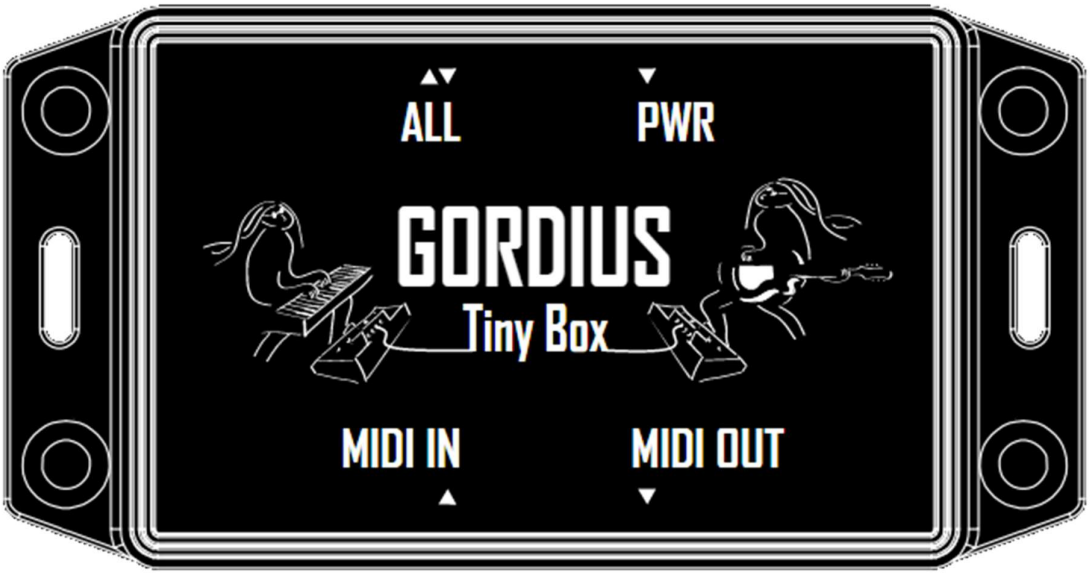
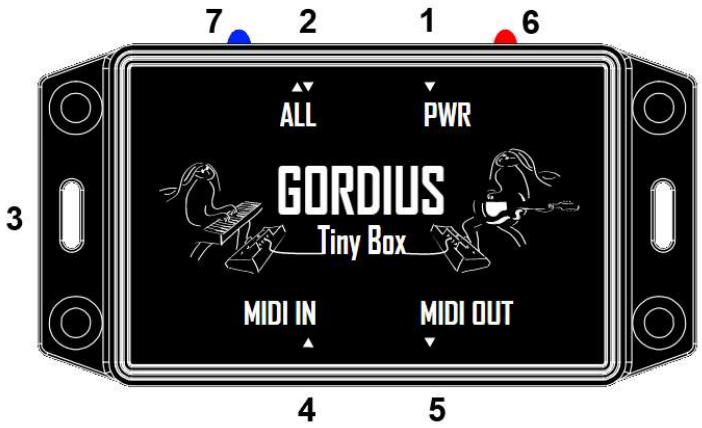


THE TINYBOX4KEMPER

User Manual v.1

Contents

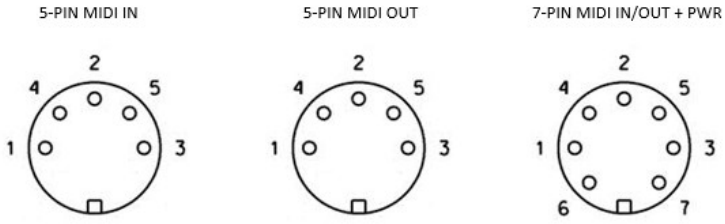
The TinyBox4Kemper hardware



1. Power input : connect any power adapter delivering 9V AC or DC, 500mA or more
2. 7-pins MIDI connector for two-way connection with FCB1010(*)
3. USB connector for connection with a Windows or Mac computer
4. MIDI IN connector, to be connected to the MIDI OUT connector of the Kemper Profiler
5. MIDI OUT connector, to be connected to the MIDI IN connector of the Kemper Profiler
6. Red power LED
7. Blue status LED
   - Slow blinking (each 2 seconds) = the TinyBox4Kemper is started and ready for connection
   - Fast blinking (each second) = the TinyBox4Kemper has detected a connected Profiler
   - LED off, except when blinking = no USB connection
   - LED on, except when blinking = USB connection with a computer detected

(*) The 7-pins cable transfers MIDI to and from FCB1010, along with FCB1010 phantom power. For plug-and-play compatibility the FCB1010 needs to be equipped with a “Single Cable Kit” (available at http://shop.tinybox.rocks) which makes the FCB1010 phantom powered and gives it the same 7-pins MIDI connector for connection with the TinyBox4Kemper.

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



5-PIN MIDI IN
1. N.C.
2. N.C.
3. N.C.
4. MINI IN +
5. MIDI IN –

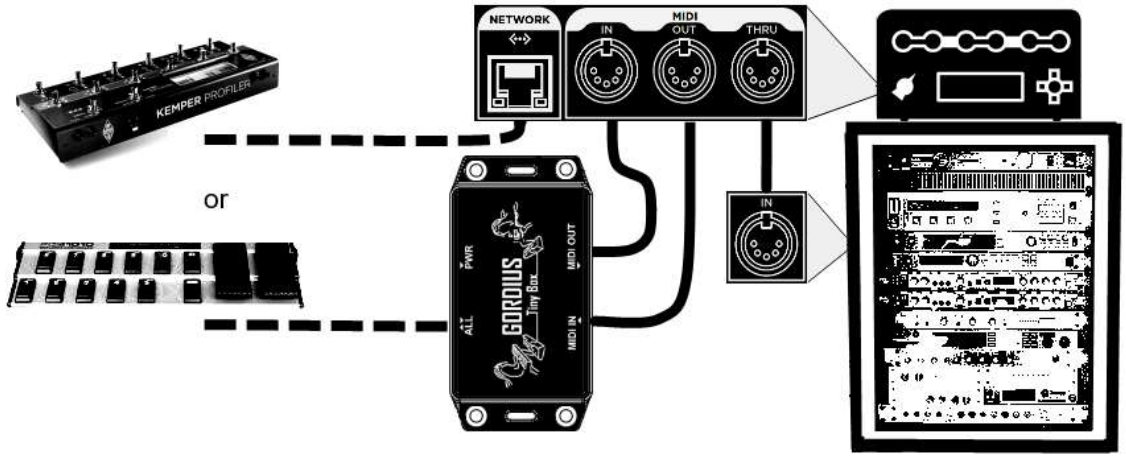
5-PIN MIDI OUT
1. N.C.
2. SHIELD
3. N.C.
4. MIDI OUT +
5. MIDI OUT –

