



UnO2 firmware & UnO2 ControlCenter

User Manual

Contents

Introduction: what is the UnO2 firmware ?	3
Getting started	5
1. Install the UnO2 chip	5
2. Install the UnO2 ControlCenter software.....	5
3. Connect FCB1010 and computer.....	5
4. Register the firmware online.....	5
5. Create a first setup	6
6. Download the setup to your FCB1010	6
7. Test your setup.....	6
The UnO2 ControlCenter menus.....	7
1. The File menu	8
2. The Edit menu (Mac only)	8
3. The FCB1010 menu.....	8
3.1. Connect.....	8
3.2. Send setup	9
4. The Setup menu	9
4.1. Select MIDI ports	9
4.2. Register UnO2 firmware.....	10
5. The Help menu	11
5.1. MIDI monitor	11
5.2. User manual	11
5.3. UnO2 website	11
The UnO2 ControlCenter MIDI monitor	12
The UnO2 setup structure	14
Example 1 : structure of a typical UnO2 setup.....	16
Example 2 : sending MIDI messages	18
Example 3 : programming expression pedals.....	19
Example 4 : using variables	20
Example 5 : using conditional commands	21
The UnO2 programming language	22
0. Comments	22
1. Defining preset, effect, trigger and sweep names	23
2. Defining the bank structure	26
3. Defining preset contents	30

3.1.	Defining MIDI channels	31
3.2.	Defining data variables.....	32
3.3.	Defining the FCB1010 initial state	33
3.4.	Defining bank initialization.....	33
3.5.	Defining the preset contents.....	34
3.6.	Defining the effect contents.....	34
3.7.	Defining the trigger contents	35
3.8.	Defining the sweep contents.....	36
4.	The command set	37
4.1.	Switch and pedal assignment commands	38
4.2.	Effect activation, relay activation and expressionpedal activation commands.....	40
4.3.	Navigation and Remote Control.....	41
4.4.	MIDI commands	42
4.5.	Continuous Control commands.....	43
4.6.	Delay command.....	44
4.7.	Variable commands.....	45
4.8.	Conditional commands.....	46
4.8.1.	The condition syntax	47
4.8.2.	if...then...else statements	48
4.8.3.	while statement.....	49
4.8.4.	switch statements	49
4.8.5.	conditions using predefined variables	50
APPENDIX : UnO2 programming language reference		52
APPENDIX : Factory reset, self-test and pedal calibration		55

Document versions

Version 1.4	04/05/2024	WaitWithoutBlocking command : info added
Version 1.3	01/03/2022	Extra functionality added in firmware v.1.3
Version 1.2	20/08/2020	Support for extra SysCommon and SysRealtime messages added SwitchOn/SwitchOff commands no longer restricted to be used in presets only – effect activation infinite loop detection added
Version 1.1	01/07/2020	Feature added : use of “while” loops Appendix about selftest & calibration added
Version 1.0	20/05/2020	Initial release

Introduction: what is the UnO2 firmware ?

In order to understand the philosophy behind the UnO2 firmware, a little word of history.

For 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 over the past years will testify, the UnO firmware has brought several enhancements to the original FCB1010. Apart from some important bugfixes (like correcting errors in the MIDI merge functionality) we added a few highly requested features, like a “real stompbox mode”.

Even with the UnO firmware installed, the FCB1010 still has a few major restrictions, mainly caused by the very limited setup storage capacity of the unit. Each preset can send a maximum of 8 different MIDI messages (5xProgramChange, 2xControlChange, 1xNoteOn), while each of these 8 messages share the same MIDI channel for all presets.

In order to overcome these limitations, our first approach was to create a hardware extension box which contains much more setup storage (128 times the FCB1010 storage, to be precise) and thus allows much more extended setups, along with more flexibility in those setups. We called it the TinyBox <https://www.tinybox.rocks/>

After finalizing the TinyBox design and its accompanying setup editor, we believed that the huge flexibility of the TinyBox approach might work in much smaller setups too, setups which could even fit inside the tiny FCB1010 setup memory. Instead of the rigid 10x10 preset structure, with 8 messages per preset, you might prefer to have just a few banks filled with much more powerful presets, and laid out in a flexible way (banks with all presets, banks with all stompboxes, or any customizable mix of both).

This brought us to the UnO2 firmware. It’s not an extension of the well known UnO firmware. It’s rather a shrunk down version of the TinyBox, which fits inside the FCB1010 EPROM. Along with the firmware comes an adapted version of the TinyBox editor, which was designed to be as user friendly as possible. It takes a completely different approach compared to the FCB/UnO ControlCenter software. It allows you to build a setup by writing it down as text in a dedicated text editor which offers auto-completion.

Next page shows a comparison overview of UnO firmware vs. UnO2 firmware vs. TinyBox. Some of the mentioned features will require a more detailed explanation, which you will find in further chapters of this manual.

