Important:

The software version on the neonious one's given out at JSConf is no longer up to date, so this manual is depreciated. Newer versions of the software have far more features, are faster and far more stable.

For serious programming, we can provide you with a software update, please contact us at info@neonious.com if interested.

Quick & Dirty Technology Preview manual

June 2nd 2018

This is not full-blown manual of high quality, as the documented neonious one Technology Preview is not a final product.

The purpose if this document is to document the neonious one TP particularities. For productive use of the neonious one TP the user still needs a good understanding of JavaScript and electronics.

1. Important notes

1.1. You are a beta-tester!

Usually, it's done when it's done.

Your neonious one Technology Preview however was built with a deadline in mind: JSConf.EU @ 2nd of June 2018. Our team was determined to get the core features ready in time. It managed to do so, but did not get much sleep in the past few weeks.

At the end, we believe we have a great proof of concept for our final neonious one board:

> A microcontroller board, programmable and debuggable out of the box in an on-board web-based IDE, combining JavaScript ES 6, TypeScript, Node.JS API and the whole npm repository with Internet directly on the board via WiFi or Ethernet and with all the other interfaces a microcontroller board usually provides.

So, take a look at the board. Stir up a great electronics project to do with it, and have fun!

If you stumble on a bug – there are several –, report it to help@neonious.com and we will keep you closely notified on our process in fixing it via over-the-air updates. It will be gone soon, we promise. And thank you for your patience.

1.2. What's included with the Technology Preview?

Included in the TP	**Part of final neonious one (get it on Kickstarter now!)**
JavaScript ES 6	TypeScript
Our great browser-based IDE with editor + fully-featured debugger	Faster JIT compiler
Ethernet & Wifi	External IDE support
HTTP server	Code hibernation, run on battery for months!
GPIO / PWM / ADC	HTTP client
	HTTPS (for user program)
	UART, I2C, SPI, I2S
	Bluetooth
	Package manager with full npm support

There are some differences between getting the Technology Preview for free at JSConf and purchasing the Technology Preview through our Kickstarter:

JSConf TP version	**Kickstarter TP version**
You will receive all bug fixes, but only select software features via over-the-air updates.	We will make sure you get all bug fixes, and all software features of the final neonious one, as soon as they are developed and tested.
If you wish to get all available over-the-air updates, you may pledge for one of the items we offer in our Kickstarter (http://www.neonious.com/Kickstarter), such as a kit or the final neonious one.	In addition to the Technology Preview board, you will also receive a neonious one board when it's done.
If you then send us the 12-digit code from the back of the packaging of your	

Technology Preview board to info@neonious.com, we will whitelist your board for all updates, just as if it was a Kickstarter version.

1.3. Notes on HTTPS

Making the device support HTTPS was one of the challenging tasks.

1.3.1. Not all HTTPS clients are supported yet

HTTPS usually requires 16K of memory for both receive and send buffers for each connection, as TLS frames may be up to this size. On the ESP32 HTTPS is only processed in acceptable speed if internal memory is used, but in internal memory there is not enough free RAM for multiple connection buffers.

The solution in the final neonious one:

We identified big pieces of internal memory which should rather be in the larger external memory, freeing up space. This way we have enough internal memory to support all HTTPS clients.

The solution in the Technology Preview:

We only support HTTPS clients which support the max_fragment_length extension, which allows the neonious one to tell the client that the frames must be of smaller size. This means that the clients must use

- mbedtls or
- OpenSSL >= 1.1.1

For example, if you use Debian stable, you will have problems.

1.3.2. Certificate warnings are by design

neonious one is shipped with an self-signed certificate which prompts a warning in modern browsers.

The reason for this is that with an self-signed certificate you cannot notice whether somebody does an man in the middle attack, which is difficult but not unheard of local networks. The self-signed certificate still makes sure that no third party not relaying the data can monitor your communication, which is the greater threat.

Using an self-signed certificate is by design, as we cannot ship with a CA-signed certificate, as this would require the neonious one to be accessible through a domain name.

