F | 01 NoteOn...... 010 127
20:28:31.479 | 80 0A 7F | 01 NoteOff..... 010 127
20:31:44.528 | C0 00    | 01 ProgChange.. 000 ...
20:31:44.622 | B0 19 00 | 01 CtrlChange.. 025 000
20:31:44.715 | B0 32 7F | 01 CtrlChange.. 050 127
20:31:44.810 | B0 1C 7F | 01 CtrlChange.. 028 127
20:31:44.904 | B0 24 7F | 01 CtrlChange.. 036 127
20:31:45.000 | B0 2B 00 | 01 CtrlChange.. 043 000
20:31:50.680 | C0 03    | 01 ProgChange.. 003 ...
20:31:50.773 | B0 19 7F | 01 CtrlChange.. 025 127
20:31:50.869 | B0 32 00 | 01 CtrlChange.. 050 000
20:31:50.963 | B0 1C 7F | 01 CtrlChange.. 028 127
20:31:51.060 | B0 24 7F | 01 CtrlChange.. 036 127
20:31:51.157 | B0 2B 00 | 01 CtrlChange.. 043 000
```

E

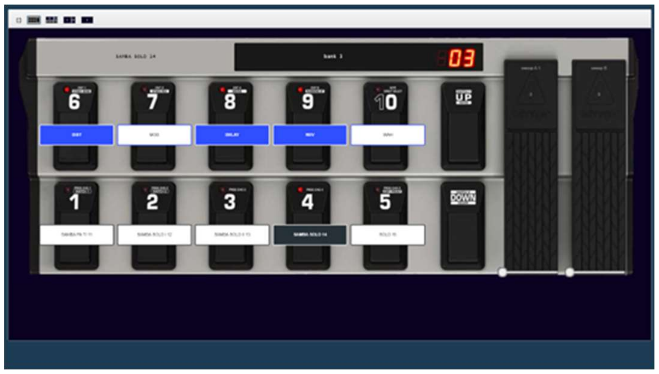
Use the full-screen editor page while creating your setup.



```
GLOBALSWITCH 6  = DIST
GLOBALSWITCH 7  = MOD
GLOBALSWITCH 8  = DELAY
GLOBALSWITCH 9  = REV
GLOBALSWITCH 10 = WAH

BANKS =
{
   bank 1  : CLEAN MIX 1       | TELE 2            | 18 WATTS 3         | LEAD PLEXI 4      | SOLO 5          | DIST | MOD | DELAY | REV | WAH
   bank 2  : EUROPA 6          | EUROPA CHORUS 7   | EUROPA SOLO 8      | EUROPA SOLO II 9  | SOLO 10         | DIST | MOD | DELAY | REV | WAH
   bank 3  : SAMBA PA TI 11    | SAMBA SOLO I 12   | SAMBA SOLO II 13   | SAMBA SOLO 14     | SOLO 15         | DIST | MOD | DELAY | REV | WAH
   bank 4  : CLEAN 16          | CRUNCH 17         | 18 WATTS 18        | SOLO 19           | CLEAN MIX 20    | DIST | MOD | DELAY | REV | WAH
   bank 5  : CLEAN MIX 21      | CLEAN MIX 22      | 18 WATTS 23        | LEAD BLU 24       | PAN SOLO 25     | DIST | MOD | DELAY | REV | WAH
   bank 6  : PLAYER-I 26       | PLAYER-II 27      | PLAYER-III 28      | PLAYER-IV 29      | SOLO5 30        | DIST | MOD | DELAY | REV | WAH
   bank 7  : RANDE-VOUS 31     | RANDE-ZWR 32      | RANDE-WST 33       | RANDE-SOLO 34     | PAN SOLO 35     | DIST | MOD | DELAY | REV | WAH
   bank 8  : WEHIKUL WSTEP 36  | WEHIKUL ZWR 37    | WEHIKUL REF 38     | WEHIKUL SOLO 39   | PAN SOLO 40     | DIST | MOD | DELAY | REV | WAH
   bank 9  : CLEAN PAW 41      | CRUNCH PAW 42     | DRIVE PAW 43       | LEAD PAW 44       | PAN SOLO 45     | DIST | MOD | DELAY | REV | WAH
   bank 10 : LIST CLEAN 46     | LIST REFREN 47    | LIST DRIVE 48      | LIST LEAD 49      | SOLO 50         | DIST | MOD | DELAY | REV | WAH
   bank 11 : CALUJ MNIE 51     | CALUJ MNIE  52    | CALUJ SOLO 53      | CALUJ SOLO II 54  | CALUJ MNIE 55   | DIST | MOD | DELAY | REV | WAH
   bank 12 : WSZYSTKO WST 56   | WSZYSTKO ZWR 57   | WSZYSTKO REF 58    | WSZYSTKO 59       | PAN SOLO 60     | DIST | MOD | DELAY | REV | WAH
   bank 13 : TANGO  WST 61     | TANGO ZWR 62      | TANGO REF 63       | TANGO SOLO 64     | LIST LEAD 65    | DIST | MOD | DELAY | REV | WAH
   bank 14 : ESPINADO 66       | KRYZYSOWA 67      | LADY PANK 68       | ZAMKI NA 69       | CLEAN DRIVE 70  | DIST | MOD | DELAY | REV | WAH
   bank 15 : JOLKA ZWR 71      | CALUJ MNIE 72     | CLEAN DRIVE 73     | PAN SOLO 74       | SOLO 75         | DIST | MOD | DELAY | REV | WAH
   bank 16 : CLEAN CHORUS 76   | BOGNER GREEN 77   | 33 WATTS 78        | BOGNER RED 79     | SOLO 80         | DIST | MOD | DELAY | REV | WAH
   bank 17 : APACHE/STRAT 81   | CLEAN CH/STRAT 8  | CRUNCH/STRAT 83    | 33 WATTS/STR 84   | SOLO/STRAT 85   | DIST | MOD | DELAY | REV | WAH
   bank 18 : PRESET_086        | PRESET_087        | PRESET_088         | PRESET_089        | PRESET_090      | DIST | MOD | DELAY | REV | WAH
   bank 19 : PRESET_091        | PRESET_092        | PRESET_093         | PRESET_094        | PRESET_095      | DIST | MOD | DELAY | REV | WAH
}

CHANNEL 01 = 1
CHANNEL 02 = 2
CHANNEL 03 = 3
CHANNEL 04 = 4
CHANNEL 16 = 16
```

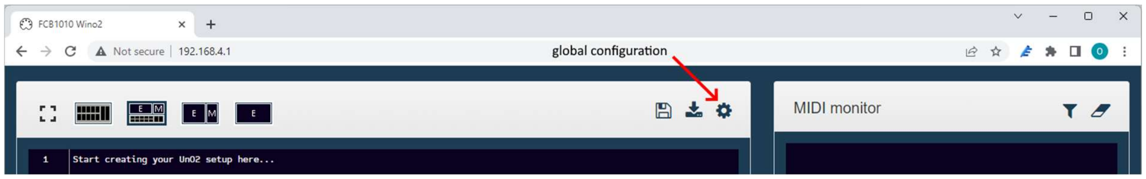
95 presets, 5 effects, 3 triggers, 3 sweeps, 20 banks. Setup memory: 83.01 % used

The icon to the left of the 4 display option buttons toggles the browser window between “fullscreen” and normal mode. Fullscreen mode will mainly be interesting for displaying the status screen during a gig, since you won’t be using your iPad or laptop for anything else at that moment.



Global configuration

A few global configuration settings can be configured by clicking the cog icon at the right side of the header bar :