	UnO firmware	UnO2 firmware	TinyBox
# banks	10	Up to 200 (theoretically *)	Up to 200
# presets	100	Up to 1000 (theoretically *)	Up to 1000
# messages/preset	8	Unlimited (theoretically *)	Unlimited
# MIDI channels	8 (global)	16 (for each message)	16 (for each message)
MIDI msg types	PC, CC, Note	All	All
MIDI Clock	-	Start, Continue, Stop	Start, Continue, Stop, Clock
Expr.pedal msg	CC	CC, AfterTouch, PitchBend	CC, AfterTouch, PitchBend
Pedal sweep curve	Linear	Linear, FastRising, SlowRising	Linear, FastRising, SlowRising
Bank layout possibilities	10 x 10 presets or 19 x 5 presets + 5 global stompboxes	Any mix of presets, stompboxes, momentary effects for each bank	Any mix of presets, stompboxes, momentary effects for each bank
Direct bank	-	+	+
Programmable delay	-	+	+
Use of data variables	-	+	+
if..then...else logic	-	+	+
Max. setup size	2048 bytes	2048 bytes	262144 bytes
Setlist support	-	-	+
MIDI filter/router	-	-	+
Built-in MIDI-USB itf	-	-	+
Wireless status display	-	-	+
Phantom power	-	-	+

(*) the total number of banks or presets that can be programmed is limited by the 2048-byte setup memory, and therefore depends on the content of each of the presets. An online version of the UnO2 ControlCenter editor allows you to create a setup and inspect its size, in order to verify if it fits in the UnO2 based solution, or if it would require the TinyBox hardware extension.

Getting started

1. Install the UnO2 chip

Instructions for replacing the firmware chip can be found in this Behringer manual :

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

2. Install the UnO2 ControlCenter software

The software installer (both Windows and Mac versions) can be downloaded at

<https://www.fcb1010.eu/ccdownload.php>

You will need the registration data mentioned on the chip packaging to access the downloads. Store the package safely, since you will need this registration data later on for downloading software updates or for getting a discount on future firmware update chips!

3. Connect FCB1010 and computer

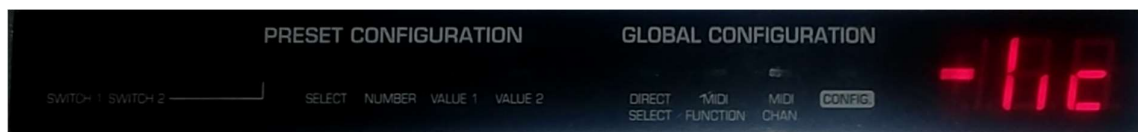
Connect a MIDI cable from the FCB1010 MIDI OUT connector to the MIDI IN connector of your MIDI-USB interface, and a second MIDI cable from the MIDI OUT connector of the MIDI-USB interface to the MIDI IN connector of the FCB1010.

Launch UnO2 ControlCenter and select the MIDI IN and MIDI OUT port you will be using. See also topic 4.1 of the next chapter.

Click the “Connect” command to initiate the 2-way communication between FCB1010 and computer. See topic 3.1 of the next chapter for more details.

4. Register the firmware online

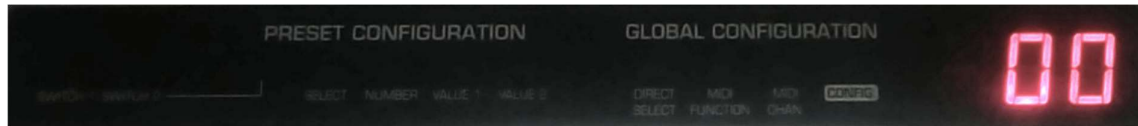
When first powering the FCB1010 after upgrade, the display will show “-lic” :



This means the firmware needs to be registered online.

Make sure your computer is connected to the internet and follow the instructions in topic 4.2 of the next chapter to do the registration.

Once the registration is completed the firmware is unlocked and ready to use :



5. Create a first setup

Before you can create your first setup, you will need to take your time and go through the next chapters of the manual. These explain in detail the UnO2 setup structure, the available UnO2 features, and the “programming language” used to describe your setup.

If you have been using FCB/UnO ControlCenter in the past to create your FCB1010 setups, you can take a jump start by exporting your old setup to UnO2 format. The latest FCB/UnO ControlCenter version has a menu option for this. However, be aware that not all “old” setups will fit in the UnO2 setup structure just like that. If the setup has multiple MIDI messages enabled for all 100 presets, the according UnO2 setup size will probably be too large to fit in the FCB1010 EEPROM. You will need to filter all unused messages first, and then have a look at the different ways of rewriting your setup in a more efficient way, taking advantage of the advanced UnO2 command set.