7-PIN MIDI IN/OUT + PWR
1. MIDI OUT +
2. N.C.
3. MIDI OUT –
4. MIDI IN +
5. MIDI IN –
6. Power Adapter TIP
7. Power Adapter RING

Connecting the TinyBox4Kemper to your rig

The TinyBox4Kemper allows you to synchronize a complex MIDI based rig with your Kemper Profiler. To do so, it listens to rig and effect changes in the Profiler. This is possible by connecting a MIDI cable from Profiler MIDI OUT to TinyBox MIDI IN.

Each change can trigger a TinyBox4Kemper preset, which sends any number of additional MIDI commands. These commands travel from TinyBox4Kemper MIDI OUT to Profiler MIDI IN and from there through its MIDI THRU connector to all other devices down the MIDI chain.

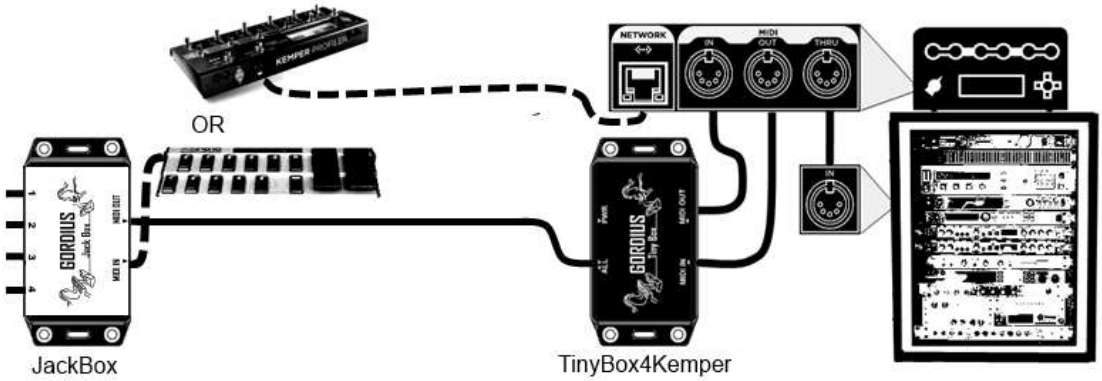


The TinyBox4Kemper reacts on Profiler rig changes to send extra MIDI messages to other devices. It doesn't matter how the rig change is done: it can be through the local buttons on the Profiler, by using the Kemper Remote, or by using a third party MIDI controller like the FCB1010.

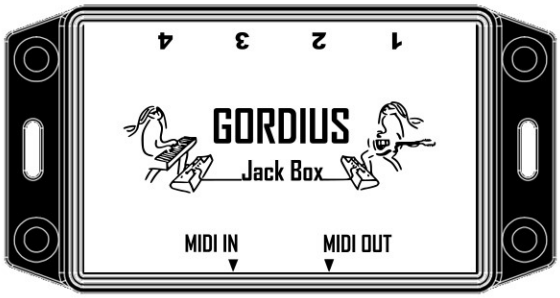
When using the FCB1010 with UnO4Kemper firmware, the TinyBox4Kemper allows you to connect it to your rig through 1 single cable, just like the Remote. For that a "Single Cable Kit" is available, which adds a phantom power option to the FCB1010, and replaces the two 5-pins MIDI connectors with a single 7-pins MIDI connector and a power switch. As the TinyBox also has a 7-pins connector, 2-way MIDI plus power can be sent through a single MIDI cable. No more power cable required at your feet, the FCB1010 is powered from the TinyBox4Kemper.

Adding the JackBox to your rig

The JackBox(*) allows you to turn up to 4 regular expression pedals into continuous MIDI controllers. The MIDI stream of the JackBox can be seamlessly merged with the UnO4Kemper MIDI stream coming from the FCB1010, by connecting the FCB1010 to the JackBox as depicted below, using a short 7-pin MIDI cable (this type of cable can be optionally ordered along with the JackBox).



Also when using the Remote as MIDI controller, you can add the JackBox to your rig by connecting it to the TinyBox4Kemper as shown above. This wiring gives the advantage that you don't need to run multiple long analog cables from your expression pedals to the Profiler. Instead you run one single MIDI cable which carries power and the digital MIDI signal. The TinyBox adds lots of possibilities to the expression pedals connected to it through the JackBox. You can easily program the expression pedals to have a different function for each rig, to have a linear sweep or volume (logarithmic) sweep, to use full range or part of the adjustment range, etc. All this is explained in great detail in later chapters.

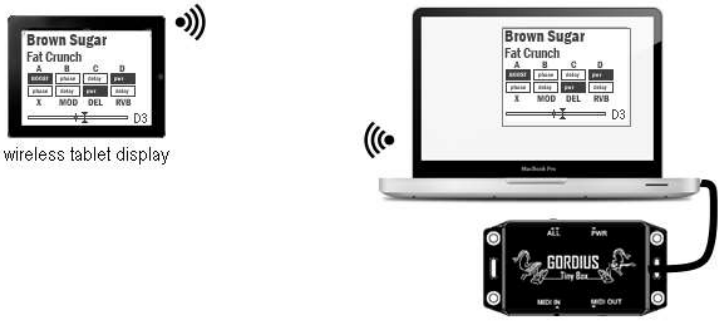


(*) a separate user manual is available for the JackBox

Using a computer with the TinyBox4Kemper

In order to program the TinyBox4Kemper, you need to connect it to a computer (Windows PC or Mac) using a USB cable. The computer runs the "TinyBoxServer" software. You will find out all details about this software in the next chapter.