You may however upload your own certificate and key in the settings tab of the IDE. If you then access your neonious one via domain name, the warnings are gone.

2. Access your neonious one

2.1. General steps

To access your neonious one, you need to open your web browser and point it to the board's address.

The exact steps depend on the current settings.

via Wifi / Pre-set settings @ JSConf

Enter the conference network

Find the IP address to connect to by using the Find My Neonious feature (see 2.3.)

If you no longer are at JSConf, you need to reset the network settings (see 3.1.).

Via Wifi / After factory reset

Connect to the access point your neonious one is providing (with the name "neonious one <Code>", with <Code> being the code on the back of the packaging of your neonious one.

The Wifi password is written down on the back of the packaging of your neonious one.

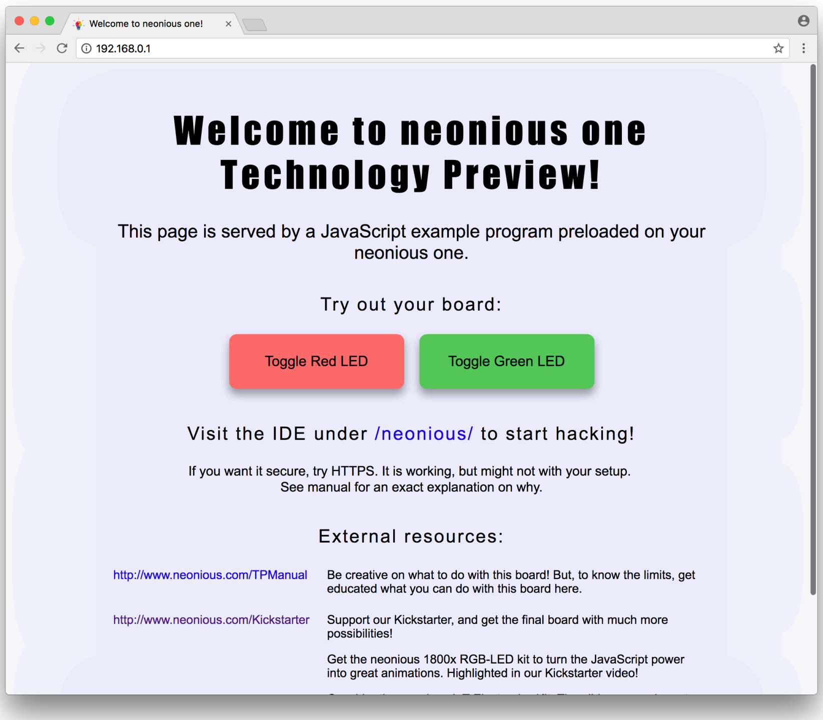
The boards IP address is 192.168.0.1

via Ethernet

The pre-set and factory reset settings are the same for the Ethernet interface: neonious one waits for an IP address via DHCP.

Use the Find My Neonious feature to determine the IP address you received.

If the user program runs an HTTP server, you will see the user program's page. Otherwise you will get redirected to the IDE at /neonious/.



Welcome to neonious one Technology Preview!

This page is served by a JavaScript example program preloaded on your neonious one.

Try out your board:

Visit the IDE under /neonious/ to start hacking!

If you want it secure, try HTTPS. It is working, but might not with your setup.
See manual for an exact explanation on why.

External resources:

http://www.neonious.com/TPManual	Be creative on what to do with this board! But, to know the limits, get educated what you can do with this board here.
http://www.neonious.com/Kickstarter	Support our Kickstarter, and get the final board with much more possibilities!
	Get the neonious 1800x RGB-LED kit to turn the JavaScript power into great animations. Highlighted in our Kickstarter video!

The pre-installed example program

If you do NOT see the user program's page, but the browser keeps spinning, it makes sense to go to the IDE: Your user program might have thrown an exception, in which case your program is paused. In this case, you can see where it crashed, fix the problem and restart the program in the IDE.