6. Download the setup to your FCB1010

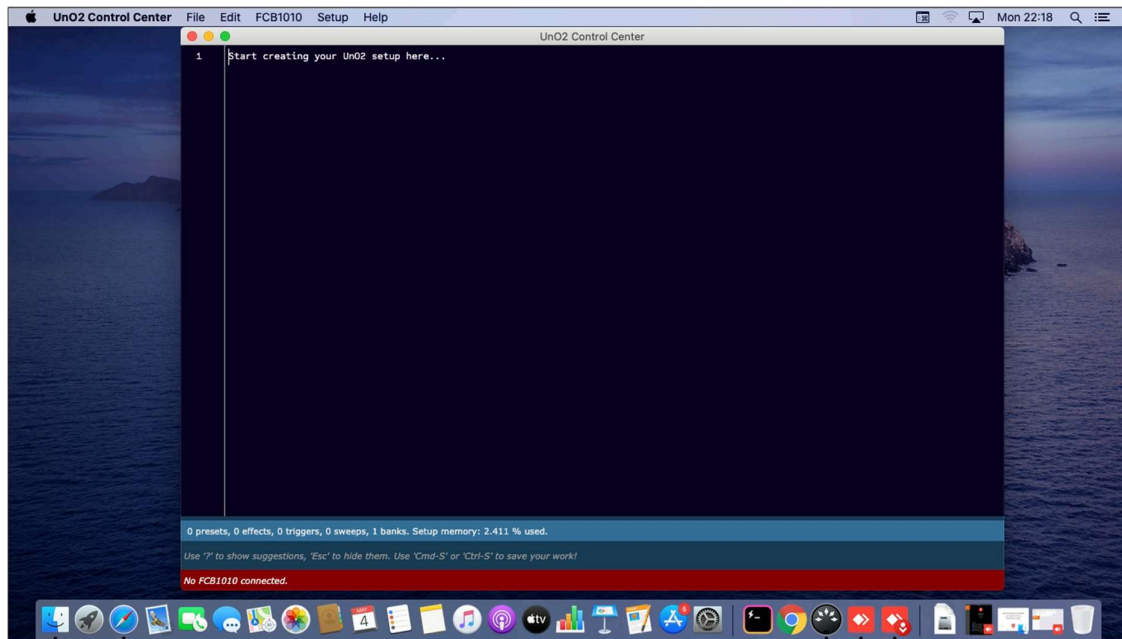
When saving your setup it is automatically compiled into binary data which can be sent to the FCB1010 with a single click. See also topic 3.2 of the next chapter.

7. Test your setup

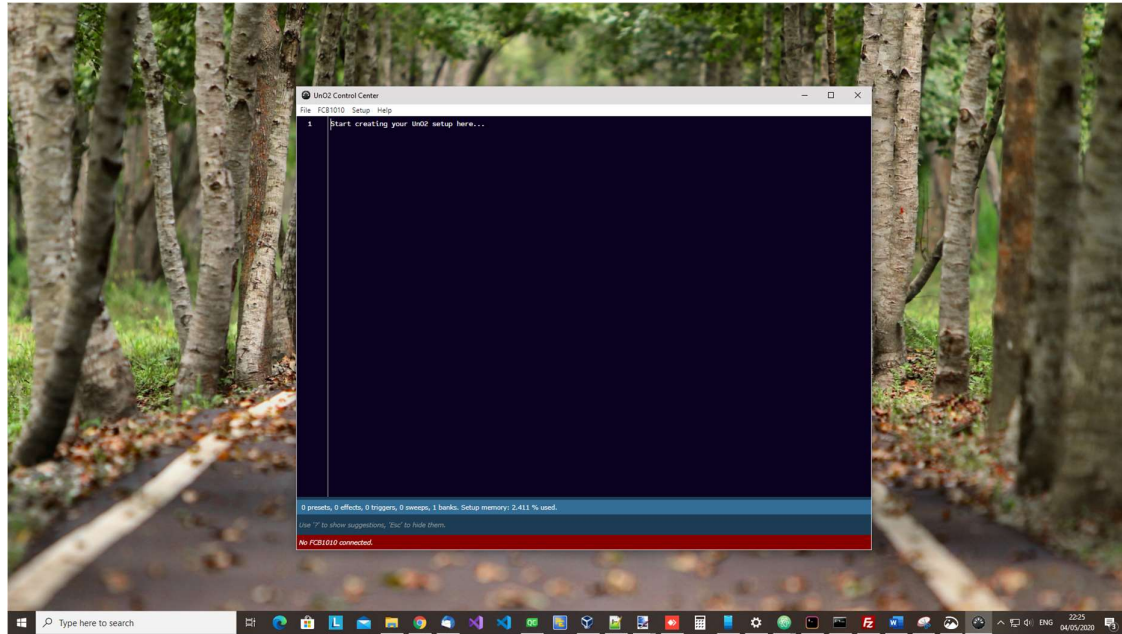
Things are not always working correctly right from the start. If your gear doesn’t behave as expected you might want to inspect the MIDI messages which the FCB1010 sends. UnO2 ControlCenter has a built-in MIDI monitor for that. With this monitor you can get a detailed inside view on all MIDI traffic, and also “manually” send MIDI messages to your gear in order to experiment and find the correct control messages. More details about the MIDI monitor can be found in a next chapter of this manual.

- *Remark* : make sure the FCB1010 expression pedals are correctly calibrated. 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”.

The UnO2 ControlCenter menus



UnO2 ControlCenter on Mac



UnO2 ControlCenter on Windows

1. The File menu

The File menu offers all regular options to create, delete, save or copy setups. While these setup files are stored as user settings “inside” the application, you can very easily store your setup as a text file on USB stick for backup, or share it online, by copy-and-pasting the full text into a regular text editor or notepad application.

2. The Edit menu (Mac only)

This menu is needed on Mac only, just to enable the regular copy-paste keyboard shortcuts in the text editor : Cmd-X, Cmd-C, Cmd-V, Cmd-A for Cut – Copy – Paste – Select All. Also on Windows the standard Ctrl-X, Ctrl-C, Ctrl-V, Ctrl-A keyboard shortcuts are available.

3. The FCB1010 menu

3.1. Connect

The Connect command will create a 2-way connection between computer and FCB1010. Several menu options will remain disabled as long as no connection has been made: sending a setup, registering the firmware or using the MIDI monitor is not possible without FCB1010 connection.

