

Espruino Hardware Reference

The Espruino Interpreter runs on a variety of different hardware. Please click on the thumbnails below to see technical data on each board.

Official Boards

The official boards are designed specifically for the Espruino JavaScript interpreter. They come with Espruino pre-installed so are ready out of the box. Got a problem? We provide support [on the Espruino forums](#).



Third Party boards ▲

Third party boards are designed, sold and supported by companies who help to support Espruino's continued development. They may or may not come with Espruino pre-installed, however we ensure that [up to date firmware for these boards](#) is available so you always have access to the newest features.



Unsupported boards ▼

Espruino Software Reference

Version 2v18

Contents

- [Globals](#)
- [AES](#)
- [Array](#)
- [ArrayBuffer](#)
- [ArrayBufferView](#)
- [Bangle](#)
- [BluetoothDevice](#)
- [BluetoothRemoteGATTCharacteristic](#)
- [BluetoothRemoteGATTServer](#)
- [BluetoothRemoteGATTService](#)
- [Boolean](#)
- [CC3000](#)
- [console](#)
- [crypto](#)
- [DataView](#)
- [Date](#)
- [dgram](#)
- [dgramSocket](#)
- [E](#)
- [Error](#)
- [ESP32](#)

- [ESP8266](#)
- [Ethernet](#)
- [File](#)
- [Flash](#)
- [Float32Array](#)
- [Float64Array](#)
- [fs](#)
- [Function](#)
- [Graphics](#)
- [heatshrink](#)
- [http](#)
- [httpCRq](#)
- [httpCRs](#)
- [httpSRq](#)
- [httpSRs](#)
- [httpSrv](#)
- [I2C](#)
- [Int16Array](#)
- [Int32Array](#)
- [Int8Array](#)
- [InternalError](#)
- [JSON](#)
- [Math](#)
- [Microbit](#)
- [Modules](#)
- [neopixel](#)
- [net](#)
- [NetworkJS](#)
- [NodeMCU](#)
- [NRF](#)
- [Nucleo](#)
- [Number](#)
- [Object](#)
- [OneWire](#)
- [Pin](#)
- [Pixel](#)
- [process](#)
- [Promise](#)
- [Puck](#)
- [ReferenceError](#)
- [RegExp](#)
- [Serial](#)
- [Server](#)
- [Socket](#)
- [SPI](#)
- [Storage](#)
- [StorageFile](#)
- [String](#)
- [SyntaxError](#)
- [TelnetServer](#)
- [tensorflow](#)
- [TFMicroInterpreter](#)
- [tls](#)
- [tv](#)
- [TypeError](#)
- [Uint16Array](#)
- [Uint24Array](#)
- [Uint32Array](#)
- [Uint8Array](#)
- [Uint8ClampedArray](#)
- [Unistroke](#)
- [url](#)
- [Waveform](#)
- [Wifi](#)
- [WiOLTE](#)
- [WIZnet](#)
- [WLAN](#)

[Globals](#)

Methods and Fields

[\(top\)](#)

- [variable FILE](#)
- [function acceleration\(\)](#)
- [function analogRead\(pin\)](#)
- [function analogWrite\(pin, value, options\)](#)
- [variable arguments](#)
- [function atob\(base64Data\)](#)
- [Bluetooth](#)
- [function btoa\(binaryData\)](#)
- [function changeInterval\(id, time\)](#)
- [function clearInterval\(id, ...\)](#)
- [function clearTimeout\(id, ...\)](#)
- [function clearWatch\(id, ...\)](#)
- [function compass\(\)](#)
- [function decodeURIComponent\(str\)](#)
- [function digitalPulse\(pin, value, time\)](#)
- [function digitalRead\(pin\)](#)
- [function digitalWrite\(pin, value\)](#)
- [function dump\(\)](#)
- [function echo\(echoOn\)](#)
- [function edit\(funcName\)](#)
- [function encodeURIComponent\(str\)](#)
- [function eval\(code\)](#)
- [variable FET](#)
- [function getPinMode\(pin\)](#)
- [function getSerial\(\)](#)
- [function getTime\(\)](#)
- [variable global](#)
- [variable HIGH](#)
- [I2C1](#)
- [I2C2](#)
- [I2C3](#)
- [variable Infinity](#)
- [function isFinite\(x\)](#)
- [function isNaN\(x\)](#)
- [function load\(filename\)](#)
- [LoopbackA](#)
- [LoopbackB](#)
- [variable LOW](#)
- [variable NaN](#)
- [function parseFloat\(string\)](#)
- [function parseInt\(string, radix\)](#)
- [function peek16\(addr, count\)](#)
- [function peek32\(addr, count\)](#)
- [function peek8\(addr, count\)](#)
- [function pinMode\(pin, mode, automatic\)](#)
- [function poke16\(addr, value\)](#)
- [function poke32\(addr, value\)](#)
- [function poke8\(addr, value\)](#)
- [function print\(text, ...\)](#)
- [function require\(moduleName\)](#)
- [function reset\(clearFlash\)](#)
- [function save\(\)](#)
- [variable SCL](#)
- [variable SDA](#)
- [Serial1](#)
- [Serial2](#)
- [Serial3](#)
- [Serial4](#)
- [Serial5](#)
- [Serial6](#)
- [function setBusyIndicator\(pin\)](#)
- [function setDeepSleep\(sleep\)](#)
- [function setInterval\(function, timeout, args, ...\)](#)
- [function setSleepIndicator\(pin\)](#)
- [function setTime\(time\)](#)
- [function setTimeout\(function, timeout, args, ...\)](#)
- [function setWatch\(function, pin, options\)](#)
- [function shiftOut\(pins, options, data\)](#)
- [function show\(image\)](#)
- [SPI1](#)
- [SPI2](#)

- [SPI3](#)
- [Telnet](#)
- [Terminal](#)
- [function trace\(root\)](#)
- [USB](#)
- [variable VIBRATE](#)

[variable FILE](#) ⇒

Call type:

[\(top\)](#)

```
variable __FILE__
```

Returns

The filename of the JavaScript that is currently executing

Description

The filename of the JavaScript that is currently executing.