The TinyBoxServer software not only allows you to program the TinyBox4Kemper, but it can also communicate with the TinyBox4Kemper during live use to request realtime status info of the Profiler: the currently selected performance, activated effects, tuner info when in tuner mode, etc. Since the TinyBoxServer software acts as a real lightweight web server, any web browser can connect to that server and display live status info. In the simplest scenario you can use the browser of the computer itself, however you can as well use an iPad or smartphone as "wireless status display". Of course for this your iPad or smartphone must have a WIFI connection with your computer. Most nowadays computers have WIFI built in and can easily be turned into a WIFI access point[1]. An alternative is to connect through an external WIFI router, which can be purchased new for around 20 USD.



using laptop as WIFI access point

using an external WIFI router

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

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

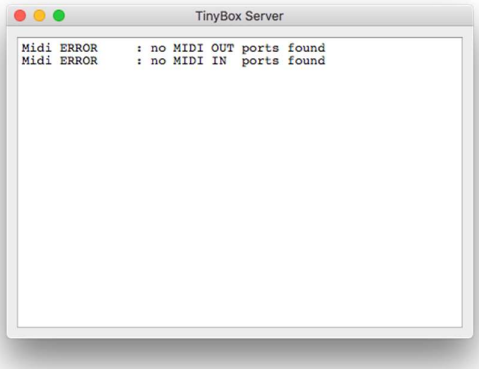
The TinyBoxServer software

This software can be freely downloaded from our webshop. Once you have purchased a TinyBox a download link will appear on your account page. Both a Mac installer and a Windows installer are available. The TinyBoxServer is actually a lightweight web server which communicates with the TinyBox through USB, and with a remote web browser through WIFI or wired network (next to communicating with a local browser on the same computer of course)

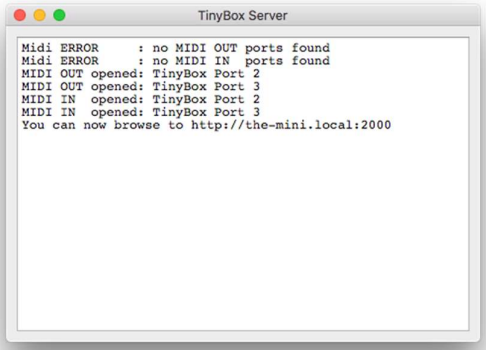
Launching TinyBoxServer

After launching the software, a status window appears. As with most “server” software, you will never interact with the TinyBoxServer software directly, everything is done in the browser instead. The status window just gives you a way to check if everything is correctly connected and up and running.

When no TinyBox4Kemper is connected, the TinyBoxServer status window will say something like this:



As soon as you connect the TinyBox4Kemper USB cable, this will change into :



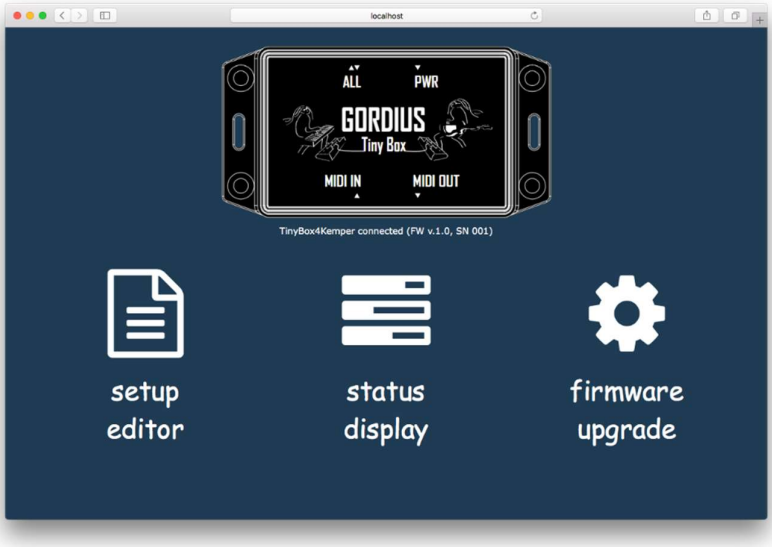
The TinyBoxServer opens 2 MIDI IN ports and 2 MIDI OUT ports on the TinyBox. One set of ports is used for transferring TinyBox4Kemper setups, the other set of ports is used for getting Profiler status info and for remotely controlling the Profiler from your iPad or laptop.

Connecting to TinyBoxServer

The webserver which is built into TinyBoxServer can be reached from a browser, using the address mentioned in the TinyBoxServer status window. This address will always be http://the-name-of-your-computer:2000 When you use the browser on the laptop which runs the server software you can as well use the address http://localhost:2000 . Alternatively, when you happen to know the IP address of your laptop you can also use that for connecting remotely: for instance http://192.68.0.120:2000

Don't forget to always include the ":2000" at the end. The webserver is listening on this specific port 2000 in order to keep the default port (80) available for regular internet browsing.

This is what you see when browsing to the given address :