The Connect command itself is disabled as long as no MIDI ports have been configured. See topic 4.1.

A red status bar at the bottom of the application indicates that a connection needs to be made :




No FCB1010 connected.

Once the Connect button is clicked, the status bar turns blue and shows the firmware version of the connected FCB1010 :



An FCB1010 with UnO2 firmware is connected. Firmware version is 1.0

If no connection can be made, a timeout will occur :



A timeout occurred. Please check the MIDI connections.

In that case double check that both MIDI cables are correctly connected between computer and FCB1010 and that the correct MIDI ports have been selected in the settings screen (see topic 4.1)

3.2. Send setup

Click this command to send the current setup to your FCB1010. If the command is disabled, first make a connection with the FCB1010 (see topic 3.1)

Whenever you modify your setup in the text editor, the built-in compiler evaluates the text. Below the text area a status message shows if any error in the text is detected. If there is no error, the built-in compiler turns the text into binary data, and the status bar shows how much of the FCB1010 setup memory is required for the setup. As long as this memory usage stays below 100% you can send the setup to the FCB1010 by clicking the “send setup” command.

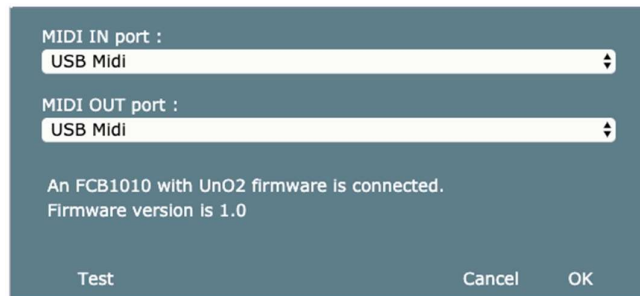
4. The Setup menu

4.1. Select MIDI ports

In order to let UnO2 ControlCenter communicate with the FCB1010, you need to specify which MIDI-USB ports it can use :



You will not be able to confirm your choice unless successful communication through those ports has been verified. Click the Test button. A message will appear when 2-way communication has succeeded:



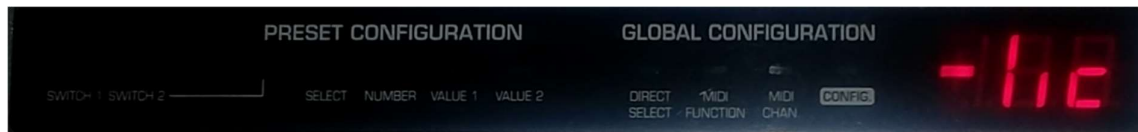
If communication fails, please double check the MIDI connections between MIDI-USB interface and FCB1010.

Selecting the MIDI ports needs to be done just once. Next time you launch UnO2ControlCenter you can go directly to the “Connect” command without selecting the MIDI ports again. Unless the ports are unavailable of course. In that case close ControlCenter, connect the interface, launch ControlCenter and again do the MIDI port selection.

4.2. Register UnO2 firmware

Before you can start using the UnO2 firmware it needs to be registered online. This registration has to be done just once in order to unlock the UnO2 functionality. For this the FCB1010 communication needs to be initiated (by clicking the Connect button – see 3.1), and the editor needs to be connected to the internet in order to talk to the UnO2 license server.

Along with the firmware chip you have received a unique registration code, along with a copy of the personal data entered when placing the order. Copy this data into the registration form, and click “Register”. The firmware serial number doesn’t need to be filled in, it is detected by the software. If all data is correct, the firmware will be unlocked and ready to use.



UnO2 before registration

Fill in the registration data received along with the firmware chip.
(use the exact data as received, with accents and special characters removed!)

Name :

Email :

Code :

Firmware serial number :

Register Close

registration form



UnO2 after registration

5. The Help menu

5.1. MIDI monitor

A built-in MIDI monitor is available for troubleshooting purposes. For more info have a look at the next chapter which covers this MIDI monitor. If the MIDI monitor option is disabled you need to make a connection with the FCB1010 first – see topic 3.1