If [load](#) has been called with a filename (eg `load("myfile.js")`) then [FILE](#) is set to that filename. Otherwise (eg [load\(\)](#)) or immediately after booting, [FILE](#) is not set.

[function acceleration](#) ⇒

Call type:

[\(top\)](#)

```
function acceleration()
```

Returns

An object with x, y, and z fields in it

Description

Note: This function is only available on the [BBC micro:bit](#) board

Get the current acceleration of the micro:bit from the on-board accelerometer

This is deprecated. Please use [Microbit.accel](#) instead.

Note: This is only available in BBC micro:bit boards

[function analogRead](#) ⇒

Call type:

[\(top\)](#)

```
function analogRead(pin)
```

Parameters

`pin` - The pin to use

You can find out which pins to use by looking at [your board's reference page](#) and searching for pins with the ADC markers.

Returns

The analog Value of the Pin between 0 and 1

Description

Get the analogue value of the given pin

This is different to Arduino which only returns an integer between 0 and 1023

However only pins connected to an ADC will work (see the datasheet)

Note: if you didn't call `pinMode` beforehand then this function will also reset pin's state to "analog"

[function analogWrite](#) ⇒

Call type:

[\(top\)](#)

```
function analogWrite(pin, value, options)
```

Parameters

`pin` - The pin to use

You can find out which pins to use by looking at [your board's reference page](#) and searching for pins with the `PWM` or `DAC` markers.

`value` - A value between 0 and 1

`options` - An object containing options for analog output - see below

Description

Set the analog Value of a pin. It will be output using PWM.

Objects can contain:

- `freq` - pulse frequency in Hz, e.g. `analogWrite(A0,0.5,{ freq : 10 });` - specifying a frequency will force PWM output, even if the pin has a DAC
- `soft` - boolean, If true software PWM is used if hardware is not available.
- `forceSoft` - boolean, If true software PWM is used even if hardware PWM or a DAC is available

Note: if you didn't call `pinMode` beforehand then this function will also reset pin's state to "output"

[variable arguments](#) ⇒

Call type:

[\(top\)](#)

```
variable arguments
```

Returns

An array containing all the arguments given to the function

Description

A variable containing the arguments given to the function:

```
function hello() {
  console.log(arguments.length, JSON.stringify(arguments));
}
hello()      // 0 []
hello("Test") // 1 ["Test"]
hello(1,2,3)  // 3 [1,2,3]
```

Note: Due to the way Espruino works this is doesn't behave exactly the same as in normal JavaScript. The length of the arguments array will never be less than the number of arguments specified in the function declaration: `(function(a){ return arguments.length; })()` == 1. Normal JavaScript interpreters would return 0 in the above case.

[function atob](#) ⇒

Call type:

[\(top\)](#)

```
function atob(base64Data)
```

Parameters

`base64Data` - A string of base64 data to decode

Returns

A string containing the decoded data

Description

Decode the supplied base64 string into a normal string

Note: This is not available in devices with low flash memory

[Bluetooth](#) ⇒

Instance of [Serial](#)

[\(top\)](#)

Description

The Bluetooth Serial port - used when data is sent or received over Bluetooth Smart on nRF51/nRF52 chips.

Note: This is only available in devices with Bluetooth LE capability

[function btoa](#) ⇒

Call type:

[\(top\)](#)

```
function btoa(binaryData)
```

Parameters

`binaryData` - A string of data to encode

Returns

A base64 encoded string

Description

Encode the supplied string (or array) into a base64 string

Note: This is not available in devices with low flash memory

[function changeInterval](#) ⇒

Call type:

[\(top\)](#)

```
function changeInterval(id, time)
```

Parameters

`id` - The id returned by a previous call to `setInterval`

`time` - The new time period in ms

Description

Change the Interval on a callback created with `setInterval`, for example:

```
var id = setInterval(function () { print('foo'); }, 1000); // every second  
changeInterval(id, 1500); // now runs every 1.5 seconds
```

This takes effect immediately and resets the timeout, so in the example above, regardless of when you call `changeInterval`, the next interval will occur 1500ms after it.

[function clearInterval](#) ⇒

Call type:

[\(top\)](#)

```
function clearInterval(id, ...)
```

Parameters

`id, ...` - The id returned by a previous call to setInterval. **Only one argument is allowed.**

Description

Clear the Interval that was created with [setInterval](#), for example:

```
var id = setInterval(function () { print('foo'); }, 1000);
clearInterval(id);
```

If no argument is supplied, all timeouts and intervals are stopped.

To avoid accidentally deleting all Intervals, if a parameter is supplied but is `undefined` then an Exception will be thrown.

[function clearTimeout](#) ⇒

Call type:

```
function clearTimeout(id, ...)
```

Parameters

`id, ...` - The id returned by a previous call to setTimeout. **Only one argument is allowed.**

Description

Clear the Timeout that was created with [setTimeout](#), for example:

```
var id = setTimeout(function () { print('foo'); }, 1000);
clearTimeout(id);
```

If no argument is supplied, all timeouts and intervals are stopped.

To avoid accidentally deleting all Timeouts, if a parameter is supplied but is `undefined` then an Exception will be thrown.

[function clearWatch](#) ⇒

Call type:

```
function clearWatch(id, ...)
```

Parameters

`id, ...` - The id returned by a previous call to setWatch. **Only one argument is allowed.** (or pass nothing to clear all watches)

Description

Clear the Watch that was created with setWatch. If no parameter is supplied, all watches will be removed.