```
1   Start creating your Un02 setup here...
```

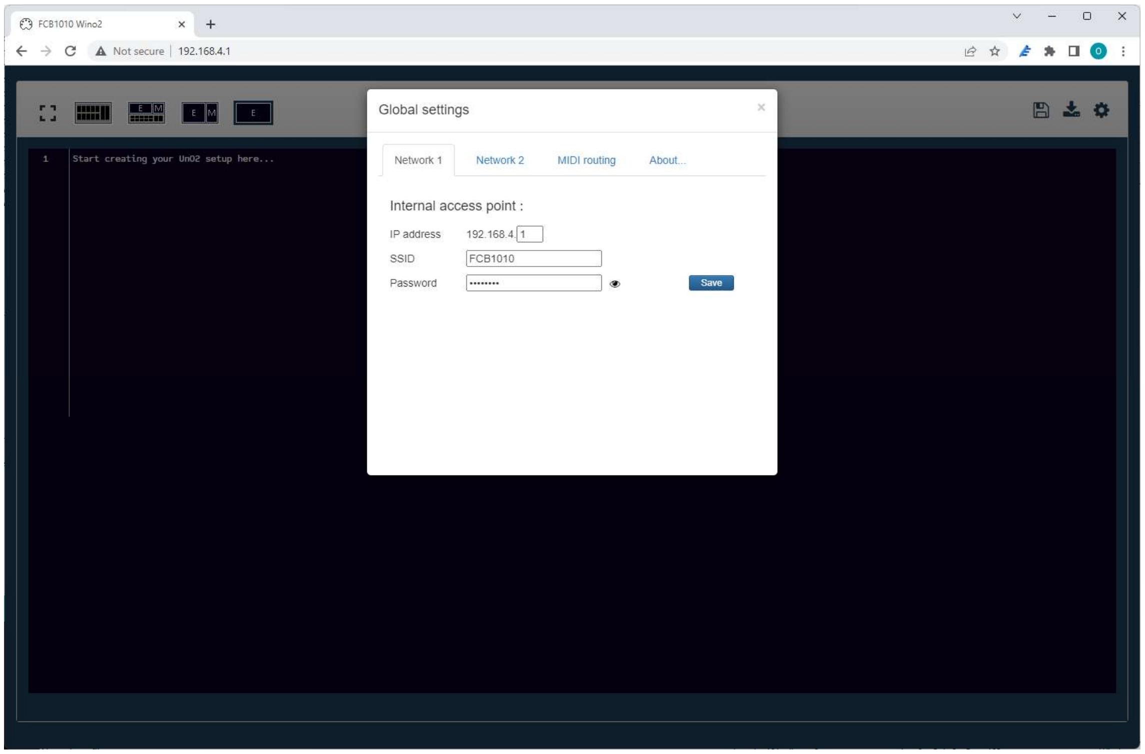
1. Changing the embedded access point SSID or password

The first tab of the configuration screen lets you modify the IP address of the Wino2 module (actually there is little reason to do so, but you can…) or the SSID and password of the Wino2 access point.

Attention : the password needs to be 8 characters minimum!

These settings can be reset at all time (for instance in case you would forget the modified password) by keeping the FCB1010 1+2 footswitches pressed during powerup. This reverts all access point settings to its defaults, which are :

```
IP       : 192.168.4.1
SSID     : FCB1010
Password : FCB_Wino
```



```
1   Start creating your Un02 setup here...
```

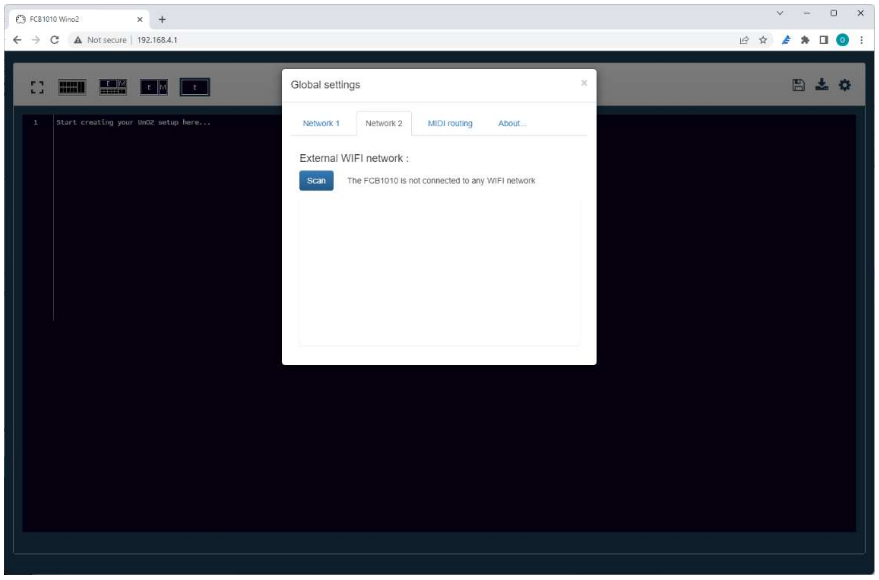
Internal access point :

IP address	192.168.4.1
SSID	FCB1010
Password	••••••••

Connecting to the FCB1010 using your home WIFI access point

The Wino2 module has an interesting extra WIFI option: next to the built-in access point, which we used so far, you can also let the Wino2 module connect to your regular WIFI access point at home. This way both your FCB1010 and your laptop can connect to the same access point, and you can reach the FCB1010 by surfing to its newly assigned IP address (see below). The major advantage of this is that your laptop doesn't lose its internet connectivity this way (provided that your WIFI access point is connected to the internet through your home router of course)

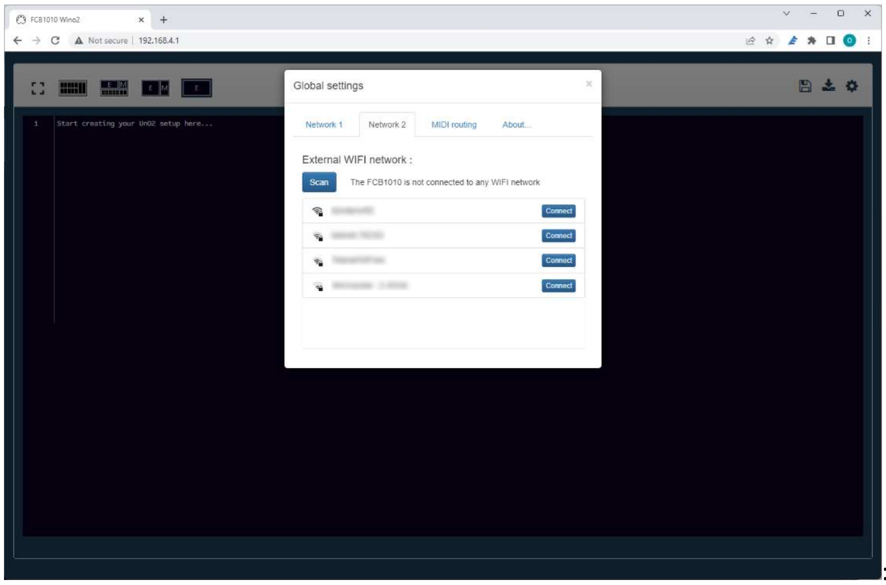
While currently still connected to the Wino2 embedded access point, click the second tab of the global settings menu, labeled "Network 2" :