5.2. User manual

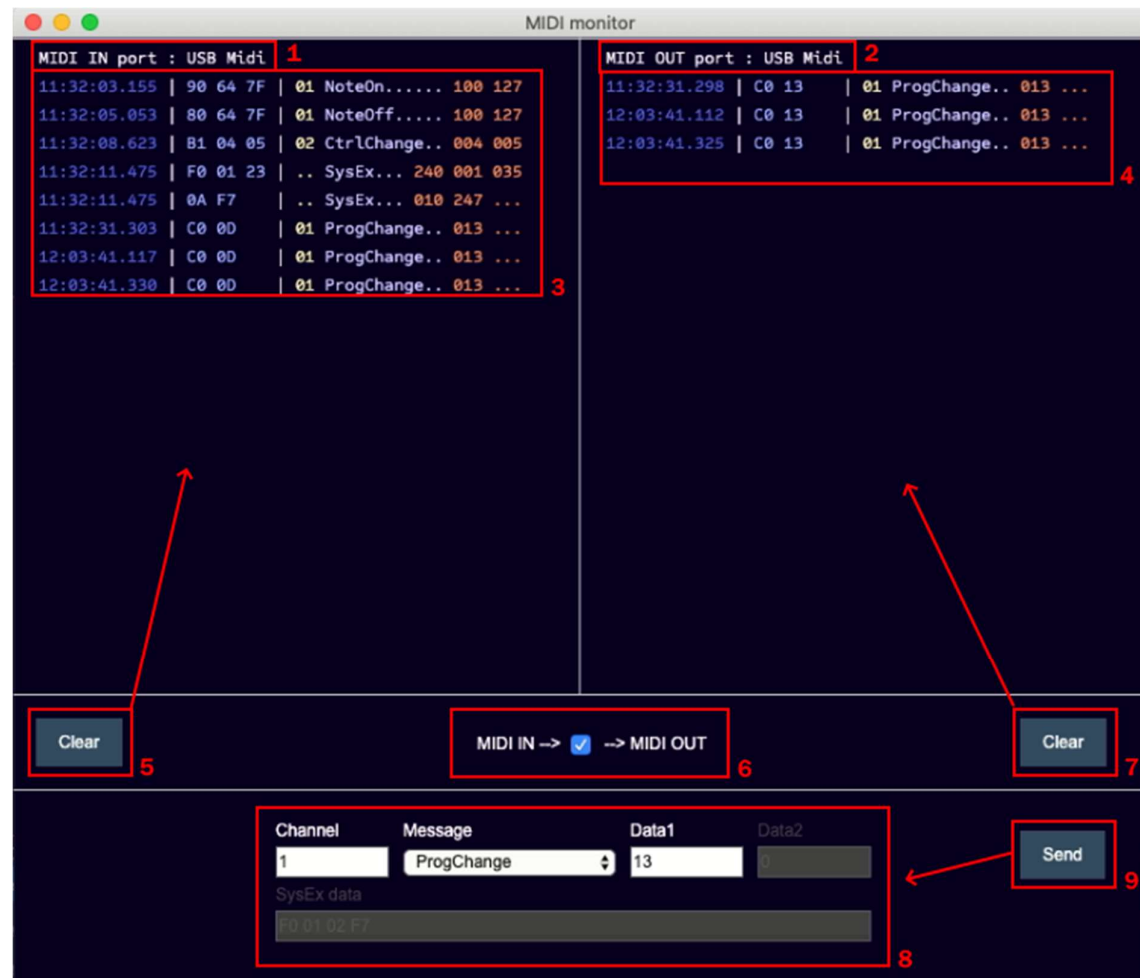
This menu option opens a link to the latest version of this manual.

5.3. UnO2 website

This menu option opens a link to the fcb1010.eu website.

The UnO2 ControlCenter MIDI monitor

Before diving into the details of creating a UnO2 setup, which will take the major part of this manual, let's quickly describe a useful helper tool which is part of UnO2 ControlCenter: the MIDI monitor.



Overview of the MIDI monitor user interface :

- 1 : MIDI IN port in use by ControlCenter
- 2 : MIDI OUT port in use by ControlCenter
- 3 : List of incoming MIDI messages
- 4 : List of outgoing MIDI messages
- 5 : Clear button for the MIDI IN listview
- 7 : Clear button for the MIDI OUT listview
- 6 : MIDI THRU function = forward all incoming MIDI messages to the MIDI OUT port
- 8 : MIDI message selection : specify any MIDI message to be sent to the MIDI OUT port
- 9 : Send button to transmit the selected MIDI message to the MIDI OUT port

The MIDI message list contains following data for each received or 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)

With the “MIDI THRU” function you can forward all incoming MIDI messages to the MIDI OUT port. You do that simply by ticking the checkbox between “MIDI IN -->” and “--> MIDI OUT”.

You will need this “MIDI THRU” functionality if you want to test your current FCB1010 setup. To do so, connect a MIDI cable from FCB1010 MIDI OUT to the computer, and from computer MIDI OUT to your gear. When clicking an FCB1010 footswitch, all transmitted MIDI messages will appear in the MIDI IN message list, and since you forward these to the MIDI OUT port they will also appear in the MIDI OUT message list. If your gear doesn't behave the way you expected and you want to find out which MIDI messages should be sent instead, use the Send button to manually send individual MIDI messages. They will also appear in the MIDI OUT list.

ATTENTION :

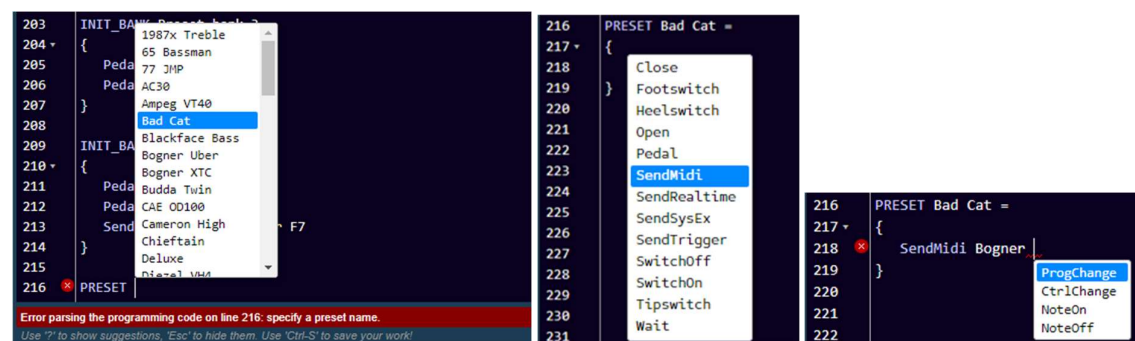
Never activate the “MIDI THRU” function when you have both MIDI cables connected to the FCB1010 (which is the case while doing patchdumps) ! The FCB1010 with UnO2 firmware also has “MIDI THRU” behavior: all messages arriving at the FCB1010 MIDI IN port are automatically forwarded to the MIDI OUT port (except specific SysEx messages like version requests, patchdumps etc. which are processed internally). You will understand that having MIDI THRU functionality both in the FCB1010 and in the MIDI monitor would create an infinite loop: a message sent by the FCB would be forwarded by the MIDI monitor and returned to the MIDI IN port of the FCB, where it is again forwarded to its MIDI OUT port, etc... Messages will start running around forever, and the message lists of the MIDI monitor will quickly fill up with an endless message stream. When you notice this, just untick the MIDI THRU checkbox: the loop will be interrupted and everything should return to normal. Use the Clear buttons to empty the message lists.