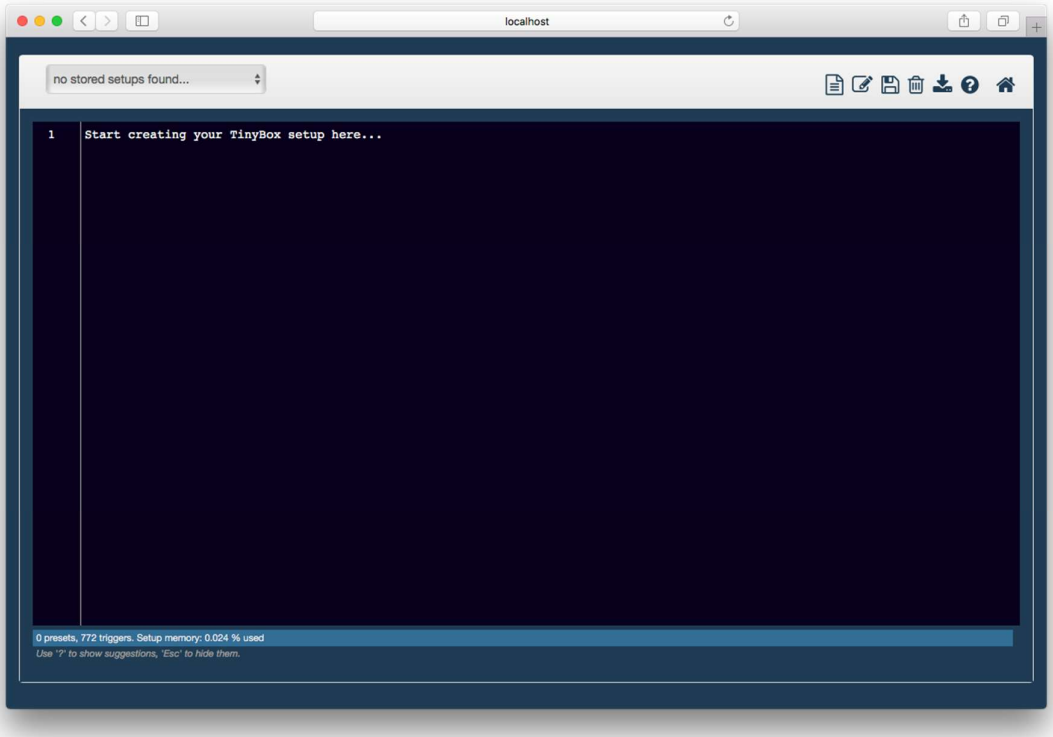
The start page shows you the 3 options which the software offers :

- Creating and editing TinyBox4Kemper setups
- Showing a realtime status display of the Profiler
- Upgrading the TinyBox4Kemper firmware

Tip:
If you wish you could create a shortcut on your laptop referring to the address of the editor, which is http://the-name-of-your-computer:2000/editor_kpa.html
On your iPad you could create a shortcut referring to the address of the status display, which is http://the-name-of-your-computer:2000/kpa.html

Using the TinyBox4Kemper editor

From the TinyBoxServer homepage shown above click the large “setup editor” icon.



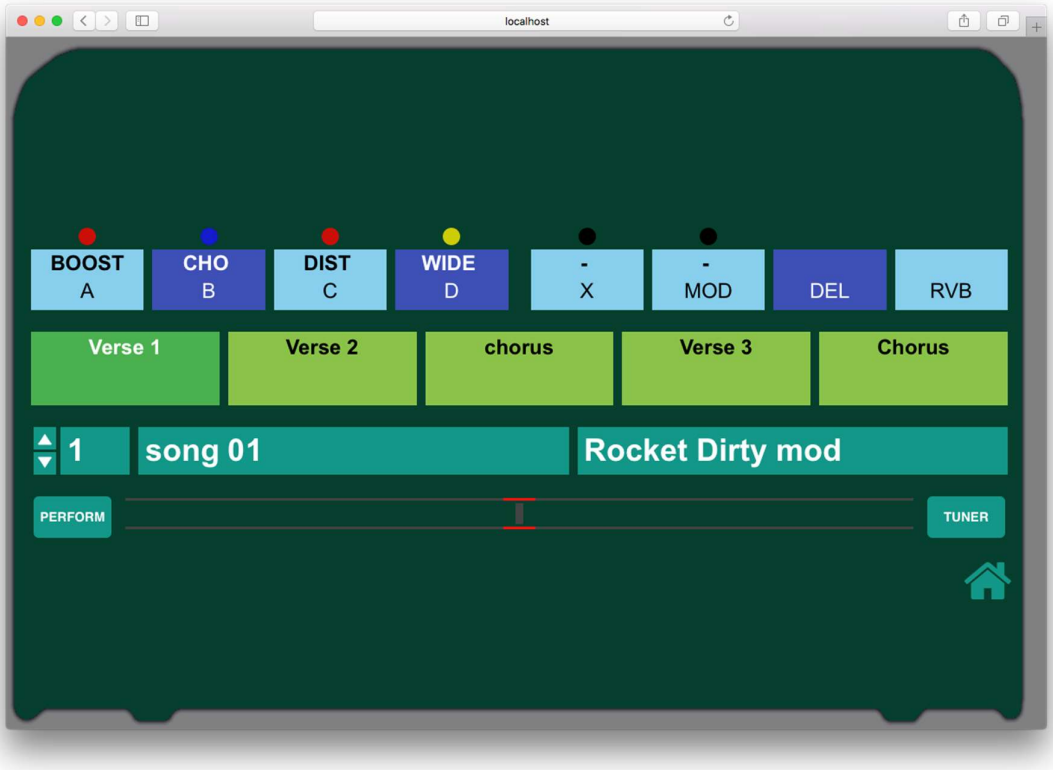
On this web page you can create, edit and manage your TinyBox4Kemper setups. The menu options are fairly self explanatory. They allow you to create, rename and delete setups, save changes, and download a setup to your TinyBox4Kemper. The Home button brings you back to the overview page shown in previous chapter, and the Help button shows the necessary info to assist you in writing a TinyBox4Kemper setup. The same info is also available in later chapters of this user manual.

Remember that TinyBoxServer is a web server, which means that you can edit your setup on any device which has a browser connected to the web server. However in practice the laptop running the server software will most probably also be the best tool for editing setups. A decent keyboard and mouse are still the best tools for text editing… No matter on which device you are editing your setup, all changes are always stored on the server side, that is on the computer running the TinyBoxServer software. One of the advantages of the TinyBox4Kemper approach is that its setups are purely text based. So you can very easily copy-and-paste your setup into an external text file for backup on a stick, for sharing with others on a forum, etc…

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

Displaying Profiler status info, and remotely controlling your Profiler

From the TinyBoxServer homepage shown earlier click the large “status display” icon.