```
Start creating your Uno2 setup here...
```

The FCB1010 is not connected to any WIFI network

Click the "Scan" button to search for nearby WIFI access points :

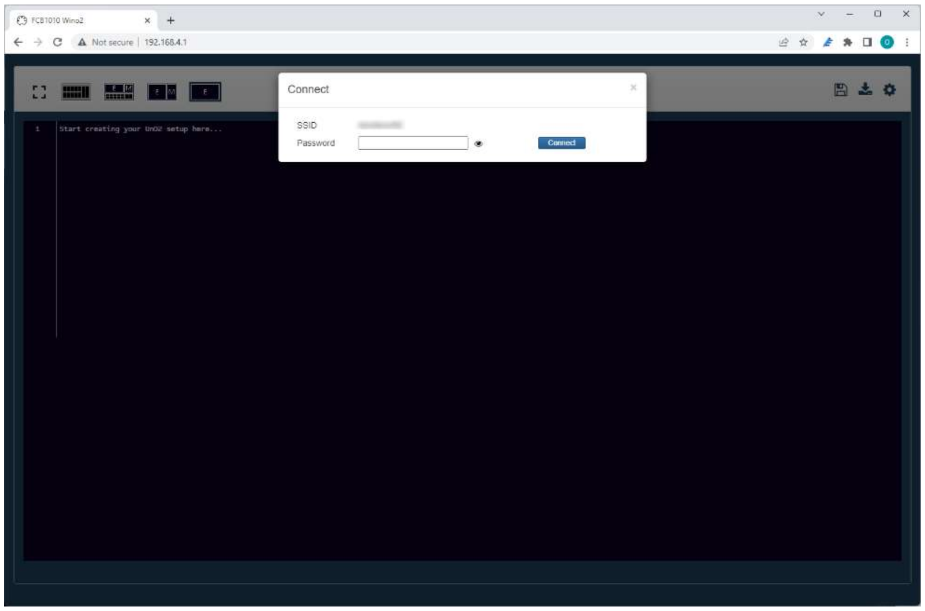