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:



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 :

110	BANKS =									
111	{									
112	Preset bank 1 :	Vox Ac-30	65 Bassman	HiWatt Bright	Soldano SLO1	Budda Twin	Plexi Treble	1987x Treble	JCM 800	Mesa MkIV
113	Preset bank 2 :	RectoOrangeV	Friedman HBE	Trainmeck X	Cameron High	JVM 410 OD1	Marshall Bro	RectoRedMod	Bogner Uber	Peavey S150
114	Preset bank 3 :	Engl Powerb	Fry D60 Mor	FAS Modern	Diziel VM4	Prince Tone	AC30	Mesa Roadking	Marshall TL560	Mesa Mark IV
115	Preset bank 4 :	Marshall JMP	Marshall JCM2000	Ampeg VT40	Marshall JMC250	Mesa Mark II	Bad Cat	Blackface Bass	Deluxe	Mesa Dual
116	Preset bank 5 :	Jazz Chorus	Little Prince	Little Rebel	London	Matchless	Mr. Jack	Silverface clone	Twin	Soldano
117	Preset bank 6 :	Chieftain	Sansamp	Vibrolux	RainHeart	Fuchs OD530	CAE OD100	JMP100	Bogner XTC	KH ES 2002
118	Looper :	Looper on/off	Looper rec/play	Looper dub	Looper reverse	Looper half	Looper bank	Looper metronome	LOOP play once	LOOP undo
119	Standard song :	Bogner Uber	Ampeg VT40	Prince Tone	Looper rec/play	Looper on/off	Solo	Drive	Delay	Wahwah X/Y
120	}									

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
    }
}
```

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. Optionally define commands to be sent when powering the unit
4. Optionally define commands to be sent when selecting each bank
5. 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]
    ...
}

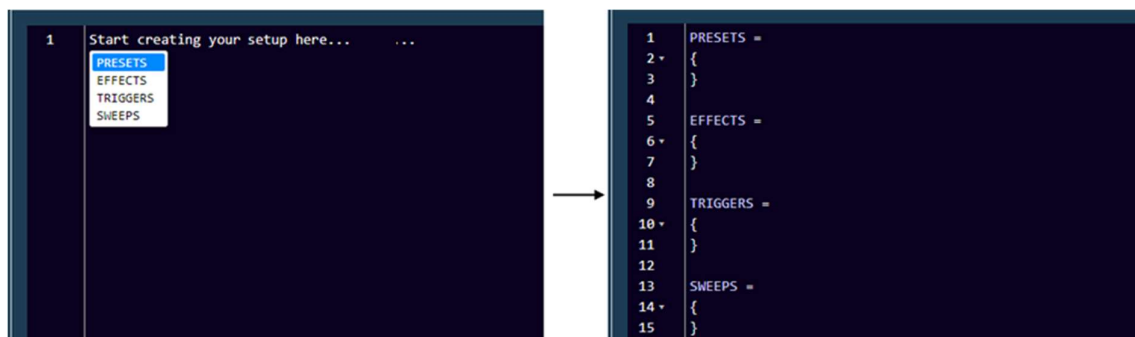
EFFECTS =
{
    [effect name]
    ...
}

TRIGGERS =
{
    [trigger name]
    ...
}

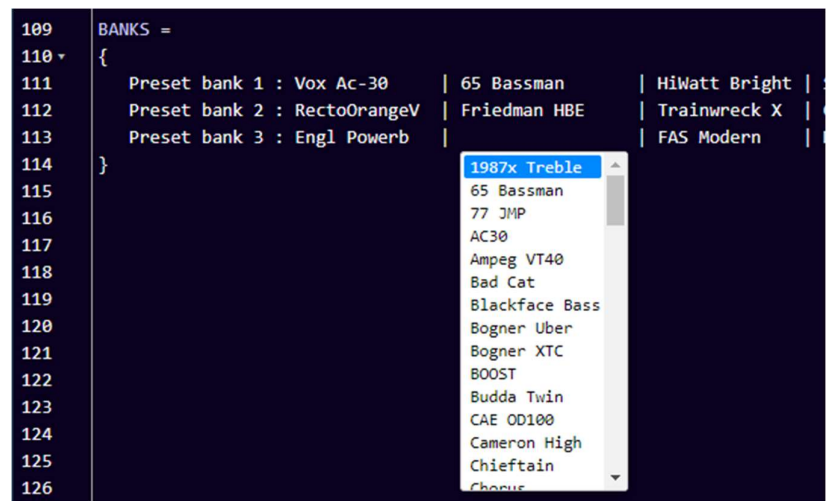
SWEEPS =
{
    [continuous control name]
    ...
}
```