This gives you a representation of your Profiler, showing detailed info about its current status. In the screenshot above, taken with the Profiler in performance mode, you can see :

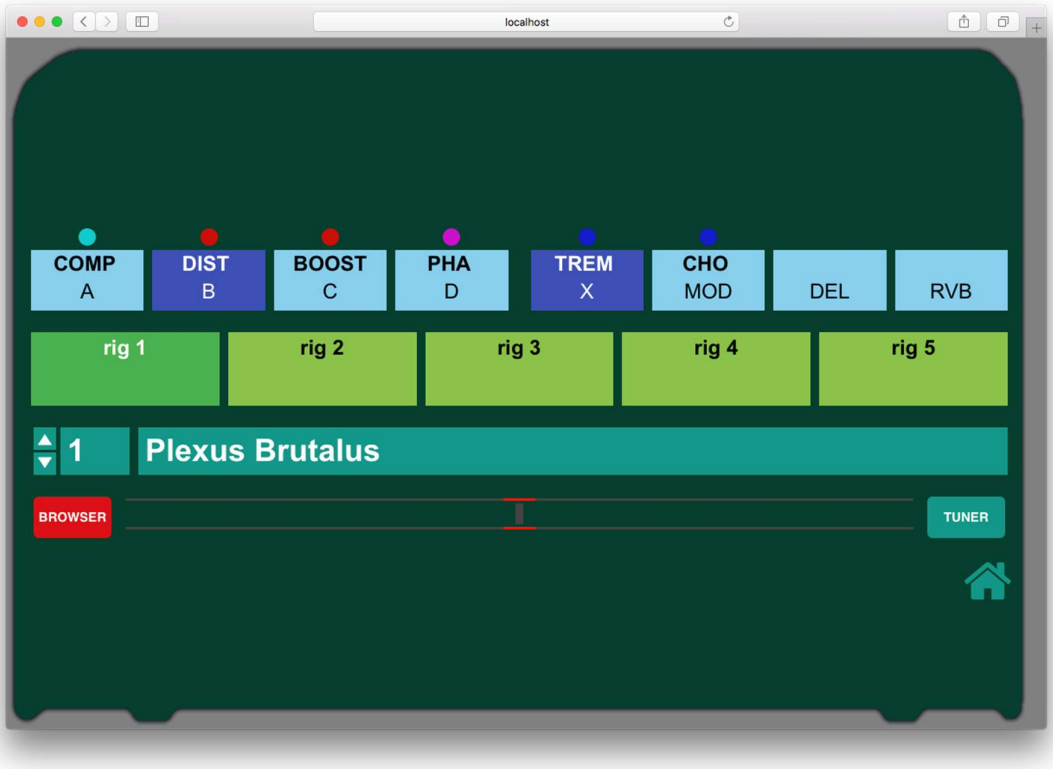
- which performance is currently selected (“song 01” in this test setup)
- which slot of the performance is selected (first slot, called “Verse 1”)
- which rig is active (“Rocket Dirty mod”)
- which effects are available (the X and MOD effects, showing ‘-‘, are not available for this rig)
- which effect is linked to each of the A/B/C/D/X/MOD effect slots (‘BOOST’,’CHO’,’DIST’,…)
- what type of effects they are (colored LEDs, similar to the Profiler front panel)
- which effects are active (they have a dark blue color)

Next to showing you the current status, this screen can also be used as an actual remote control for the Profiler. This way you can browse through performances, activate effects, or activate the tuner by using your iPad !

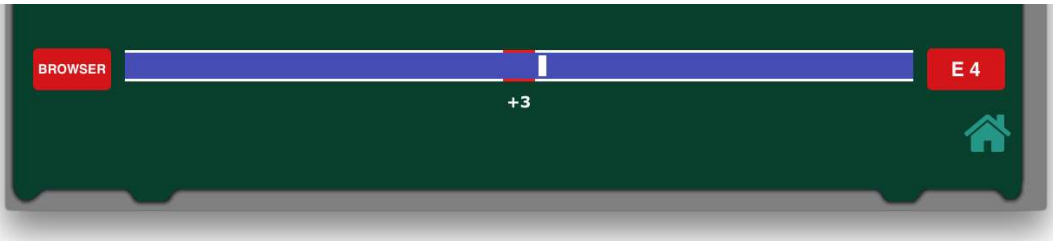
- click an effect box to activate or deactivate it
- click a slot box to select a slot within a performance
- click the up or down arrow to go to next or previous performance
- click the Performance toggle button to toggle between performance mode and browse mode
- click the Tuner toggle button to activate or deactivate the tuner

When using the Profiler in browse mode the status screen looks almost identical. You don't have a performance name and slot names, just the name of the selected rig. The 125 available rigs can be selected in banks of 5.

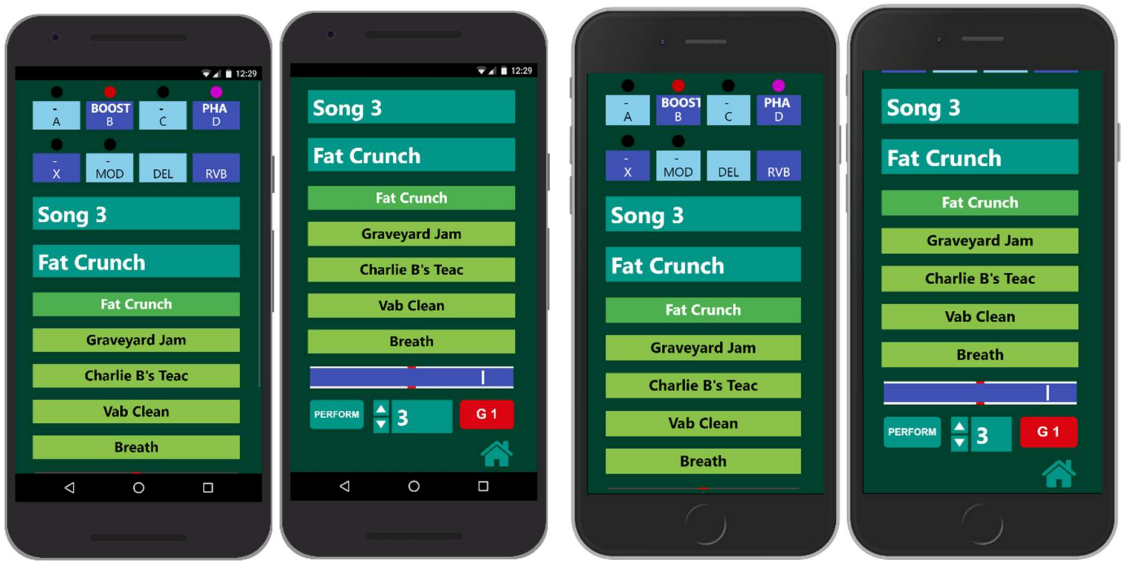
Tip : one of the Profiler system menus allows you to assign a ProgramChange number to a rig, which makes it available for selection in browse mode.