```
Start creating your Uno2 setup here...
```

The FCB1010 is not connected to any WIFI network

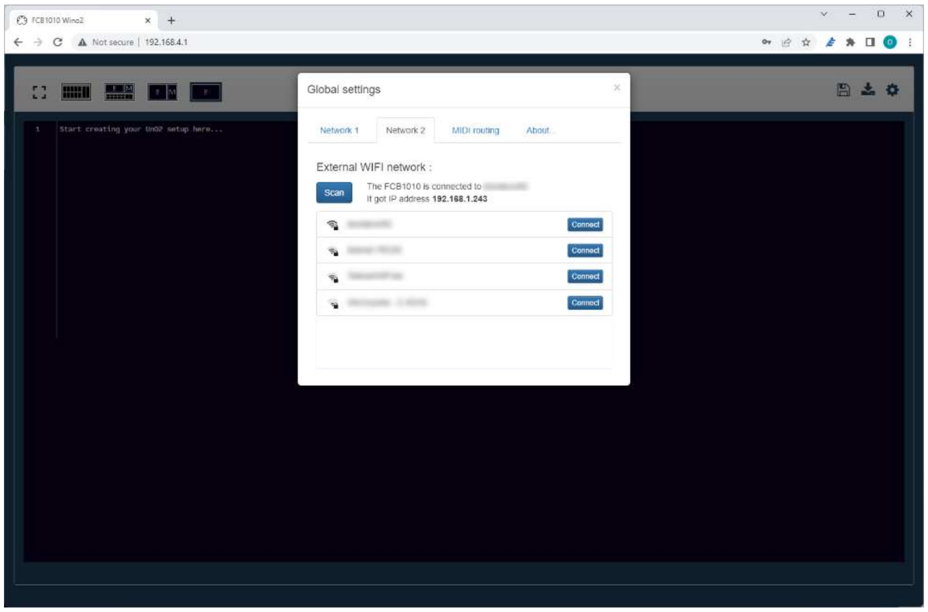
:

Then click the "Connect" button of the preferred access point, and (if required) enter its password :



Start creating your Uno2 setup here...

When connection succeeds, the access point will supply an IP address to the FCB1010. It is mentioned in the setup menu :



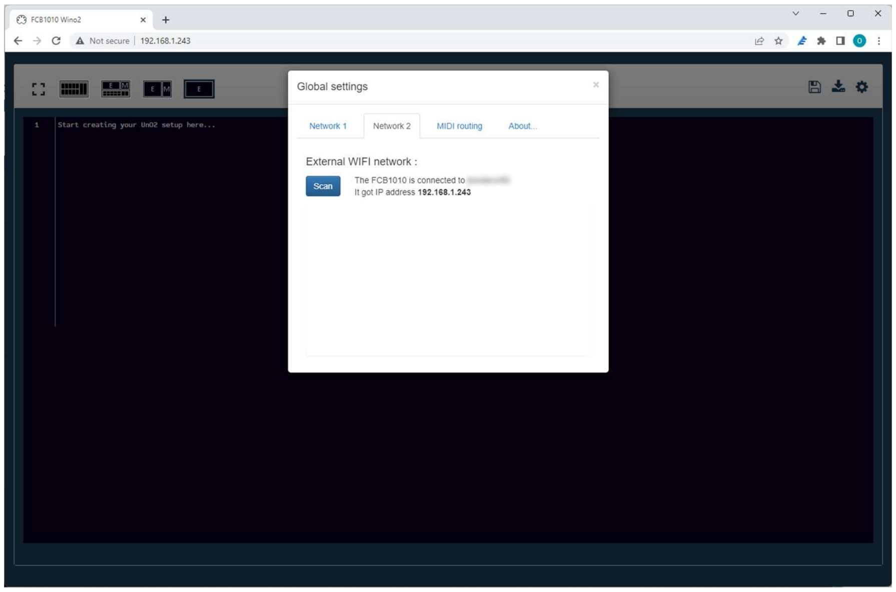
Start creating your Uno2 setup here...

External WIFI network :

The FCB1010 is connected to [redacted]
It got IP address **192.168.1.243**

With this configuration now stored in the Wino2 module, you can disconnect your laptop from the Wino2 embedded access point, and connect to your regular home access point again. Now open your browser, and instead of entering the default Wino2 IP address 192.168.4.1, enter the IP address which was mentioned in the setup menu above. Your laptop will again connect wirelessly to the FCB1010, while maintaining its internet connection through your home router.

Pay attention to the modified IP address typed in the browser address bar in screenshot below:



External WIFI network :

The FCB1010 is connected to [redacted]
It got IP address **192.168.1.243**

For creating or editing your Wino2 setup, you will prefer to use this new WIFI configuration. It allows you to reach your FCB1010 by surfing to its assigned IP address, without changing any of your original laptop WIFI settings. On the other hand, while at a gig, you can have a live status view of your FCB1010 by connecting your iPad to the embedded Wino2 access point and surfing to 192.168.4.1. Indeed, you would not want to rely on any public access point for viewing status info during a gig.

Both connection methods (through embedded access point or via an external access point) remain simultaneously available, so after returning from a gig you can again connect through your home router without any reconfiguration required.

2. Sending MIDI over WIFI

Having WIFI connectivity available allows us to let the FCB1010 send wireless MIDI to a laptop, or receive wireless MIDI from a laptop and forward it to other MIDI devices through its MIDI OUT connector. No more need for an extra “WIDI” dongle!

The Wino2 module uses the “RTP-MIDI” technology for this. More info about this MIDI standard can be found on following links :

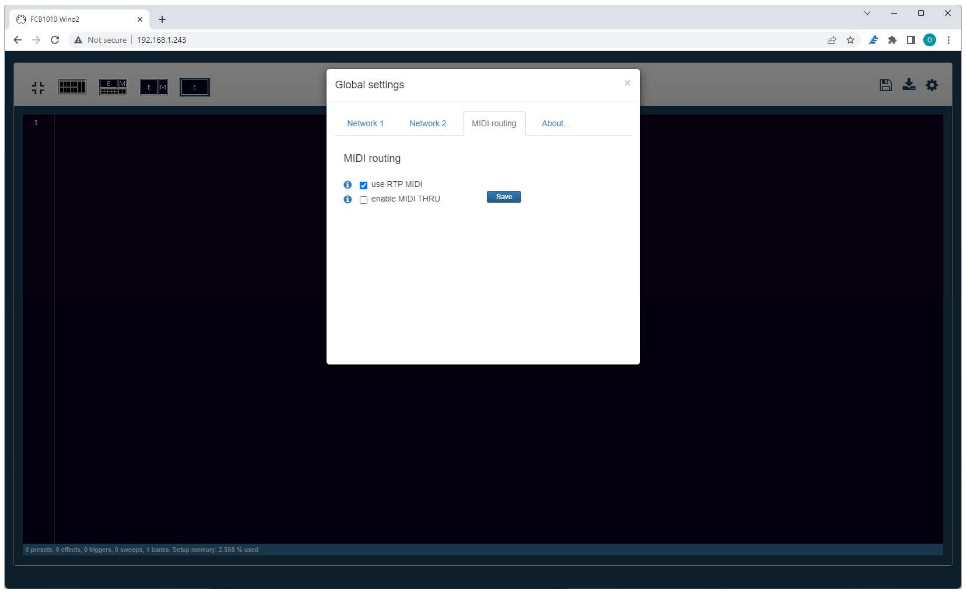
- https://www.midi.org/midi-articles/rtp-midi-or-midi-over-networks
- https://en.wikipedia.org/wiki/RTP-MIDI

On macOS and iOS, RTP-MIDI (also known as AppleMIDI) is supported out of the box.

On Windows, a free RTP-MIDI driver can be installed. More info about that in the links above.

On Android, an app called MIDI Connector adds support for the RTP-MIDI protocol.

You can enable RTP-MIDI in the “MIDI routing” tab of the setup menu :



MIDI routing

[✓] use RTP MIDI
[ ] enable MIDI THRU

0 presets, 0 effects, 0 triggers, 0 sweeps, 1 banks. Setup memory: 2.588 % used

3. Enabling MIDI THRU

The same “MIDI routing” tab above also has a checkbox to enable MIDI THRU. When doing so, all MIDI sent to the FCB1010 through its MIDI IN connector will be forwarded to the MIDI OUT connector.

Saving and sending FCB1010 setups

Without going into detail now about how to program the Wino2 equipped FCB1010 setup (this is covered in great detail in later chapters) let's continue going through all menu options of the web application. Next to the cog icon discussed above, the editor window also has a Save button and a Send button

1. Saving the setup source

The Save button stores the setup text, which you have entered in the editor window, in the non-volatile memory of the Wino2 module. Click the Save button regularly in order not to lose your work while creating your setup.

As you will notice, the Wino2 module can store one single setup source file. However, thanks to the fact that this "setup source" is actually plain text, it is very easy to copy-and-paste the full setup text from the browser into any text editor and save the setup on your laptop, and vice versa load one of multiple alternative setups, stored on your laptop, into the web application and save it on the Wino2 module.

It is definitely a good idea to make an extra copy of your setup source text on your laptop this way. In case a hickup would occur when saving the setup to the Wino2 memory, you then still have a backup of the setup source on your laptop.

2. Sending the setup binary

In order for your setup to be used by the FCB1010, it is first "compiled" into binary data. When clicking the "Send" button this binary data is sent from the Wino2 module to the FCB1010 main board, where it is stored in the permanent setup memory (the same memory which stores a Behringer, UnO or UnO2 setup when no Wino2 module is installed)

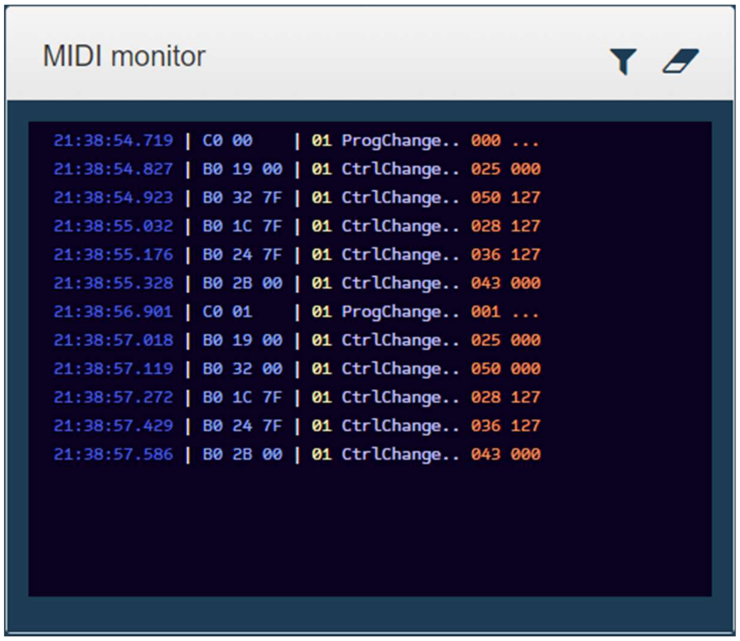
For those who really like to know every little detail of the system:
If you look closely at the popup shown after clicking the Send button, you will notice 2 different status messages: first it says "Saving setup structure", followed by "Storing setup…". Indeed, before storing the binary setup data in the FCB1010 setup memory, the setup *structure* (bank names, bank layout, switch assignments, …) is retrieved from the setup source text, and stored separately by the Wino2 embedded web server. During a gig this info is combined with live data sent from the FCB1010 main board to the Wino2 module to show current status info in the browser.

MIDI monitor screen

Things are not always working correctly right from the start. If your gear doesn't behave as expected you might want to inspect the MIDI messages which the FCB1010 sends. You can do so with the monitor screen of the built-in web application. With this monitor you can get a detailed inside view on all outgoing MIDI traffic.

The MIDI message list shown in that screen contains following data for each transmitted MIDI message:

- Timestamp on which the message was received or sent
- Hexadecimal representation of the MIDI message. A MIDI message can contain 1, 2 or 3 bytes, except for SysEx messages which can contain more bytes and are shown on multiple lines in the message list
- Text representation of the MIDI message, which is :
    - The MIDI channel (between 01 and 16), where applicable
    - The MIDI message type
    - 0, 1 or 2 MIDI data bytes (or multiple bytes in case of SysEx)

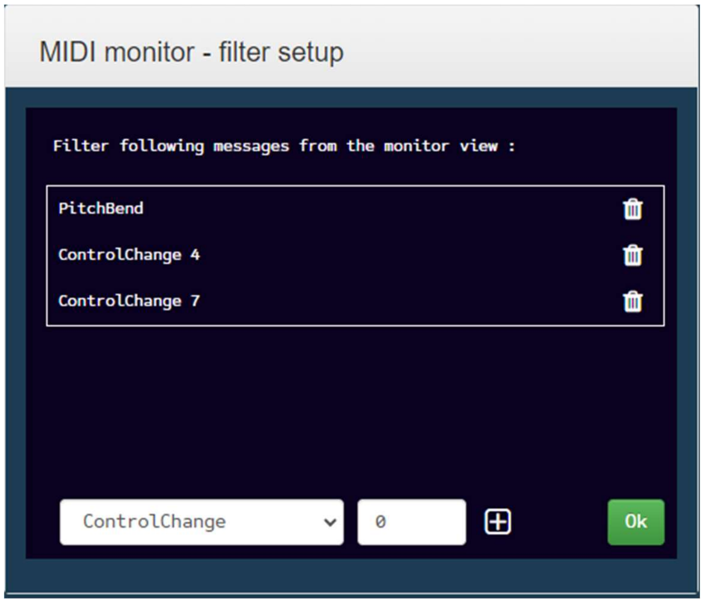