The password settings for the IDE are the following:

Pre-set settings @ JSConf	**After factory reset**
The password is written down on the back of the packaging of your neonious one.	The default password is empty.

2.2. Find My Neonious to determine the device's IP address

When connecting to an DHCP-enabled network, your neonious one might get a random IP address and you might have difficulties to determine it. If your board has Internet access in this new network and you did not disable the tracking feature in the IDE settings, you can determine the board's IP address via the Find My Neonious feature.

1. Go to https://www.neonious.com/FindMyNeonious
2. Enter the 12-digit code from the back of your board's packaging

If the website does not report the Internet address of your board, you might have entered wrong network credentials. Reset network settings (see 3.1.) and try again.

2.3. You lost the packaging of your neonious one?

The code and password on the sticker on the back of the packaging are usually required to get access to your neonious one, if you did not change to custom settings.

There is still one way to access your neonious one, however:

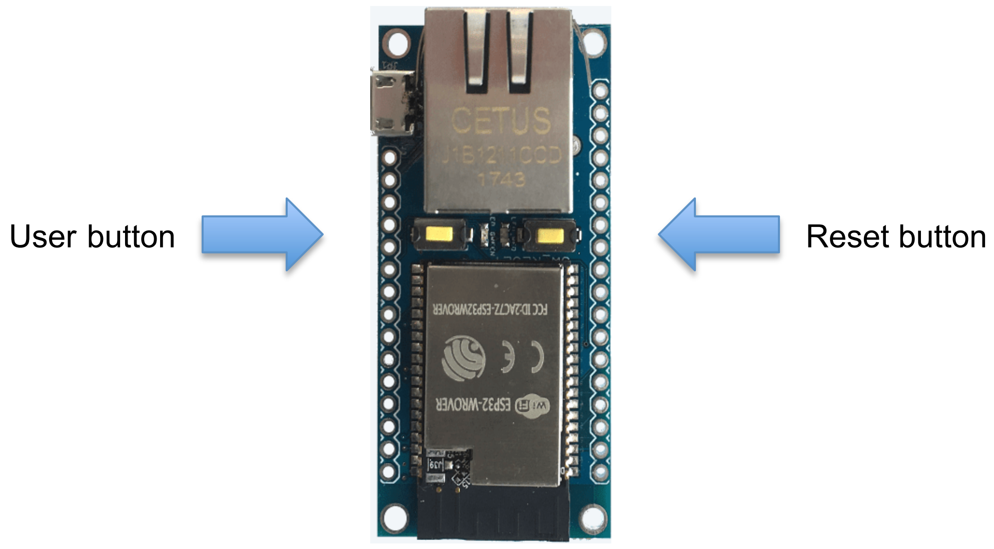
To access your neonious one, you can reset to factory, and then connect via Ethernet. This way you do not need any password to enter the IDE.

Then, in the IDE settings, change the password of the Wifi access point. You can know also use Wifi.

3. Updates / Reset options

3.1. Reset network settings

If you did a mistake while setting your network settings, you might no longer be able to connect to your neonious one. You may then reset the network settings to factory default, which allows you to access your neonious one via the factory access point (see 2.1.).



1. Press and hold the User button
2. Tap the Reset button
3. The red and green LEDs will flash alternately

4. Release the User button as soon as the green light starts flashing (after about 5 seconds).

Do NOT hold the User button for 10 seconds, because this will trigger a factory reset, and all your data will be lost.

3.2. Reset to factory

Do the same as in 3.1., but hold the User button till the right LED starts flashing (after about 10 seconds).

After this, all your data will be lost.

3.3. Software Updates

Tell us, if you wish to be notified about new updates: info@neonious.com !

Software Updates are installable right from the IDE. To see them, neonious one needs Internet access. So this does not work, if your neonious one is it’s own access point.

Note: If network settings change, your software update might not complete, as your neonious one needs Internet while updating.