When activating the tuner (through the Profiler chickenhead knob, with an external MIDI controller, or by clicking the Tuner button in the TinyBoxServer status screen) realtime tuner info is displayed in the status screen :



Above screenshots were taken on a Mac computer, but the screen looks identical on iPad or Windows PC or tablet. While setup editing will always require a decent screen and keyboard for practical reasons, the Profiler status screen could also be displayed on the smaller screen of an iPhone or Android or Windows smartphone. In that case the layout is different in order to optimize the readability of all info on the small screen :



The status screen has been tested on following browsers so far :

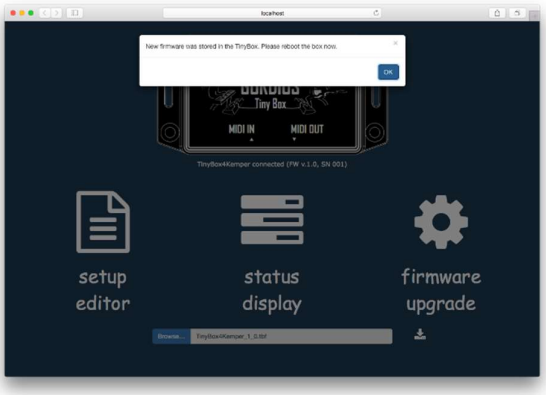
- Edge and Chrome on Windows
- Safari on Mac, iPad and iPhone
- Chrome on Android

Upgrading the TinyBox4Kemper firmware

From the TinyBoxServer homepage shown earlier click the large “firmware upgrade” icon. A “Browse” button appears on the screen, use it to select the TinyBox4Kemper firmware file (with extension .tbf).

Once the firmware file is selected, a Download button appears. Click it to download the firmware to the TinyBox4Kemper device:

After downloading the new firmware, you will be asked to reboot the TinyBox4Kemper. That concludes the firmware upgrade.



Programming the TinyBox4Kemper using the G# language

To specify the content of each TinyBox4Kemper preset, you use the "G#" programming language, which contains a limited number of easy commands like `SendMidi`, `ActivateSweep`, etc. While creating your setup, the G# 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 basic TinyBox4Kemper preset can contain just one or a few MIDI commands. However, as any other programming language G# also allows you to create complex presets by making use of variables. The content of a variable can be set or modified in any preset. Then you can use conditional "if... then... else..." statements to act upon the variable contents. This way you can create a very dynamic and complex setup.

The TinyBox4Kemper setup structure

A TinyBox4Kemper setup can contain :

- up to 625 presets when the Profiler runs in performance mode, one linked to each of the 5 slots in each of the 125 performances
- up to 128 presets when the Profiler runs in browse mode, one linked to each of the 128 rigs which has a MIDI ProgChange message assigned to it through the Profiler System Menu
- up to 16 presets in both performance and browse mode, linked to the ON or OFF event of each effect: A, B, C, D, X, MOD, DLY or RVB

A TinyBox4Kemper setup can use up to 128 *integer* variables, up to 256 *boolean* variables, and up to 256 *constant* variables.

- an integer variable can contain any number in the range 0-127. It can be used in any MIDI command instead of specifying a fixed value.
- an integer variable can be incremented, decremented, added or subtracted, and compared to other 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 constant variable can contain any of the constant values which you first define in your setup. Using constant values instead of integers can make your setup more readable.

Each TinyBox4Kemper preset can contain any combination of different types of commands:

- MIDI commands, like *ProgChange, CtrlChange, NoteOn, NoteOff, MIDIStart, MIDIContinue, MIDIStop, MIDIClock, SysEx*
- expression pedal commands, which let you specify the used CC number, value range and sweep curve of 4 connected expression pedals
- a *Delay* command, which halts MIDI transmission for a programmable number of milliseconds or seconds
- variable commands which set or modify the content of the different types of variables
- conditional commands which act upon the current content of those variables

Example 1 : the general setup syntax

```
// Below you see the general structure of a TinyBox4Kemper setup.
// You can add as much text comment to a setup as you like.
// A comment line starts with 2 slashes, as this line.

// 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:

DEFINE CHANNEL KPA      = 1
DEFINE CHANNEL Setlist  = 2
DEFINE CHANNEL TimeLine = 3
DEFINE CHANNEL Octaver  = 10

// Next step is defining all variables which will be used in the setup
// (if you need any. You can of course create a setup without using any variables)
// There are 3 types of variables : integer, boolean, or constant.
// The initial value of each variable indicates the variable type.
// In order to use a 'constant' variable, you first need to define
// the possible constant values to choose from :

// A constant value always starts with '#' :

DEFINE CONST #RYTHM
DEFINE CONST #CLEAN
DEFINE CONST #SOLO

// A variable name always starts with '$' :

DEFINE VAR $presetnumber = 1   // this is an integer variable
DEFINE VAR $use_delay = false  // this is a boolean variable
DEFINE VAR $mode = #CLEAN      // this is a 'constant' variable

// Below the channel and variable definitions come all preset definitions :

// The preset content can be a simple single command... :

DEFINE PRESET AddLowerOctave    = SendMidi Octaver CtrlChange 100 127
DEFINE PRESET RemoveLowerOctave = SendMidi Octaver CtrlChange 100 0
DEFINE PRESET NoOneKnows        = SendMidi Setlist ProgChange 1
DEFINE PRESET GoWithTheFlow     = SendMidi Setlist ProgChange 2

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

DEFINE PRESET SongForTheDead =
{
   SendMidi Setlist ProgChange 3
   SendRealtime MIDIStart
   SendRealtime MIDIClock 107 BPM
}

// You might want to create a preset which is activated right after powering the unit.
// This preset can contain some general initialization commands.
// Here the preset is called 'Initialize', but you can choose any name.

DEFINE PRESET Initialize =
{
   ScaleSweep CC07 KPA 0-127 SlowRising
   ScaleSweep CC07 Octaver 0-127 Linear
}

// The last step is to link each of the defined presets to one rig, slot, or effect :
// The initialization preset is linked to 'STARTUP' :
LINK Initialize TO STARTUP

// a preset can be linked to the ON or OFF state of any Profiler effect :
LINK AddLowerOctave    TO EFFECT STOMP_A ON
LINK RemoveLowerOctave TO EFFECT STOMP_A OFF

// or to any slot in performance mode :
LINK NoOneKnows    TO PERFORMANCE 1 SLOT 1
LINK GoWithTheFlow TO PERFORMANCE 2 SLOT 1

// or to any of the 128 'assigned' rigs in Browse mode :
LINK SongForTheDead TO RIG 1
```