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

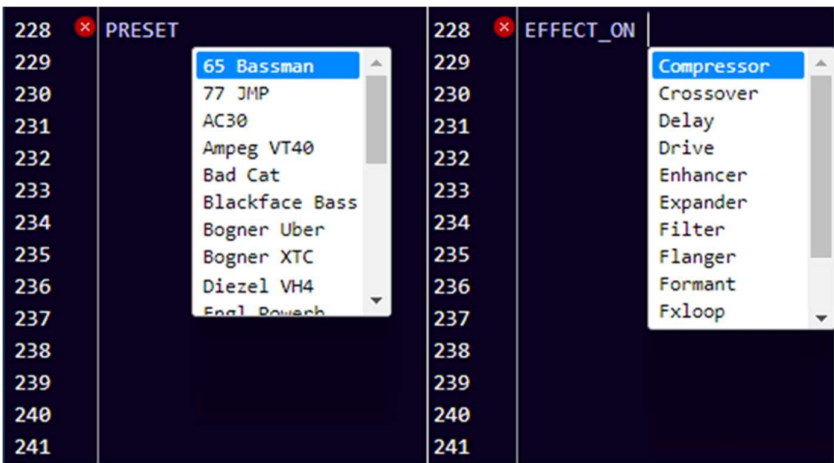
The autocomplete or “intellisense” functionality of the UnO2ControlCenter editor helps you setting up this initial setup structure : after creating a new setup, type ‘?’ to get a dropdownlist of available commands. Clicking <ENTER> 4 times on the proposed command list will give you the 4 empty lists which you can now start filling with preset names :



UnO2ControlCenter uses these name lists in order to help you when specifying further parts of the setup. For instance when defining the bank layout (as explained later on in this chapter), the editor will automatically propose a list of available presets to choose from :



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



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

2. Defining the bank structure

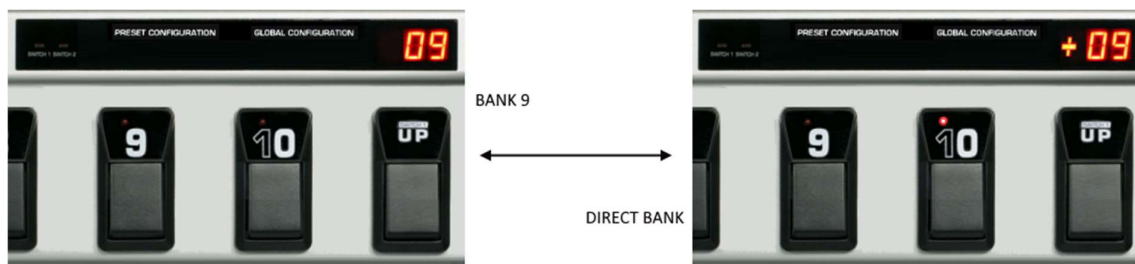
Once you have listed all presets, effects and triggers to be used in your setup, you can start organizing them in banks.

By default a 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 :

<pre>18 GLOBALSWITCH 5 = TapTempo 19 20 BANKS = 21 { 22 }</pre>	<p><ENTER></p> 	<pre>18 GLOBALSWITCH 5 = TapTempo 19 20 BANKS = 21 { 22 bank 1 : TapTempo 23 } 24</pre>
---	--	---

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



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

```
BANKS =
{
  Preset bank 1 : Vox Ac-30 | 65 Bassman | HiWatt Bright | Soldano SL01 | 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 OD530 | CAE OD100 | JMP100 | Bogner XTC | KH ES 2002 | Fuchs OD530
}
```

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.

Remark :