To avoid accidentally deleting all Watches, if a parameter is supplied but is `undefined` then an Exception will be thrown.

[function compass](#) ⇒

Call type:

```
function compass()
```

Returns

An object with x, y, and z fields in it

Description

[\(top\)](#)

[\(top\)](#)

[\(top\)](#)

Note: This function is only available on the [BBC micro:bit](#) board

Get the current compass position for the micro:bit from the on-board magnetometer

This is deprecated. Please use [Microbit.mag](#) instead.

Note: This is only available in BBC micro:bit boards

[function decodeURIComponent](#) ⇒

Call type:

[\(top\)](#)

```
function decodeURIComponent(str)
```

Parameters

str - A string to decode from a URI

Returns

A string containing the decoded data

Description

Convert any groups of characters of the form '%ZZ', into characters with hex code '0xZZ'

Note: This is not available in devices with low flash memory

[function digitalPulse](#) ⇒

Call type:

[\(top\)](#)

```
function digitalPulse(pin, value, time)
```

Parameters

pin - The pin to use

value - Whether to pulse high (true) or low (false)

time - A time in milliseconds, or an array of times (in which case a square wave will be output starting with a pulse of 'value')

Description

Pulse the pin with the value for the given time in milliseconds. It uses a hardware timer to produce accurate pulses, and returns immediately (before the pulse has finished). Use `digitalPulse(A0, 1, 0)` to wait until a previous pulse has finished.

e.g. `digitalPulse(A0, 1, 5);` pulses A0 high for 5ms. `digitalPulse(A0, 1, [5, 2, 4]);` pulses A0 high for 5ms, low for 2ms, and high for 4ms

Note: if you didn't call `pinMode` beforehand then this function will also reset pin's state to "output"

`digitalPulse` is for SHORT pulses that need to be very accurate. If you're doing anything over a few milliseconds, use `setTimeout` instead.

[function digitalRead](#) ⇒

Call type:

[\(top\)](#)

```
function digitalRead(pin)
```

Parameters

pin - The pin to use

Returns

The digital Value of the Pin

Description

Get the digital value of the given pin.

Note: if you didn't call `pinMode` beforehand then this function will also reset pin's state to "input"

If the pin argument is an array of pins (e.g. `[A2,A1,A0]`) the value returned will be a number where the last array element is the least significant bit, for example if `A0=A1=1` and `A2=0, digitalRead([A2,A1,A0]) == 0b011`

If the pin argument is an object with a `read` method, the `read` method will be called and the integer value it returns passed back.

[function digitalWrite](#) ⇒

Call type:

[\(top\)](#)

```
function digitalWrite(pin, value)
```

Parameters

`pin` - The pin to use

`value` - Whether to pulse high (true) or low (false)

Description

Set the digital value of the given pin.

Note: if you didn't call `pinMode` beforehand then this function will also reset pin's state to "output"

If pin argument is an array of pins (e.g. `[A2,A1,A0]`) the value argument will be treated as an array of bits where the last array element is the least significant bit.

In this case, pin values are set least significant bit first (from the right-hand side of the array of pins). This means you can use the same pin multiple times, for example `digitalWrite([A1,A1,A0,A0],0b0101)` would pulse A0 followed by A1.

If the pin argument is an object with a `write` method, the `write` method will be called with the value passed through.

[function dump](#) ⇒

Call type:

[\(top\)](#)

```
function dump()
```

Description

Output current interpreter state in a text form such that it can be copied to a new device

Espruino keeps its current state in RAM (even if the function code is stored in Flash). When you type `dump()`, it dumps the current state of code in RAM plus the hardware state, then if there's code saved in flash it writes "`// Code saved with E.setBootCode`" and dumps that too.

Note: 'Internal' functions are currently not handled correctly. You will need to recreate these in the `onInit` function.

Note: This is not available in devices with low flash memory

[function echo](#) ⇒

Call type:

[\(top\)](#)

```
function echo(echoOn)
```

Parameters

`echoOn` -

Description

Should Espruino echo what you type back to you? true = yes (Default), false = no. When echo is off, the result of executing a command is not returned. Instead, you must use 'print' to send output.

[function edit](#) ⇒

Call type:

```
function edit(funcName)
```

Parameters

funcName - The name of the function to edit (either a string or just the unquoted name)

Description

Fill the console with the contents of the given function, so you can edit it.

NOTE: This is a convenience function - it will not edit 'inner functions'. For that, you must edit the 'outer function' and re-execute it.

Note: This is not available in devices with low flash memory

[function encodeURIComponent](#) ⇒

Call type:

```
function encodeURIComponent(str)
```

Parameters

str - A string to encode as a URI

Returns

A string containing the encoded data

Description

Convert a string with any character not alphanumeric or - _ . ! ~ * ' () converted to the form %xy where xy is its hexadecimal representation

Note: This is not available in devices with low flash memory

[function eval](#) ⇒

Call type:

```
function eval(code)
```

Parameters

code -

Returns

The result of evaluating the string

Description

Evaluate a string containing JavaScript code

[variable FET](#) ⇒

Call type:

```
variable FET
```

Returns

[\(top\)](#)

[\(top\)](#)

[\(top\)](#)

See description above

Description

On Puck.js V2 (not v1.0) this is the pin that controls the FET, for high-powered outputs.

Note: This is only available in Puck.js devices

[function getPinMode](#) ⇒

Call type:

[\(top\)](#)

```
function getPinMode(pin)
```

Parameters

pin - The pin to check

Returns

The pin mode, as a string

Description

Return the current mode of the given pin. See [pinMode](#) for more information on returned values.