4. Callable API

Please note that this documentation style is rudimentary. The final neonious one will track the Node.JS API, but still include an documentation of each single method and property.

We cannot provide this with the Technology Preview however, as the Technology Preview is a moving target.

If you need help on how to get something done with the current API, write us at help@neonious.com.

4.1. JavaScript features

You may use all JavaScript ES 6 features, such as ES 6 modules, async/await, and so on. Clases such as TextEncoder, Proxy, Symbol or Promise can be used.

For the newest features, take a look at http://www.es6-features.org/

One notable missing feature is Set / Map. When including this, we will have to look into integrating WeakMap correctly with the garbage collector, which is why this did not make it into the Technology Preview.

4.2. Global properties

The following globals of the Node.JS API are currently supported:

- Class: Buffer
- __dirname
- __filename
- clearImmediate(immediateObject)
- clearInterval(intervalObject)
- clearTimeout(timeoutObject)
- console
- exports
- global
- module
- ~~process~~ currently missing
- require()
- setImmediate(callback[, ...args])
- setInterval(callback, delay[, ...args])
- setTimeout(callback, delay[, ...args])

Notes about console

The console calls currently return when writing is finished. Because writing Flash is not really fast, console calls can slow down your program a lot.

We will allow console to finish writes in the background in the final neonious one, maxing single console calls fast. If you need an fast asyncronous alternative, use fs.writeFile for now.

4.3. Buffer

Node-type Buffer is supported. Please not that instead of using Buffer.alloc and Buffer.from, you currently have to use new Buffer.

4.4. events / path / querystring

This modules are in sync with the Node.JS API.

4.5. url

This Node.JS API module is available, too. The newer WHATWG URL API is still missing however (new url.URL(…) returns an older urlObject for now).

4.6. http

The server side API of the Node.JS http module is supported already!

http.ServerResponse it is not yet derived from stream.Readable. However, it does behave like a stream already, with one exception: callbacks given to write and end are not called.

4.6. stream

The stream module currently only supports stream.Readable and stream.Writable from the current Node.JS API.

4.7. fs

The following methods of the Node.JS API are currently supported:

- fs.readFile
- fs.writeFile
- fs.open
- fs.close
- fs.read
- fs.write
- fs.fstat
- fs.readFileSync
- fs.writeFileSync
- fs.openSync
- fs.closeSync
- fs.readSync
- fs.writeSync
- fs.fstatSync

Notes

Paths must be absolute paths.

Note that seeking, even done indirectly by giving a position in fs.read or fs.write, currently results in an error. This is because files on neonious one are compressed on the fly, and compressed files cannot be seeked/skipped.

neonious one does support saving files uncompressed, and soon fs.open will support a flag to tell the system to do just that.

4.8. neonious

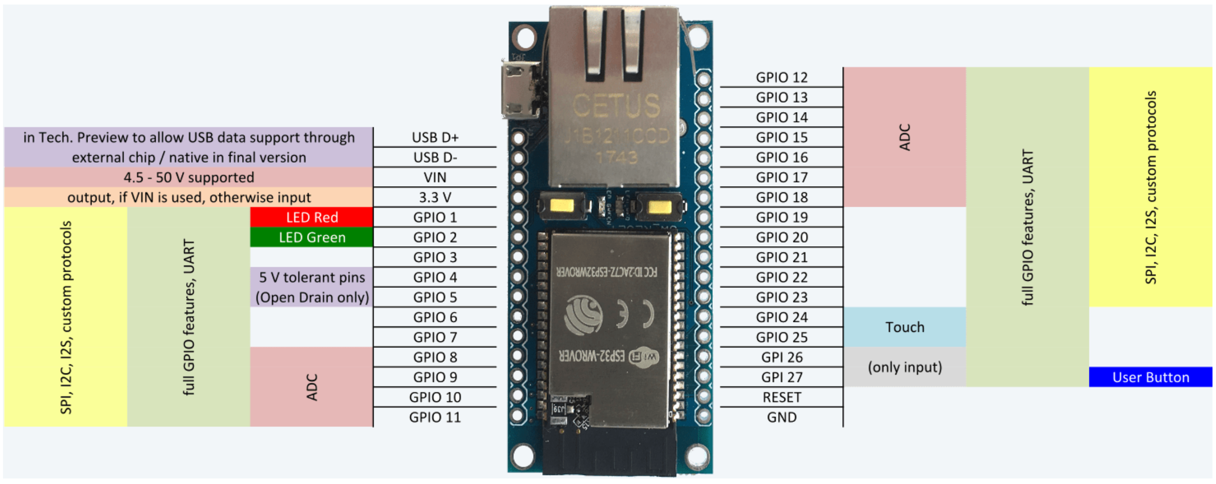
neonious.gc()