```
21:38:54.719 | C0 00    | 01 ProgChange.. 000 ...
21:38:54.827 | B0 19 00 | 01 CtrlChange.. 025 000
21:38:54.923 | B0 32 7F | 01 CtrlChange.. 050 127
21:38:55.032 | B0 1C 7F | 01 CtrlChange.. 028 127
21:38:55.176 | B0 24 7F | 01 CtrlChange.. 036 127
21:38:55.328 | B0 2B 00 | 01 CtrlChange.. 043 000
21:38:56.901 | C0 01    | 01 ProgChange.. 001 ...
21:38:57.018 | B0 19 00 | 01 CtrlChange.. 025 000
21:38:57.119 | B0 32 00 | 01 CtrlChange.. 050 000
21:38:57.272 | B0 1C 7F | 01 CtrlChange.. 028 127
21:38:57.429 | B0 24 7F | 01 CtrlChange.. 036 127
21:38:57.586 | B0 2B 00 | 01 CtrlChange.. 043 000
```

Attention : be aware that showing a lot of MIDI info in the browser can slow down the transmission of MIDI by the Wino2 module. Therefore use the monitor screen only while inspecting or troubleshooting your setup, never during live use of the FCB1010.

Message filtering

The amount of MIDI data sent by the FCB1010 can be large, especially when using the expression pedals. Therefore the monitor screen has the option to filter out certain message types, typically sent when using the expression pedals. For ControlChange messages you can choose which CC number is being filtered (e.g. CC07 & CC04, which are continuous control messages)


MIDI monitor - filter setup

Filter following messages from the monitor view :

PitchBend

ControlChange 4

ControlChange 7

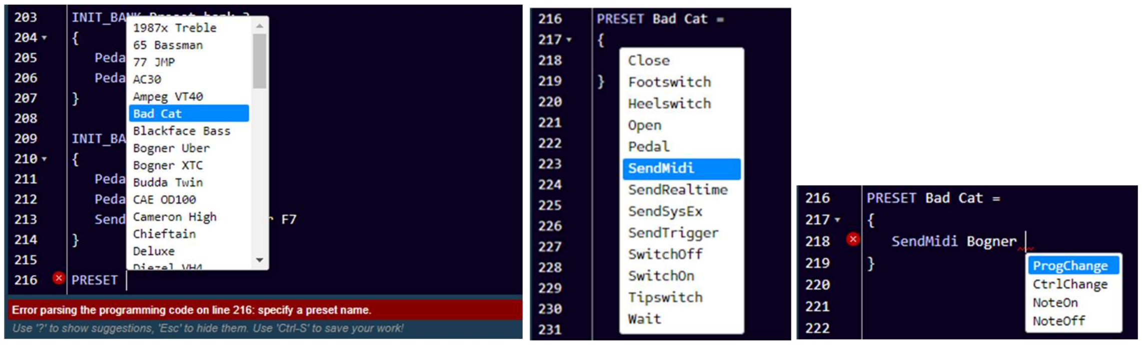
*This concludes the "Wino2" specific part of this manual. The way you program your Wino2 equipped FCB1010, and the commands used in the setup editor, are identical with an UnO2 equipped FCB1010. Therefore the rest of this manual is a copy of the UnO2 manual, and we will refer to "**UnO2**" instead of "**Wino2**" from here on.*

The UnO2 setup structure

To specify the content of a UnO2 preset, you use a programming language which contains a limited number of easy commands like `SendMidi`, `SendSysEx`, etc. While creating your setup, the editor constantly gives you hints about the commands which are available in the current context. At the start of a new line just type ‘?’ to get a list of possible commands.

A setup can be kept very straightforward, just specifying one or a few MIDI commands for each preset in your setup. However, the used programming language allows you to take it a step further if you wish, and for instance make use of variables, an essential concept in any programming language. The content of a variable can be set or modified in one preset, then in another preset you can use conditional "if... then... else..." statements to act upon the variable contents. This way you can create a very dynamic and smart setup.

As a UnO2 setup can be described in plain text format, it is very easy to save, share or backup your setups as text files. Also editing a setup is very simple: any text editor will do. However, we do provide a very intuitive setup editor which can do more than a regular text editor: it has “intelligent code completion” which assists you by suggesting the possible choices at any place in your setup:



```
PRESET
```

Error parsing the programming code on line 216: specify a preset name.

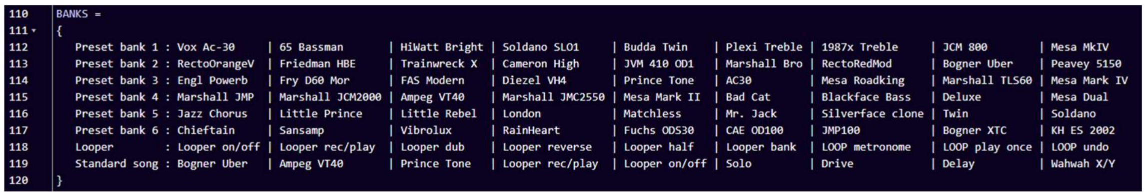
Use '?' to show suggestions, 'Esc' to hide them. Use 'Ctrl-S' to save your work!

```
PRESET Bad Cat =
{
}
```

```
PRESET Bad Cat =
{
    SendMidi Bogner
}
```

Editor with code completion

It also has some clever auto-formatting which results in a spreadsheet like setup overview which stays nicely aligned as you type :



```
BANKS =
{
    Preset bank 1 : Vox Ac-30     | 65 Bassman      | HiWatt Bright | Soldano SLO1     | Budda Twin     | Plexi Treble | 1987x Treble      | JCM 800        | Mesa MkIV
    Preset bank 2 : RectoOrangeV  | Friedman HBE    | Trainwreck X  | Cameron High     | JVM 410 OD1    | Marshall Bro | RectoRedMod       | Bogner Uber    | Peavey 5150
    Preset bank 3 : Engl Powerb   | Fry D60 Mor     | FAS Modern    | Diezel VH4       | Prince Tone    | AC30         | Mesa Roadking     | Marshall TLS60 | Mesa Mark IV
    Preset bank 4 : Marshall JMP  | Marshall JCM2000 | Ampeg VT40   | Marshall JMC2550 | Mesa Mark II   | Bad Cat      | Blackface Bass    | Deluxe         | Mesa Dual
    Preset bank 5 : Jazz Chorus   | Little Prince   | Little Rebel  | London           | Matchless      | Mr. Jack     | Silverface clone  | Twin           | Soldano
    Preset bank 6 : Chieftain     | Sansamp         | Vibrolux      | RainHeart        | Fuchs ODS30    | CAE OD100    | JMP100            | Bogner XTC     | KH ES 2002
    Looper        : Looper on/off | Looper rec/play | Looper dub    | Looper reverse   | Looper half    | Looper bank  | LOOP metronome    | LOOP play once | LOOP undo
    Standard song : Bogner Uber   | Ampeg VT40      | Prince Tone   | Looper rec/play  | Looper on/off  | Solo         | Drive             | Delay          | Wahwah X/Y
}
```

A UnO2 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.

Each UnO2 preset, effect or trigger can contain any combination of different types of commands:

- MIDI commands like *ProgChange, CtrlChange, NoteOn, NoteOff, MIDIStart, MIDIContinue, MIDIStop, 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
- 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 UnO2 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.

Important remark :

The UnO2 setup structure was borrowed from “big brother” TinyBox : a hardware extension for the FCB1010 using the same setup structure and programming language. Therefore the overview above mentions huge upper limits like 200 banks, 1000 presets, 1000 effects, unlimited MIDI commands... However the size of a UnO2 setup is heavily restricted by the limited available storage space in the FCB1010 (2048 bytes, compared to the 262000 bytes available in the TinyBox). So although the setup structure has very high built-in upper limits, the FCB1010 storage space will limit the actual size of a UnO2 setup.

Example 1 : structure of a typical UnO2 setup

```
/* Below you see the general structure of a UnO2 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…
}


// 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 FCB1010 is powered :

INIT_FCB =
{
   Pedal 1 = volume
   Pedal 2 = wah
}
```

```
// It’s also possible to define MIDI commands to be sent when a certain bank
// is activated :

INIT_BANK Preset bank 3 =
{
   SendMidi Profiler ProgChange 125
   SendMidi Helicon ProgChange 3
   SwitchOn Chorus
   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 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 MIDIStart
   SendRealtime MIDIContinue
   SendRealtime MIDIStop
   SendRealtime SystemReset

   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.
```

Example 4 : using variables

```
// The use of 'variables' is something very common in programming languages.
// It adds huge possibilities to the UnO2 setup.