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

DEFINE CHANNEL MyGear = 10
DEFINE CHANNEL MySynth = 3

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

DEFINE PRESET SimplePreset = SendMidi MyGear ProgChange 1

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

   SendRealtime MIDIClock 120 BPM
   SendRealtime MIDIStart
   SendRealtime MIDIContinue
   SendRealtime MIDIStop
   SendSysEx F0 00 20 33 02 7F 01 00 32 59 00 40 F7

   // Integer variables can be used for all of 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

```
DEFINE CHANNEL MyGear = 10
DEFINE VAR $whammy = false

// You can define the range and sweep type for each of the 4 standard 'continuous control'
// messages CC01, CC04, CC07 and CC11.
// Sweep type is linear by default, but can also be set to SlowRising or FastRising

DEFINE PRESET Startup = ScaleSweep CC07 MyGear 32-127 SlowRising

// An example of how to change the expression pedal behavior on a preset per preset base.
// Imagine the $whammy variable was set to true when selecting a certain preset,
// and imagine "my gear" has a good whammy effect controlled by ControlChange number 24.
// Following code turns the volume pedal into a whammy pedal when activating Stomp D :

DEFINE PRESET StompD_On =
{
   if($whammy)
   {
      ScaleSweep CC07 MyGear 0-127 Linear
      ActivateSweep CC07 MyGear CtrlChange 24
   }
}

// switching Stomp D off will reset the pedal to its original behavior :

DEFINE PRESET StompD_Off =
{
   ScaleSweep CC07 MyGear 32-127 SlowRising
   ActivateSweep CC07 MyGear CtrlChange 7
}

LINK Startup TO STARTUP
LINK StompD_On TO EFFECT STOMP_D ON
LINK StompD_Off TO EFFECT STOMP_D OFF
```

Example 4 : using variables

```
// The use of 'variables' is something very common in programming languages.
// We also added it to our 'G#' language. It adds huge possibilities to the TinyBox.
// There are 3 types of variables: integer, boolean, and 'constant'
// You can recognize a variable by its leading '$' :

DEFINE VAR $CurrentBank = 1           // this is an integer variable
DEFINE VAR $Delay = false             // this is a boolean variable (can be true or false)

// You cannot use constant variables unless you first have defined some constant values.
// After that, a constant variable can contain any of those constant values.
// You can recognize a constant value by its leading '#' :

DEFINE CONST #GoWithTheFlow
DEFINE CONST #NoOneKnows
DEFINE CONST #FirstItGiveth
DEFINE VAR $CurrentSong = #GoWithTheFlow   // this is a 'constant' variable


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

DEFINE PRESET Perf03_Slot01 =
{
   $CurrentSong = #NoOneKnows
   $Delay = true
   // ...
}

// you can change variable values in different ways, for instance increment or decrement :

DEFINE PRESET BankUp   = $CurrentBank++
DEFINE PRESET BankDown = $CurrentBank--

// The true strength of variables will become clear in the next example!
```

Example 5 : using conditional commands

```
// 'Conditional commands' are yet another concept which we borrowed from traditional
// programming languages. Everybody probably knows those typical 'if...then...else...'
// statements which allow an application to make decisions. The examples below show
// how your setup can behave differently, depending on the value of any 'variable'.

DEFINE PRESET Sample =
{
   if($Solo)    // here $Solo is a boolean variable, it can be true or false
   {
      // send a set of MIDI messages...
   }
   else if($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($CurrentPreset)
   {
      case #ACOUSTIC:
         SendMidi MyGear CtrlChange 112 127
         break
      case #SOLO:
         SendMidi MyGear CtrlChange 12 0
         SendMidi MyGear CtrlChange 113 127
         break
      // and so on...
      default:
         SendMidi MyGear CtrlChange 12 127
         break
   }
}
```

G# language reference

Comments

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

can be used at the start of any line or after any command, to add your comments to the setup

DEFINE

```
DEFINE CONST #constname
```

defines a constant value, which can be used as value for any constant variable.

```
DEFINE VAR $intvarname = [0...127]
```

defines an integer variable. It can have any value between 0 and 127. A setup can use up to 128 different integer variables.

```
DEFINE VAR $boolvarname = [true/false]
```

defines a boolean variable, which can be set to true or false. A setup can use up to 256 different boolean variables.

```
DEFINE VAR $constvarname = #constname
```

defines a 'constant' variable, which can contain any of the predefined constant values. A setup can use up to 256 different constant variables.

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

The channelnames will be used in all MIDI commands, instead of specifying the number 1 to 16. This way you can easily move a device to a different channel at any time.

```
DEFINE PRESET presetname = [single command]
```

A preset is the main component in your TinyBox setup. Each rig change can trigger a preset. A simple preset can consist of one single command to be executed.

```
DEFINE PRESET presetname = { [list of commands] }
```

A more complex preset can consist of many different commands. In that case the preset content is written on multiple lines, and it is surrounded by curly brackets.

LINK

LINK presetname TO PERFORMANCE n SLOT m

links a preset to given slot (1-5) of the given performance (1-125). The preset will be triggered as soon as that slot is selected using footcontroller or Profiler buttons.