This option is available in firmware v.1.3 or higher

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)
```

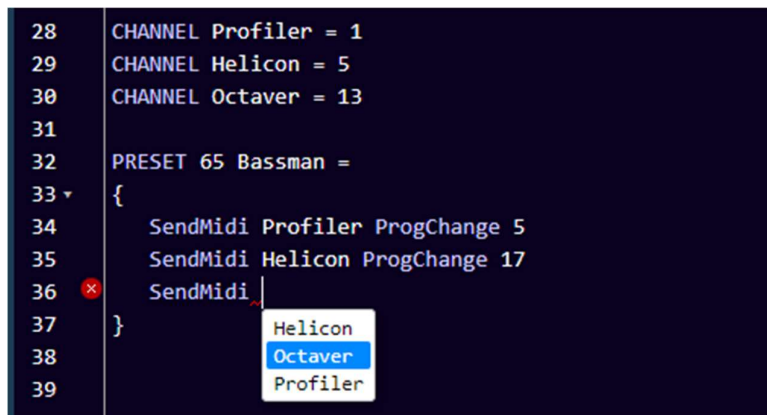
Now starts the main part of the setup, containing all details of which MIDI commands each preset will send. The next sub-topics cover each of the setup parts listed above. The actual format of the commands which will be contained in each of the preset definitions is described in later chapters.

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 with those names each time a MIDI channel needs to be specified :

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

A screenshot of a MIDI setup editor with a dark background. It shows a list of MIDI channels defined at the top: 'CHANNEL Profiler = 1', 'CHANNEL Helicon = 5', and 'CHANNEL Octaver = 13'. Below these, a preset named 'PRESET 65 Bassman =' is shown with a list of MIDI commands: 'SendMidi Profiler ProgChange 5', 'SendMidi Helicon ProgChange 17', and 'SendMidi'. A dropdown menu is open next to the 'SendMidi' command on line 36, showing three options: 'Helicon', 'Octaver' (which is highlighted in blue), and 'Profiler'. The lines are numbered from 28 to 39 on the left side.

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.

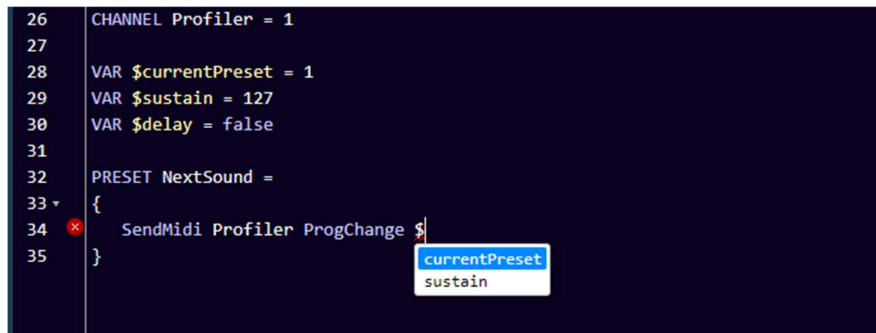
3.2. Defining data variables

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 :



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.

(*) available in firmware v.1.3 or higher

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  
    // ...  
}
```

Remark :

From firmware version v.1.3 on, it is possible to specify bank initialization commands for the Direct Bank too.

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.

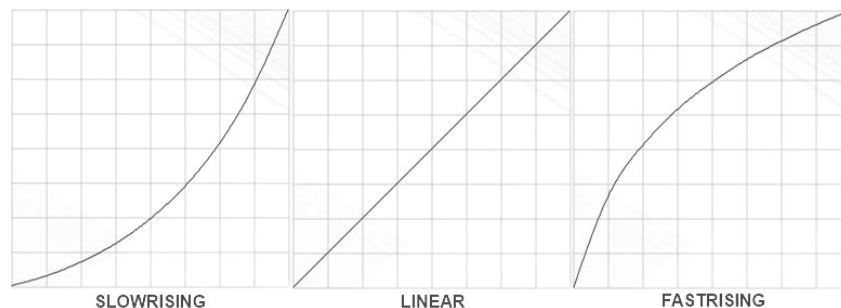
The syntax for defining sweep contents will look familiar by now :

```
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!

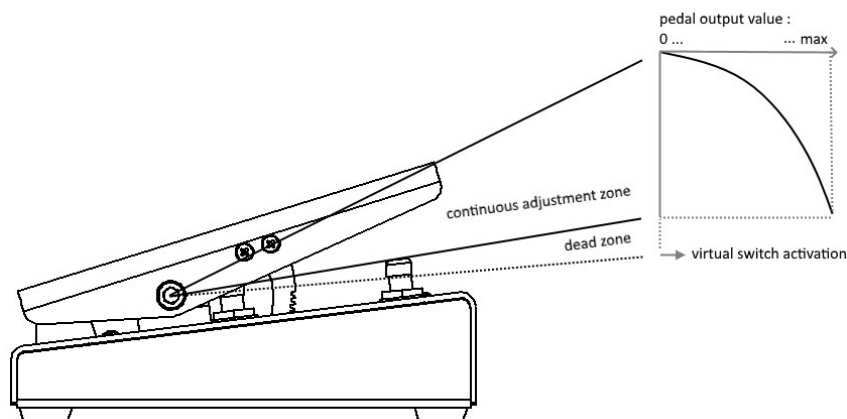
Similarly a “virtual heel switch” can be defined. This one is triggered by moving the expression pedal all the way up (heel-down). Functionality which probably doesn’t even exist with “real” footswitches. A great example of how to make use of this feature is the following setup fragment, which automatically 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 without automatically triggering the tip (or heel) switch. In this case it is very helpful to add some resistance to the pedal at a point between full adjustment range and switch engagement. This can be done by sticking a piece of foam underneath the pedal, or even by mounting a real (unconnected) footswitch.



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.

(*) available in firmware v.1.3 or higher

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
```

These commands are available in firmware v.1.3 or higher

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”.

(*) available in firmware v.1.3 or higher

4.6. Delay command

SYNTAX :

```
Wait value  
WaitWithoutBlocking 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.

A `WaitWithoutBlocking` command was added in order to enable more advanced features, like triggering different commands depending on how long a switch is being pressed, or being able to detect a switch release during the wait.

However, please be aware that under certain conditions this command might introduce an instability in the FCB1010: if too many commands have to be processed during the non-blocking wait the FCB1010 microcontroller might be unable to handle the amount of logic, and the system might even crash (for the techies : a stack overflow may occur...)

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) {
    case[0...127]:
        ...
        break
    ...
    default:
        ...
        break
}

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

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

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

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

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") {
    ...
}
```

This functionality is available in firmware v.1.3 or higher

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

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



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)
WaitWithoutBlocking value ( " )
```

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 factory reset possibility, which clears the current UnO2 setup :

- Factory reset : keep footswitches 1+4 pressed during startup