// 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
   }

   // You can also use 2 predefined “constants” to check on the currently active bank or preset:

   if(#CURRENT_BANK == LooperControl)
   {
     // ...
   }
   if(#CURRENT_PRESET == Vox Ac-30)
   {
     // ...
   }
}
```

The UnO2 programming language

The following pages describe in detail the syntax to be used for creating a UnO2 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. Define user friendly MIDI channel names
4. Optionally define commands to be sent when powering the unit
5. Optionally define commands to be sent when selecting each 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.

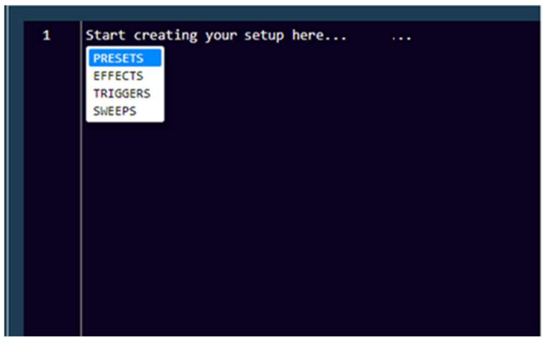
- a **sweep** is a very specific type of preset. You cannot link it to any footswitch, instead you link it to one of the two expression pedals of the FCB1010. The sweep content will specify which continuous MIDI commands each of the pedals will send when moving them – it will typically be ControlChange commands for modifying volume, expression, or continuous effects like wah or whammy. The fact that a UnO2 setup provides "sweep" elements allows you to easily modify the expression pedal behavior depending on the currently active preset for instance.

SYNTAX :

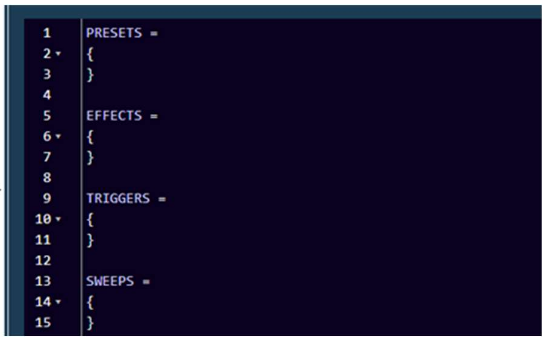
```
PRESETS =
{
   [preset name]
   …
}

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 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 :

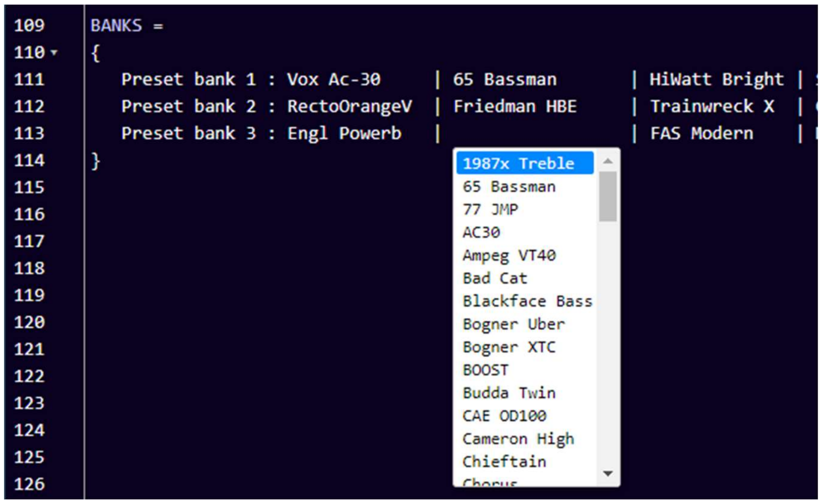


```
Start creating your setup here...     ...
```



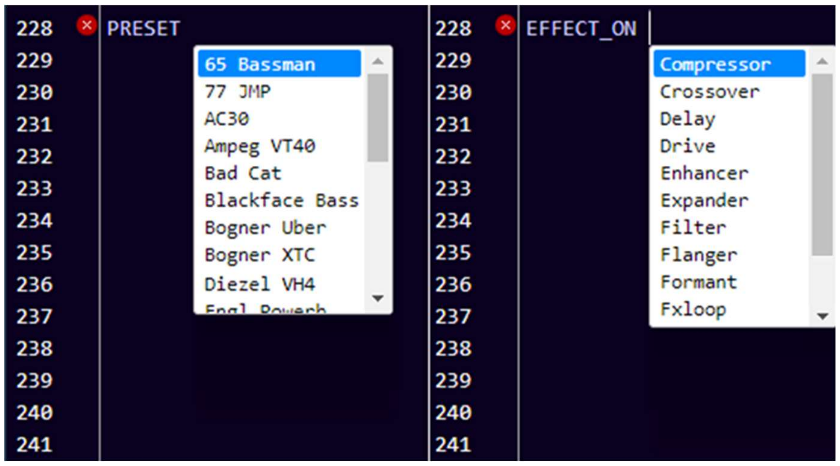
```
PRESETS =
{
}

EFFECTS =
{
}

TRIGGERS =
{
}

SWEEPS =
{
}
```

The 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 :



```
BANKS =
{
    Preset bank 1 : Vox Ac-30    | 65 Bassman    | HiWatt Bright |
    Preset bank 2 : RectoOrangeV | Friedman HBE  | Trainwreck X  |
    Preset bank 3 : Engl Powerb  |               | FAS Modern    |
}
```

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 :



```
PRESET
```

```
EFFECT_ON
```

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.

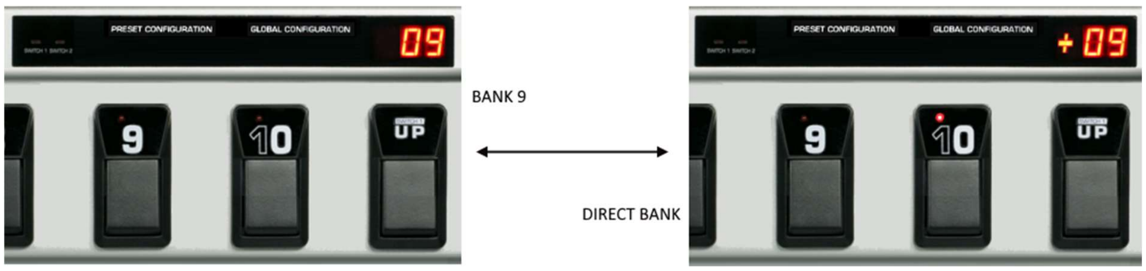
2. Defining the bank structure

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 UnO2 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. In theory 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 :

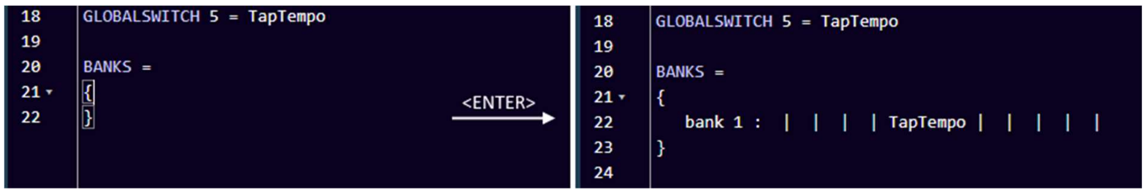
```
GLOBALSWITCH [1…10] = [presetname]

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

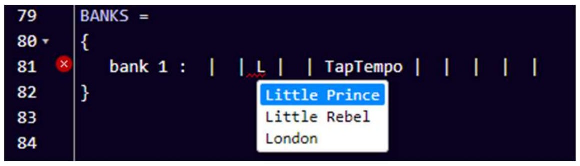
Even without using the "Direct Bank" structure you can have one or a few effects available in all banks, simply by linking the same preset or effect to the same switch in each bank. In order to make that easy for you, you can add one or several "GLOBALSWITCH" definitions prior to the BANKS specification. 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 :