Note: This is not available in devices with low flash memory

[function getSerial](#) ⇒

Call type:

[\(top\)](#)

```
function getSerial()
```

Returns

The board's serial number

Description

Get the serial number of this board

[function getTime](#) ⇒

Call type:

[\(top\)](#)

```
function getTime()
```

Returns

See description above

Description

Return the current system time in Seconds (as a floating point number)

[variable global](#) ⇒

Call type:

[\(top\)](#)

```
variable global
```

Returns

The global scope

Description

A reference to the global scope, where everything is defined.

[variable HIGH](#) ⇒

Call type:

[\(top\)](#)

```
variable HIGH
```

Returns

Logic 1 for Arduino compatibility - this is the same as just typing 1

[I2C1](#) ⇒

Instance of [I2C](#)

[\(top\)](#)

Description

The first I2C port

Note: This is only available in devices with more than 1 I2C peripherals

[I2C2](#) ⇒

Instance of [I2C](#)

[\(top\)](#)

Description

The second I2C port

Note: This is only available in devices with more than 2 I2C peripherals

[I2C3](#) ⇒

Instance of [I2C](#)

[\(top\)](#)

Description

The third I2C port

Note: This is only available in devices with more than 3 I2C peripherals

[variable Infinity](#) ⇒

Call type:

[\(top\)](#)

```
variable Infinity
```

Returns

Positive Infinity (1/0)

[function isFinite](#) ⇒

Call type:

```
function isFinite(x)
```

Parameters

x -

Returns

True if the value is a Finite number, false if not.

Description

Is the parameter a finite number or not? If needed, the parameter is first converted to a number.

[function isNaN](#) ⇒

Call type:

```
function isNaN(x)
```

Parameters

x -

Returns

True if the value is NaN, false if not.

Description

Whether the x is NaN (Not a Number) or not

[function load](#) ⇒

Call type:

```
function load(filename)
```

Parameters

filename - [optional] The name of a text JS file to load from Storage after reset

Description

Restart and load the program out of flash - this has an effect similar to completely rebooting Espruino (power off/power on), but without actually performing a full reset of the hardware.

This command only executes when the Interpreter returns to the Idle state - for instance `a=1;load();a=2;` will still leave 'a' as undefined (or what it was set to in the saved program).

Espruino will resume from where it was when you last typed [save\(\)](#). If you want code to be executed right after loading (for instance to initialise devices connected to Espruino), add an `init` event handler to `E` with

```
E.on('init',
  function() { ... your_code ... });
```

. This will then be automatically executed by Espruino every time it starts.

If you specify a filename in the argument then that file will be loaded from Storage after reset in much the same way as calling [reset\(\)](#) then `eval(require("Storage").read(filename))`

[LoopbackA](#) ⇒

Instance of [Serial](#)

[\(top\)](#)

Description

[\(top\)](#)

A loopback serial device. Data sent to [LoopbackA](#) comes out of [LoopbackB](#) and vice versa

[LoopbackB](#) ⇒

Instance of [Serial](#)

[\(top\)](#)

Description

A loopback serial device. Data sent to [LoopbackA](#) comes out of [LoopbackB](#) and vice versa

[variable LOW](#) ⇒

Call type:

[\(top\)](#)

```
variable LOW
```

Returns

Logic 0 for Arduino compatibility - this is the same as just typing 0

[variable NaN](#) ⇒

Call type:

[\(top\)](#)

```
variable NaN
```

Returns

Not a Number

[function parseFloat](#) ⇒

Call type:

[\(top\)](#)

```
function parseFloat(string)
```

Parameters

string -

Returns

The value of the string

Description

Convert a string representing a number into an float

[function parseInt](#) ⇒

Call type:

[\(top\)](#)

```
function parseInt(string, radix)
```

Parameters

string -

`radix` - [optional] The Radix of the string

Returns

The integer value of the string (or NaN)

Description

Convert a string representing a number into an integer

[function peek16](#) ⇒

Call type:

[\(top\)](#)

```
function peek16(addr, count)
```

Parameters

`addr` - The address in memory to read

`count` - [optional] the number of items to read. If >1 a Uint16Array will be returned.

Returns

The value of memory at the given location

Description

Read 16 bits of memory at the given location - DANGEROUS!

[function peek32](#) ⇒

Call type:

[\(top\)](#)

```
function peek32(addr, count)
```

Parameters

`addr` - The address in memory to read

`count` - [optional] the number of items to read. If >1 a Uint32Array will be returned.

Returns

The value of memory at the given location

Description

Read 32 bits of memory at the given location - DANGEROUS!

[function peek8](#) ⇒

Call type:

[\(top\)](#)

```
function peek8(addr, count)
```

Parameters

`addr` - The address in memory to read

`count` - [optional] the number of items to read. If >1 a Uint8Array will be returned.

Returns

The value of memory at the given location

Description

Read 8 bits of memory at the given location - DANGEROUS!

[function pinMode](#) ⇒

Call type:

[\(top\)](#)

```
function pinMode(pin, mode, automatic)
```

Parameters

pin - The pin to set pin mode for

mode - The mode - a string that is either 'analog', 'input', 'input_pullup', 'input_pulldown', 'output', 'opendrain', 'af_output' or 'af_opendrain'. Do not include this argument or use 'auto' if you want to revert to automatic pin mode setting.

automatic - Optional, default is false. If true, subsequent commands will automatically change the state (see notes below)

Description

Set the mode of the given pin.