Calls the garbage collector. You may notice that this makes sense right now, because a full heap slows down the program immensely.

This will be fixed soon, by using a besser heap allocation algorithm.

4.9. gpio

Access I/O pin via gpio.pin[<NUM>], with <NUM> being a value between 1 and 27.



The following constants can also be used instead of <NUM>:

- gpio.LED_RED
- gpio.LED_GREEN
- gpio.BUTTON – the user button (left)

Methods of gpio.pin[<NUM>]

- setType(type)
  With type being one of
  gpio.INPUT
  gpio.INPUT_PULLUP
  gpio.INPUT_PULLDOWN
  gpio.OUTPUT
  gpio.OUTPUT_OPENDRAIN

  Notes
  26 and 27 only support INPUT
  4 and 5 only support INPUT or OUTPUT_OPENDRAIN, but are 5 V tolerant and support 20 mA! Great to interface 5 V devices with external pull-up to 5 V.
  6 does not work yet, but will soon

- getType() returns type

- setValue(val)
  0 – low level
  1 – high level
  value in between – PWM
- getValue([type,]callback)
  type: gpio.DIGITAL or gpio.ANALOG (12-bit ADC normalized to 0-1)
  callback(err, value)

Events of gpio.pin[<NUM>]

- fall
  emitted on input pins when driven low

- rise
  emitted on input pins when driven high

4.10. signal

Allows to send an signal with a length of up to 2^32 nanoseconds (a bit over 4 seconds) to up to 8 pins, with up to 6 defined points in time (events) when the pins can individually be turned on or off. The points in time are defined in nanoseconds, but final the resolution is 33.33… nanoseconds.

Used by gpio to realize PWM. See the table radar example for an example on how to use this to drive servo motors.

signal.clear()
Stops the currently sent signal.

signal.send(`flags, pins, eventNanoSecs, callback`)
with

> flags
> currently either
> signal.ONESHOT – signal is only sent once
> signal.RESTART – signal restarts after last defined event (in this case it makes sense that the last event has the latest defined point of time
>
> pins
> Array with one object per pin used:
> [{
>   index: <pin number>,
>   setEvents: <events when pin will be set high>,
>   clearEvents: <events when pin will be set low>
> }, …]
> setEvents and clearEvents are set like:
> signal.EVENT_1 | signal.EVENT_3 …
>
> eventNanoSecs
> Array with integers of the points in time in the unit nanoseconds of the used events
> callback
> with parameter err, interrupted
> interrupted is true if signal.send or signal.clear is called while the signal is still running, and false if the signal finished running.

5. Example Programs

5.1. Interface additional LEDs with neonious one

Which pins to use

As with most modern microcontrollers, neonious one cannot drive much current per pin (3 mA). A normal-sized LED uses around 20 mA, typically requiring you to use a transistor or MOSFET in between.

neonious one has 2 special pins however, which do support 20 mA: Pins 4 and 5. With these pins you can drive LEDs directly.

These pins are open-drain only pins, however:

Mode	**Voltage at level 0**	**Voltage at level 1**
OUTPUT	GND	3.3 V
OUTPUT_OPEN_DRAIN	GND	*not connected*

This has no implications, other that we will have to make the LED light up at level 0 and not at level 1, because only at level 0 an electronic circuit is closed.