```
GLOBALSWITCH 5 = TapTempo

BANKS =
{
}
```

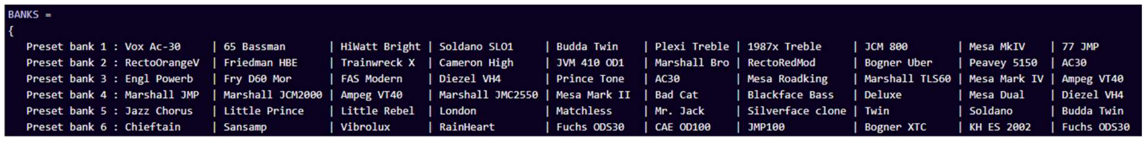
```
GLOBALSWITCH 5 = TapTempo

BANKS =
{
   bank 1 :  |  |  |  | TapTempo |  |  |  |  |
}
```

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 :



```
BANKS =
{
   bank 1 :  |  | L |  | TapTempo |  |  |  |  |
}
```

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:



```
BANKS =
{
   Preset bank 1 : Vox Ac-30    | 65 Bassman      | HiWatt Bright | Soldano SLO1    | Budda Twin   | Plexi Treble | 1987x Treble     | JCM 800        | Mesa MkIV    | 77 JMP
   Preset bank 2 : RectoOrangeV | Friedman HBE    | Trainwreck X  | Cameron High    | JVM 410 OD1  | Marshall Bro | RectoRedMod      | Bogner Uber    | Peavey 5150  | AC30
   Preset bank 3 : Engl Powerb  | Fry D60 Mor     | FAS Modern    | Diezel VH4      | Prince Tone  | AC30         | Mesa Roadking    | Marshall TLS60 | Mesa Mark IV | Ampeg VT40
   Preset bank 4 : Marshall JMP | Marshall JCM2000 | Ampeg VT40    | Marshall JMC2550 | Mesa Mark II | Bad Cat      | Blackface Bass   | Deluxe         | Mesa Dual    | Diezel VH4
   Preset bank 5 : Jazz Chorus  | Little Prince   | Little Rebel  | London          | Matchless    | Mr. Jack     | Silverface clone | Twin           | Soldano      | Budda Twin
   Preset bank 6 : Chieftain    | Sansamp         | Vibrolux      | RainHeart       | Fuchs ODS30  | CAE OD100    | JMP100           | Bogner XTC     | KH ES 2002   | Fuchs ODS30
```

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 FCB1010, 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_BANKS

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 preset contents

SYNTAX :

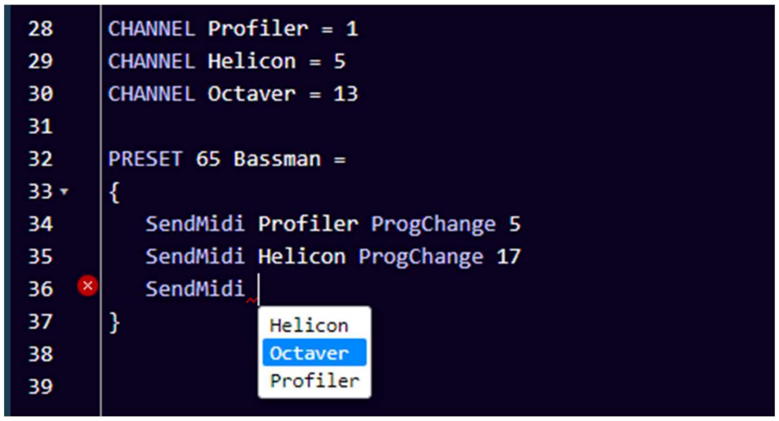
```
CHANNEL channelname = [1…16]
VAR $intvarname     = [0…127]      // up to 128 int vars
VAR $boolvarname    = [true/false] // up to 256 bool vars
VAR $stringvarname  = “any string” // up to 256 string vars

INIT_FCB                    = … // single command,
                                // or list of commands between curly braces

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.

3.1. 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 the 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 will those names each time a MIDI channel needs to be specified :



```
CHANNEL Profiler = 1
CHANNEL Helicon = 5
CHANNEL Octaver = 13

PRESET 65 Bassman =
{
   SendMidi Profiler ProgChange 5
   SendMidi Helicon ProgChange 17
   SendMidi 
}
```

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.

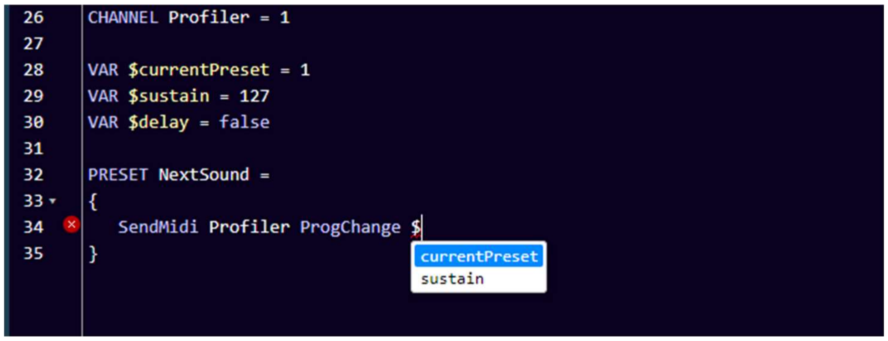
3.2. Defining data variables

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 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 :



```
CHANNEL Profiler = 1

VAR $currentPreset = 1
VAR $sustain = 127
VAR $delay = false

PRESET NextSound =
{
   SendMidi Profiler ProgChange $
}
```

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 specifying 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.

3.3. Defining the FCB1010 initial state

You might want to initialize some global settings right after powering up the FCB1010. A typical example for this could be for instance the default behavior of the FCB1010 expression pedals. The commands to run during FCB1010 initialization can be specified using following syntax :

```
INIT_FCB =
{
      // 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 FCB1010. It will be clear that in that case it is necessary to power the FCB1010 as last component in your rig, or else to do a quick power cycle of the FCB1010 once all other gear is connected and up and running.

3.4. Defining bank initialization

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

The format for specifying bank initialization commands is as follows :

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

3.5. 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
```

3.6. 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
}
```

3.7. 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
}
```

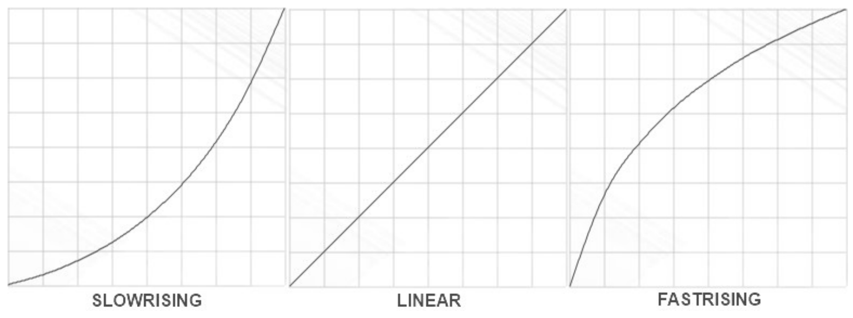
3.8. Defining the sweep contents

A “sweep” might be a less intuitive concept in the UnO2 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
```

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:



4. 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 UnO2 firmware 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 UnO2 setup :

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

4.1. 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 FCB1010 initialization mentioned before :

```
INIT_FCB =
{
      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 UnO2 firmware you can obtain this same behavior with the regular FCB1010 expression pedals!