- **auto/undefined** - Don't change state, but allow [digitalWrite](#)/etc to automatically change state as appropriate
- **analog** - Analog input
- **input** - Digital input
- **input_pullup** - Digital input with internal ~40k pull-up resistor
- **input_pulldown** - Digital input with internal ~40k pull-down resistor
- **output** - Digital output
- **opendrain** - Digital output that only ever pulls down to 0v. Sending a logical 1 leaves the pin open circuit
- **opendrain_pullup** - Digital output that pulls down to 0v. Sending a logical 1 enables internal ~40k pull-up resistor
- **af_output** - Digital output from built-in peripheral
- **af_opendrain** - Digital output from built-in peripheral that only ever pulls down to 0v. Sending a logical 1 leaves the pin open circuit

Note: [digitalRead](#)/[digitalWrite](#)/etc set the pin mode automatically *unless* [pinMode](#) has been called first. If you want [digitalRead](#)/etc to set the pin mode automatically after you have called [pinMode](#), simply call it again with no mode argument (`pinMode(pin)`), **auto** as the argument (

```
pinMode(pin,  
"auto")
```

), or with the 3rd 'automatic' argument set to true (

```
pinMode(pin,  
"output", true)
```

).

[function poke16](#) ⇒

Call type:

[\(top\)](#)

```
function poke16(addr, value)
```

Parameters

addr - The address in memory to write

value - The value to write, or an array of values

Description

Write 16 bits of memory at the given location - VERY DANGEROUS!

[function poke32](#) ⇒

Call type:

[\(top\)](#)

```
function poke32(addr, value)
```

Parameters

addr - The address in memory to write
value - The value to write, or an array of values

Description

Write 32 bits of memory at the given location - VERY DANGEROUS!

[function poke8](#) ⇒

Call type:

[\(top\)](#)

```
function poke8(addr, value)
```

Parameters

addr - The address in memory to write
value - The value to write, or an array of values

Description

Write 8 bits of memory at the given location - VERY DANGEROUS!

[function print](#) ⇒

Call type:

[\(top\)](#)

```
function print(text, ...)
```

Parameters

text, ... -

Description

Print the supplied string(s) to the console

Note: If you're connected to a computer (not a wall adaptor) via USB but **you are not running a terminal app** then when you print data Espruino may pause execution and wait until the computer requests the data it is trying to print.

[function require](#) ⇒

Call type:

[\(top\)](#)

```
function require(moduleName)
```

Parameters

moduleName - A String containing the name of the given module

Returns

The result of evaluating the string

Description

Load the given module, and return the exported functions and variables.

For example:

```
var s = require("Storage");
s.write("test", "hello world");
```

```
print(s.read("test"));
// prints "hello world"
```

Check out [the page on Modules](#) for an explanation of what modules are and how you can use them.

[function reset](#) ⇒

Call type:

```
function reset(clearFlash)
```

Parameters

clearFlash - Remove saved code from flash as well

Description

Reset the interpreter - clear program memory in RAM, and do not load a saved program from flash. This does NOT reset the underlying hardware (which allows you to reset the device without it disconnecting from USB).

This command only executes when the Interpreter returns to the Idle state - for instance `a=1;reset();a=2;` will still leave 'a' as undefined.

The safest way to do a full reset is to hit the reset button.

If `reset()` is called with no arguments, it will reset the board's state in RAM but will not reset the state in flash. When next powered on (or when `load()` is called) the board will load the previously saved code.

Calling `reset(true)` will cause *all saved code in flash memory to be cleared as well*.

[function save](#) ⇒

Call type:

```
function save()
```

([top](#))

Description

Save the state of the interpreter into flash (including the results of calling `setWatch`, `setInterval`, `pinMode`, and any listeners). The state will then be loaded automatically every time Espruino powers on or is hard-reset. To see what will get saved you can call `dump()`.

Note: If you set up intervals/etc in `onInit()` and you have already called `onInit` before running `save()`, when Espruino resumes there will be two copies of your intervals - the ones from before the save, and the ones from after - which may cause you problems.

For more information about this and other options for saving, please see the [Saving code on Espruino](#) page.

This command only executes when the Interpreter returns to the Idle state - for instance `a=1;save();a=2;` will save 'a' as 2.

When Espruino powers on, it will resume from where it was when you typed `save()`. If you want code to be executed right after loading (for instance to initialise devices connected to Espruino), add a function called `onInit`, or add a `init` event handler to `E` with

```
E.on('init', function() { ... your_code
... });
```

. This will then be automatically executed by Espruino every time it starts.

In order to stop the program saved with this command being loaded automatically, check out [the Troubleshooting guide](#)

Note: This is not available in Bangle.js smartwatches

[variable SCL](#) ⇒

Call type:

```
variable SCL
```

([top](#))

Returns

See description above

Description

The pin marked SDA on the Arduino pin footprint. This is connected directly to pin A5.

Note: This is only available in Pixl.js boards

[variable SDA](#) ⇒

Call type:

```
variable SDA
```

Returns

See description above

Description

The pin marked SDA on the Arduino pin footprint. This is connected directly to pin A5.

Note: This is only available in Pixl.js boards

[Serial1](#) ⇒

Instance of [Serial](#)

[\(top\)](#)

Description

The first Serial (USART) port

Note: This is only available in devices with more than 1 USART peripherals

[Serial2](#) ⇒

Instance of [Serial](#)

[\(top\)](#)

Description

The second Serial (USART) port

Note: This is only available in devices with more than 2 USART peripherals

[Serial3](#) ⇒

Instance of [Serial](#)

[\(top\)](#)

Description

The third Serial (USART) port

Note: This is only available in devices with more than 3 USART peripherals

[Serial4](#) ⇒

Instance of [Serial](#)

[\(top\)](#)

Description

The fourth Serial (USART) port

Note: This is only available in devices with more than 4 USART peripherals

[Serial5](#) ⇒

Instance of [Serial](#)

[\(top\)](#)

Description

The fifth Serial (USART) port

Note: This is only available in devices with more than 5 USART peripherals

[Serial6](#) ⇒

Instance of [Serial](#)

[\(top\)](#)

Description

The sixth Serial (USART) port

Note: This is only available in devices with more than 6 USART peripherals

[function setBusyIndicator](#) ⇒

Call type:

[\(top\)](#)

```
function setBusyIndicator(pin)
```

Parameters

pin -

Description

When Espruino is busy, set the pin specified here high. Set this to undefined to disable the feature.