LINK presetname TO RIG i

links a preset to given rig (1-128) when the Profiler is used in Browse mode. The rig index is specified by assigning a "ProgChange" value between 1 and 128 to it in the Profiler System Menu.

LINK presetname TO STOMP_A/STOMP_B/STOMP_C/STOMP_D/X/MOD/Delay/Reverb ON/OFF

links a preset to the ON or OFF state of any of the Profiler effects. The preset will be triggered each time the effect is activated or deactivated.

LINK presetname TO STARTUP

The preset linked to 'STARTUP' will be activated when the TinyBox is started. It typically contains general initialization commands like defining the sweep range of expression pedals, etc.

MIDI commands

SendMidi channelname ProgChange number

SendMidi channelname CtrlChange number value

SendMidi channelname NoteOn number velocity

SendMidi channelname NoteOff number velocity

SendSysEx F0 … F7

These are the most common instructions, which simply send the specified MIDI command. According to the MIDI spec, 'number', 'value' and 'velocity' are all values between 0 and 127. However, our G# language also allows you to use integer variables instead of numbers, even in a SysEx content! For instance:

```
SendMidi MyGear ProgChange $currentSound
SendSysEx F0 7F 13 $speed F7
```

SendRealtime MIDIStart

SendRealtime MIDIContinue

SendRealtime MIDIStop

SendRealtime MIDIClock [10…250] BPM

This last instruction is a special one, because it does not send one single MIDIClock message. Instead, it starts a stream of clock messages with a tempo between 10 and 250 beats per minute.

DELAY command

`Wait [1…127]`

This instruction can be inserted at any place within a preset content. It simply halts MIDI transmission for the specified time. The time is expressed in 100ms units, so it can be set from 0.1 sec to 12.7 sec.

Expression pedal commands

`ScaleSweep [CC01/CC04/CC07/CC11] channelname from-till [sweepstyle]`

With this command you specify the outgoing sweep range of each of the 4 continuous control messages sent by an expression pedal. 'from' and 'till' are values between 0 and 127. You can even inverse the sweep by choosing a 'from' value which is higher than the 'till' value!

'sweepstyle' is optional and can be `Linear`, `SlowRising` (log or audio taper) or `FastRising` (inverse log or inverse audio taper).

`ActivateSweep [CC01/CC04/CC07/CC11] channelname CtrlChange number`

With this command you activate an expression pedal on the given MIDI channel. You can keep the original CC number (01,04,07,11) or you can specify a different number to adjust any other parameter of your gear instead of the parameters linked to CC 01/04/07/11.

After blocking an expression pedal with `BlockSweep` (see below) you can reactivate it with this `ActivateSweep` command.

`ActivateSweep [CC01/CC04/CC07/CC11] channelname PitchBend`

This command turns your expression pedal into a PitchBend pedal. PitchBend values are calculated with a full 14-bit resolution. Make sure to use a linear sweep when sending out PitchBend messages.

`ActivateSweep [CC01/CC04/CC07/CC11] channelname ChannelPressure`

Use this command to send out ChannelPressure messages with an expression pedal, instead of ControlChange messages.

`BlockSweep [CC01/CC04/CC07/CC11] channelname`

This command disables expression pedal activity which was previously enabled through `ActivateSweep`.

Variable commands

`$intvarname = value`

An integer variable can be set to any value between 0 and 127.

`$intvarname = $intvarname2`

Integer variable contents can be copied to one another.

`$intvarname = $intvarname2 [+-] value`

You can do basic arithmetic with integer variables (adding and subtracting).

`$intvarname [++ --]`

An integer variable can be incremented (++) or decremented (--).

`$intvarname [+= -=] value`

`$intvarname += value` is a shortcut notation for `$intvarname = $intvarname + value`

`$boolvarname = [true false]`

A boolean variable can be set to true or false.

`$boolvarname = $boolvarname2`

Boolean variable contents can be copied to one another.

`$boolvarname = !$boolvarname2`

Boolean variables can be inverted (`'!'` means 'not')

`$constvarname = #constname`

A constant variable can be set to any of the constant values, which are first defined through a series of `DEFINE` instructions.

Conditional commands

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

You can use the classical conditional statements which some of you may already know from programming languages like javascript. The statements to be executed when a condition is true are normally written on separate lines, surrounded by curly brackets :

```
if(&intvarname > 64) {
   // any number of commands here…
}
else if(&intvarname > 32) {
   // any number of commands here…
}
else {
   // any number of commands here…
}
```

`switch(&intvarname) { case value: … break }`

The switch statement is a powerful way to say that you want to do different things depending on the value of an integer variable. The multi-line statement is formatted as follows :

```
switch(&intvarname) {
   case 10:
      // any number of commands here…
      break
   case 15:
      // any number of commands here…
      break
   …
   default:
      // the commands of this section will be executed
      // if the variable value is none of the values
      // mentioned in the different 'case' statements.
      break
}
```

`switch(&constvarname) { case value: … break }`

The same switch statement can also be used for constant variables. After having defined constant values `#abc, #def, …` for instance, you can say :

```
switch(&constvarname) {
   case #abc:
      // …
      break
   case #def:
      // …
      break
   …
   default:
      // …
      break
}
```

Conditions

`$intvarname [ > >= == != <= < ] [0…127]`

An integer variable can be compared to any value between 0 and 127. The supported comparison operators are shown between the square brackets above.

`$intvarname [ > >= == != <= < ] $intvarname2`

An integer variable can also be compared to another integer variable

`$constvarname [ == != ] #constname`

A constant variable can be compared to any predefined constant value

`[!]$boolvarname`

A boolean variable can be directly used as condition. Prepend a `'!'` when you want to check if it is false instead of true.

`$boolvarname [ == != ] $boolvarname2`

A boolean variable can also be compared to another boolean variable.

`( (condition1) && (condition2) && … )`

You can check if multiple conditions are all true (`'&&'` stands for 'and') ...

`( (condition1) || (condition2) || … )`

...or if at least one out of multiple conditions is true (`'||'` stands for 'or').

Have fun !