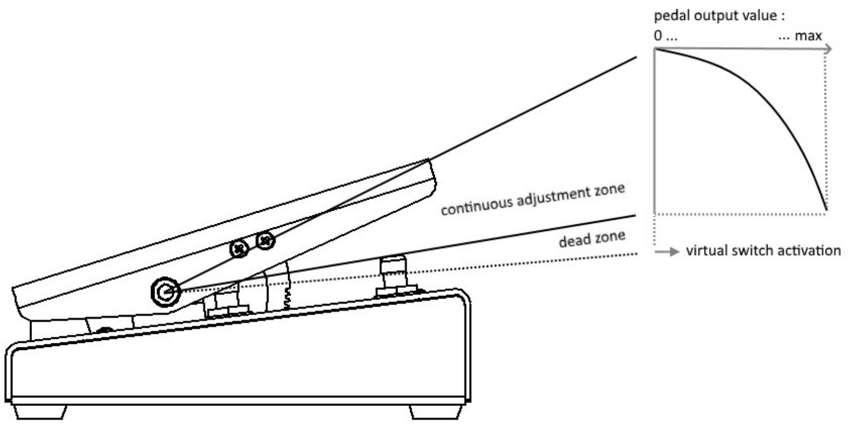
Once you assign a preset or effect to “virtual tip switch 1”, you can trigger that preset by pushing expression pedal 1 all the way down. You could for instance link an effect to the tip switch which toggles the expression pedal behavior between 2 functions. No need to sacrifice a separate footswitch 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_FCB =
{
      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 firmware will automatically introduce a small “dead zone” in the pedal range, so that you can use the full adjustment range of the pedal withouth 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.



4.2. 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 2
```

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 FCB1010 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 editor in UnO2 ControlCenter 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.

4.3. Navigation and Remote Control

SYNTAX :

```
GotoBank bankname
```

Use this command in a preset content to directly jump to a specific bank with one foot click, instead of using the up/down switches to scroll through the complete bank list.

SYNTAX :

```
REMOTE_CONTROL_CHANNEL = nn
```

When you add this command prior to the other MIDI channel definitions of your setup, you can remotely control the FCB1010 from another MIDI device, connected to MIDI IN.

Following ControlChange and ProgramChange commands can be sent on the specified remote control MIDI channel to remotely do “switch presses” and “pedal movements” on the FCB1010 :

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

4.4. 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  SystemReset
```

As the FCB1010 is in the first place a MIDI controller, “MIDI command” will without doubt be the most used command type in a UnO2 setup. One of the UnO2 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 only limited by the FCB1010 storage size, however in a realistic use case this command will be used to send rather short SysEx messages for effect or preset parameter tweaking. 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.

4.5. 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 UnO2 firmware 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 3.8 for more details)

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”.

4.6. 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 firmware cannot process any switch press or pedal movement or do anything else while it is executing the `Wait` command.

4.7. 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.

4.8. 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($intvarname) {          switch($stringvarname) {
   case[0…127]:                   case "any string":
      …                              …
      break                          break
   …                              …
   default:                       default:
      …                              …
      break                          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.

4.8.1. 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) || … )
```

4.8.2. 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 UnO2 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
```

4.8.3. while statement

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

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

4.8.4. 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($currentPreset) {
      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($currentSong) {
      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
}
```

4.8.5. conditions using predefined variables

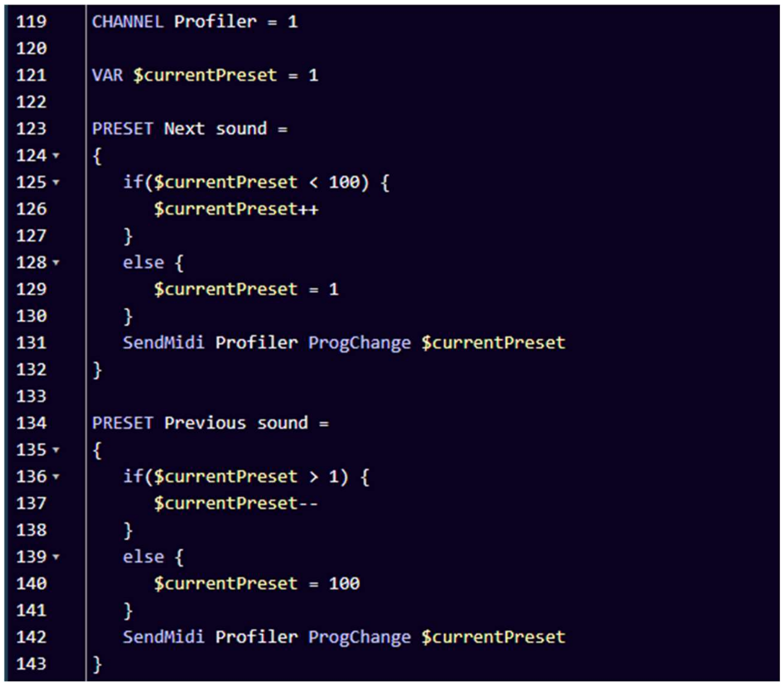
The UnO2 programming language provides a few predefined variables, which can be used in conditional commands. Those predefined variables are :

```
#CURRENT_BANK
#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 active song name yourself :

```
switch(#CURRENT_BANK) {
      …
}

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 :



```
CHANNEL Profiler = 1

VAR $currentPreset = 1

PRESET Next sound =
{
   if($currentPreset < 100) {
      $currentPreset++
   }
   else {
      $currentPreset = 1
   }
   SendMidi Profiler ProgChange $currentPreset
}

PRESET Previous sound =
{
   if($currentPreset > 1) {
      $currentPreset--
   }
   else {
      $currentPreset = 100
   }
   SendMidi Profiler ProgChange $currentPreset
}
```

And now it's time to have fun !

APPENDIX : UnO2 programming language reference

Comments :

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

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

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

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

EFFECTS =
{
   [effect name]
   …
}

TRIGGERS =
{
   [trigger name]
   …
}

SWEEPS =
{
   [continuous control name]
   …
}


NO_UPDOWN_SWITCHES

GLOBALSWITCH [1…10] = [presetname]

USE_DIRECT_BANK

BANKS =
{
   [bank name] : [preset name] | [preset name] | … | [preset name]
   [bank name] : [preset name] | [preset name] | … | [preset name]
   …
}
```

Defining preset content :

```
REMOTE_CONTROL_CHANNEL = [1…16]

CHANNEL channelname = [1…16]

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

INIT_FCB                = … (single command, or list of commands between curly braces)
INIT_BANK bank          = …   "

PRESET preset           = …   "
EFFECT_ON effect        = …   "
EFFECT_OFF effect       = …   "
TRIGGER_CLICK trigger   = …   "
TRIGGER_RELEASE trigger = …   "
SWEEP sweep             = … (single continuous control command)
```

Commands :

Dynamic switch and pedal assignment commands :

```
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"
```

Effect activation commands

```
SwitchOn    effectname
SwitchOff   effectname
SendTrigger triggername
SendPedal   [1…2]
```

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
```

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)
```

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*) {
   …
}
else if (condition*) {
   …
}
else {
   …
}

while (condition*) {
   …
}

switch($intvarname) {
   case[0…127]:
      …
      break
   …
   default:
      …
      break
}

switch($stringvarname) {
   case "any string":
      …
      break
   …
   default:
      …
      break
}

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

APPENDIX : Factory reset, self-test and pedal calibration

The UnO2 firmware contains the same self-test and expression pedal calibration procedures as the original Behringer firmware. Therefore calibration instructions can be found in the Behringer manual or online by googling for “FCB1010 calibration”.

- Self test :	keep footswitches 1+3 pressed during startup
- Calibration :	keep footswitches 1+5 pressed during startup

The factory resets available in the Behringer firmware are ***no longer*** available :

- V-AMP factory preset :	keep footswitches 1+6 pressed during startup
- Behringer guitar amp factory preset :	keep footswitches 1+7 pressed during startup
- BASS V-AMP factory preset :	keep footswitches 1+8 pressed during startup

Instead, there is a setup reset which clears the current UnO2 setup, and a WIFI reset which resets the IP address, SSID and password of the embedded WIFI access point to its default values :

- Setup reset :	keep footswitches 1+4 pressed during startup
- WIFI reset :	keep footswitches 1+2 pressed during startup