Note: This is not available in devices with low flash memory

[function setDeepSleep](#) ⇒

Call type:

[\(top\)](#)

```
function setDeepSleep(sleep)
```

Parameters

sleep -

Description

Set whether we can enter deep sleep mode, which reduces power consumption to around 100uA. This only works on STM32 Espruino Boards (nRF52 boards sleep automatically).

Please see <http://www.espruino.com/Power+Consumption> for more details on this.

Note: This is only available in STM32 devices (including Espruino Original, Pico and WiFi) and EFM32 devices

[function setInterval](#) ⇒

Call type:

[\(top\)](#)

```
function setInterval(function, timeout, args, ...)
```

Parameters

function - A Function or String to be executed

timeout - The time between calls to the function (max 3153600000000 = 100 years)

`args, ...` - Optional arguments to pass to the function when executed

Returns

An ID that can be passed to `clearInterval`

Description

Call the function (or evaluate the string) specified REPEATEDLY after the timeout in milliseconds.

For instance:

```
setInterval(function () {
  console.log("Hello World");
}, 1000);
// or
setInterval('console.log("Hello World");', 1000);
// both print 'Hello World' every second
```

You can also specify extra arguments that will be sent to the function when it is executed. For example:

```
setInterval(function (a,b) {
  console.log(a+" "+b);
}, 1000, "Hello", "World");
// prints 'Hello World' every second
```

If you want to stop your function from being called, pass the number that was returned by `setInterval` into the `clearInterval` function.

Note: If `setDeepSleep(true)` has been called and the interval is greater than 5 seconds, Espruino may execute the interval up to 1 second late. This is because Espruino can only wake from deep sleep every second - and waking early would cause Espruino to waste power while it waited for the correct time.

[function setSleepIndicator](#) ⇒

Call type:

[\(top\)](#)

```
function setSleepIndicator(pin)
```

Parameters

`pin` -

Description

When Espruino is asleep, set the pin specified here low (when it's awake, set it high). Set this to undefined to disable the feature.

Please see <http://www.espruino.com/Power+Consumption> for more details on this.

Note: This is not available in devices with low flash memory

[function setTime](#) ⇒

Call type:

[\(top\)](#)

```
function setTime(time)
```

Parameters

`time` -

Description

Set the current system time in seconds (`time` can be a floating point value).

This is used with `getTime`, the time reported from `setWatch`, as well as when using `new Date()`.

`Date.prototype.getTime()` reports the time in milliseconds, so you can set the time to a `Date` object using:

```
setTime(new Date("Tue, 19 Feb 2019 10:57")).getTime()/1000
```

To set the timezone for all new Dates, use `E.setTimeZone(hours)`.

[function setTimeout](#) ⇒

Call type:

[\(top\)](#)

```
function setTimeout(function, timeout, args, ...)
```

Parameters

function - A Function or String to be executed

timeout - The time until the function will be executed (max 3153600000000 = 100 years)

args, ... - Optional arguments to pass to the function when executed

Returns

An ID that can be passed to `clearTimeout`

Description

Call the function (or evaluate the string) specified ONCE after the timeout in milliseconds.

For instance:

```
setTimeout(function () {
  console.log("Hello World");
}, 1000);
// or
setTimeout('console.log("Hello World");', 1000);
// both print 'Hello World' after a second
```

You can also specify extra arguments that will be sent to the function when it is executed. For example:

```
setTimeout(function (a,b) {
  console.log(a+" "+b);
}, 1000, "Hello", "World");
// prints 'Hello World' after 1 second
```

If you want to stop the function from being called, pass the number that was returned by `setTimeout` into the `clearTimeout` function.

Note: If `setDeepSleep(true)` has been called and the interval is greater than 5 seconds, Espruino may execute the interval up to 1 second late. This is because Espruino can only wake from deep sleep every second - and waking early would cause Espruino to waste power while it waited for the correct time.

[function setWatch](#) ⇒

Call type:

[\(top\)](#)

```
function setWatch(function, pin, options)
```

Parameters

function - A Function or String to be executed

pin - The pin to watch

options - If a boolean or integer, it determines whether to call this once (false = default) or every time a change occurs (true). Can be an object of the form { repeat: `true/false(default)`, edge:`'rising'/'falling'/'both'(default)`, debounce:10 } - see below for more information.

Returns

An ID that can be passed to `clearWatch`

Description

Call the function specified when the pin changes. Watches set with `setWatch` can be removed using `clearWatch`.

If the `options` parameter is an object, it can contain the following information (all optional):

```
{
  // Whether to keep producing callbacks, or remove the watch after the first callback
  repeat: true/false(default),
  // Trigger on the rising or falling edge of the signal. Can be a string, or 1='rising', -1='falling', 0='both'
  edge:'rising'(default for built-in buttons) / 'falling' / 'both'(default for pins),
  // Use software-debouncing to stop multiple calls if a switch bounces
  // This is the time in milliseconds to wait for bounces to subside, or 0 to disable
  debounce:10 (0 is default for pins, 25 is default for built-in buttons),
```

```

// Advanced: If the function supplied is a 'native' function (compiled or assembly)
// setting irq:true will call that function in the interrupt itself
irq : false(default)
// Advanced: If specified, the given pin will be read whenever the watch is called
// and the state will be included as a 'data' field in the callback
data : pin
// Advanced: On Nordic devices, a watch may be 'high' or 'low' accuracy. By default low
// accuracy is used (which is better for power consumption), but this means that
// high speed pulses (less than 25us) may not be reliably received. Setting hispeed=true
// allows for detecting high speed pulses at the expense of higher idle power consumption
hispeed : true
}

```

The **function** callback is called with an argument, which is an object of type `{state:bool, time:float, lastTime:float}`.