Step 1) Connect: Pin -> LED -> resistor -> 3.3 V PIN

The resistor is needed to limit the voltage on the LED, and because of that also the current flowing through the LED.

Formula: R = (3,3 V - VLED) / ILED

Typical value for VLED is 2.2 V, for ILED is 20 mA. This would result into a resistor value of 55 Ohm. Take the next resistor with the next highest value you have.

Warning: If you connect the LED to any other pin, too much current might flow through that pin, breaking your neonious one.

Warning: If you connect the LED without the resistor, too much current might flow through the pin and the resistor, breaking your neonious one and/or your LED.

Step 2) Write code

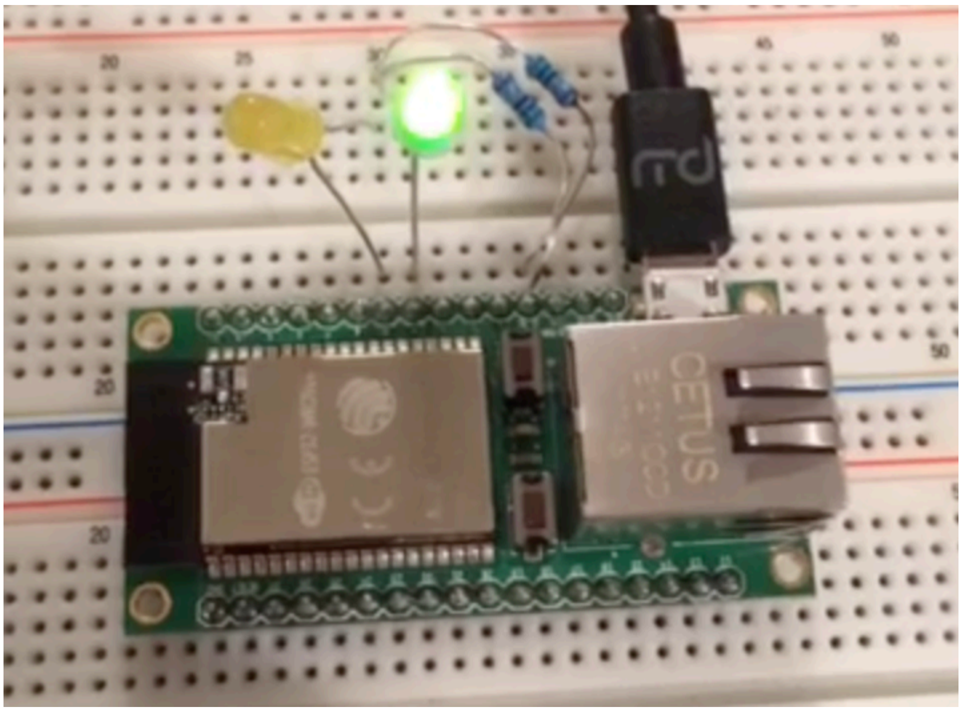
For example, if you have an LED on pin 4 and another one on pin 5, you could write into your /src/index.js:

```
let gpio = require('gpio');
gpio.pins[4].setType(gpio.OUTPUT_OPENDRAIN);
gpio.pins[5].setType(gpio.OUTPUT_OPENDRAIN);

let level4 = false;
setInterval(() => {
      level4 = !level4;
      gpio.pins[4].setValue(level4);
}, 500);

let level5 = false;
setInterval(() => {
      level5 = !level5;
      gpio.pins[5].setValue(level5);
}, 700);
```

Step 3) Press run! If the LEDs do not blink, try putting the LED(s) in the other way around.



5.2. Table Radar

Take a look at the table radar example program hosted at
https://github.com/neoniousTR/table_radar

For a video of the table radar in action, click here:
https://drive.google.com/file/d/1G6cykg5DXdzrsAG8MDHQRWeSNzXArNSn/view