- state is whether the pin is currently a 1 or a 0
- time is the time in seconds at which the pin changed state
- lastTime is the time in seconds at which the **pin last changed state**. When using edge:'rising' or edge:'falling', this is not the same as when the function was last called.
- data is included if data:pin was specified in the options, and can be used for reading in clocked data

For instance, if you want to measure the length of a positive pulse you could use

```
setWatch(function(e) { console.log(e.time-e.lastTime); }, BTN, {
repeat:true, edge:'falling'});

```

. This will only be called on the falling edge of the pulse, but will be able to measure the width of the pulse because e.lastTime is the time of the rising edge.

Internally, an interrupt writes the time of the pin's state change into a queue with the exact time that it happened, and the function supplied to `setWatch` is executed only from the main message loop. However, if the callback is a native function `void (bool state)` then you can add `irq:true` to options, which will cause the function to be called from within the IRQ. When doing this, interrupts will happen on both edges and there will be no debouncing.

Note: if you didn't call `pinMode` beforehand then this function will reset pin's state to "input"

Note: The STM32 chip (used in the [Espruino Board](#) and [Pico](#)) cannot watch two pins with the same number - e.g. A0 and B0.

Note: On nRF52 chips (used in Puck.js, Pixl.js, MDBT42Q) `setWatch` disables the GPIO output on that pin. In order to be able to write to the pin again you need to disable the watch with `clearwatch`.

[function shiftOut](#) ⇒

Call type:

[\(top\)](#)

```
function shiftOut(pins, options, data)
```

Parameters

pins - A pin, or an array of pins to use

options - Options, for instance the clock (see below)

data - The data to shift out (see [E.toInt8Array](#) for info on the forms this can take)

Description

Shift an array of data out using the pins supplied *least significant bit first*, for example:

```

// shift out to single clk+data
shiftOut(A0, { clk : A1 }, [1,0,1,0]);

// shift out a whole byte (like software SPI)
shiftOut(A0, { clk : A1, repeat: 8 }, [1,2,3,4]);

// shift out via 4 data pins
shiftOut([A3,A2,A1,A0], { clk : A4 }, [1,2,3,4]);

```

options is an object of the form:

```
{
  clk : pin, // a pin to use as the clock (undefined = no pin)
  clkPol : bool, // clock polarity - default is 0 (so 1 normally, pulsing to 0 to clock data in)
  repeat : int, // number of clocks per array item
}
```

Each item in the data array will be output to the pins, with the first pin in the array being the MSB and the last the LSB, then the clock will be pulsed in the polarity given.

repeat is the amount of times shift data out for each array item. For instance we may want to shift 8 bits out through 2 pins - in which case we need to set repeat to 4.

[function show](#) ⇒

Call type:[\(top\)](#)

```
function show(image)
```

Parameters

image - The image to show

Description

Note: This function is only available on the [BBC micro:bit](#) board

Show an image on the in-built 5x5 LED screen.

Image can be:

- A number where each bit represents a pixel (so 25 bits). e.g. 5 or 0x1FFFFFF
- A string, e.g: show("10001"). Newlines are ignored, and anything that is not a space or 0 is treated as a 1.
- An array of 4 bytes (more will be ignored), e.g show([1,2,3,0])

For instance the following works for images:

```
show("#  #"+  
" #  "+  
" #  "+  
"#  #"+  
" ##  ")
```

This means you can also use Espruino's graphics library:

```
var g = Graphics.createArrayBuffer(5,5,1)  
g.drawString("E",0,0)  
show(g.buffer)
```

Note: This is only available in BBC micro:bit boards

[**SPI1**](#) ⇒**Instance of [SPI](#)**[\(top\)](#)**Description**

The first SPI port

Note: This is only available in devices with more than 1 SPI peripherals

[**SPI2**](#) ⇒**Instance of [SPI](#)**[\(top\)](#)**Description**

The second SPI port

Note: This is only available in devices with more than 2 SPI peripherals

[**SPI3**](#) ⇒**Instance of [SPI](#)**[\(top\)](#)**Description**

The third SPI port

Note: This is only available in devices with more than 3 SPI peripherals

[**Telnet**](#) ⇒

[Instance of Serial](#)

[\(top\)](#)

Description

A telnet serial device that maps to the built-in telnet console server (devices that have built-in wifi only).

Note: This is only available in devices with Telnet enabled (Linux, ESP8266 and ESP32)

[Terminal](#) ⇒

[Instance of Serial](#)

[\(top\)](#)

Description

A simple VT100 terminal emulator.

When data is sent to the `Terminal` object, `Graphics.getInstance()` is called and if an instance of `graphics` is found then characters are written to it.

Note: This is only available in devices with VT100 terminal emulation enabled (Pixl.js only)

[function trace](#) ⇒

Call type:

[\(top\)](#)

```
function trace(root)
```

Parameters

`root` - The symbol to output (optional). If nothing is specified, everything will be output

Description

Output debugging information

Note: This is not included on boards with low amounts of flash memory, or the Espruino board.

Note: This is not available in devices with low flash memory

[USB](#) ⇒

[Instance of Serial](#)

[\(top\)](#)

Description

The USB Serial port

Note: This is only available in devices with USB

[variable VIBRATE](#) ⇒

Call type:

[\(top\)](#)

```
variable VIBRATE
```

Returns

See description above

Description

The Bangle.js's vibration motor.

Note: This is only available in Bangle.js smartwatches

[AES Class](#)

Class containing AES encryption/decryption

[\(top\)](#)

Note: This library is currently only included in builds for boards where there is space. For other boards there is `crypto.js` which implements SHA1 in JS.

Methods and Fields

- [AES.decrypt\(passphrase, key, options\)](#)
- [AES.encrypt\(passphrase, key, options\)](#)

[AES.decrypt](#) ⇒

Call type:

[\(top\)](#)

`AES.decrypt(passphrase, key, options)`

Parameters

`passphrase` - Message to decrypt

`key` - Key to encrypt message - must be an ArrayBuffer of 128, 192, or 256 BITS

`options` - [optional] An object, may specify { `iv : new Uint8Array(16), mode : 'CBC|CFB|CTR|OFB|ECB'` }

Returns

Returns an ArrayBuffer

[AES.encrypt](#) ⇒

Call type:

[\(top\)](#)

`AES.encrypt(passphrase, key, options)`

Parameters

`passphrase` - Message to encrypt

`key` - Key to encrypt message - must be an ArrayBuffer of 128, 192, or 256 BITS

`options` - [optional] An object, may specify { `iv : new Uint8Array(16), mode : 'CBC|CFB|CTR|OFB|ECB'` }

Returns

Returns an ArrayBuffer

[Array Class](#)

This is the built-in JavaScript class for arrays.

(top)

Arrays can be defined with `[]`, `new Array()`, or

```
new  
Array(length)
```

Methods and Fields

- [constructor Array\(args, ...\)](#)
- [function Array.concat\(args, ...\)](#)
- [function Array.every\(function, thisArg\)](#)
- [function Array.fill\(value, start, end\)](#)
- [function Array.filter\(function, thisArg\)](#)
- [function Array.find\(function\)](#)
- [function Array.findIndex\(function\)](#)
- [function Array.forEach\(function, thisArg\)](#)
- [function Array.includes\(value, startIndex\)](#)
- [function Array.indexOf\(value, startIndex\)](#)
- [Array.isArray\(var\)](#)
- [function Array.join\(separator\)](#)
- [property Array.length](#)
- [function Array.map\(function, thisArg\)](#)
- [function Array.pop\(\)](#)
- [function Array.push\(arguments, ...\)](#)
- [function Array.reduce\(callback, initialValue\)](#)
- [function Array.reverse\(\)](#)
- [function Array.shift\(\)](#)
- [function Array.slice\(start, end\)](#)
- [function Array.some\(function, thisArg\)](#)
- [function Array.sort\(var\)](#)
- [function Array.splice\(index, howMany, elements, ...\)](#)
- [function Array.toString\(radix\)](#)
- [function Array.unshift\(elements, ...\)](#)

[constructor Array](#) ⇒

[View MDN documentation](#)

Call type:

(top)

```
new Array(args, ...)
```

Parameters

`args, ...` - The length of the array OR any number of items to add to the array

Returns

An Array

Description

Create an Array. Either give it one integer argument (≥ 0) which is the length of the array, or any number of arguments

[function Array.concat](#) ⇒

[View MDN documentation](#)

Call type:

(top)

```
function Array.concat(args, ...)
```

Parameters

`args`, ... - Any items to add to the array

Returns

An Array

Description

Create a new array, containing the elements from this one and any arguments, if any argument is an array then those elements will be added.

Note: This is not available in devices with low flash memory

[function Array.every](#) ➔

[View MDN documentation](#)

Call type:

```
function Array.every(function, thisArg)
```

Parameters

`function` - Function to be executed

`thisArg` - [optional] If specified, the function is called with 'this' set to thisArg

Returns

A boolean containing the result

Description

Return 'true' if the callback returns 'true' for every element in the array

[function Array.fill](#) ➔

[View MDN documentation](#)

Call type:

```
function Array.fill(value, start, end)
```

[\(top\)](#)

Parameters

`value` - The value to fill the array with

`start` - Optional. The index to start from (or 0). If start is negative, it is treated as length+start where length is the length of the array

`end` - Optional. The index to end at (or the array length). If end is negative, it is treated as length+end.

Returns

This array

Description

Fill this array with the given value, for every index $\geq \text{start}$ and $< \text{end}$

Note: This is not available in devices with low flash memory

[function Array.filter](#) ➔

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
function Array.filter(function, thisArg)
```

Parameters

function - Function to be executed

thisArg - [optional] If specified, the function is called with 'this' set to thisArg

Returns

An array containing the results

Description

Return an array which contains only those elements for which the callback function returns 'true'

[function Array.find](#) ⇒

[View MDN documentation](#)

Call type:

```
function Array.find(function)
```

Parameters

function - Function to be executed

Returns

The array element where **function** returns **true**, or **undefined**

Description

Return the array element where **function** returns **true**, or **undefined** if it doesn't returns **true** for any element.

```
["Hello", "There", "World"].find(a=>a[ 0 ]=="T")
// returns "There"
```

Note: This is not available in devices with low flash memory

[function Array.findIndex](#) ⇒

[View MDN documentation](#)

Call type:

```
function Array.findIndex(function)
```

Parameters

function - Function to be executed

Returns

The array element's index where **function** returns **true**, or **-1**

Description

Return the array element's index where **function** returns **true**, or **-1** if it doesn't returns **true** for any element.

```
["Hello", "There", "World"].findIndex(a=>a[ 0 ]=="T")
// returns 1
```

Note: This is not available in devices with low flash memory

[\(top\)](#)

[function Array.forEach](#) ⇒

[View MDN documentation](#)

Call type:

(top)

```
function Array.forEach(function, thisArg)
```

Parameters

function - Function to be executed

thisArg - [optional] If specified, the function is called with 'this' set to thisArg

Description

Executes a provided function once per array element.

[function Array.includes](#) ⇒

[View MDN documentation](#)

Call type:

(top)

```
function Array.includes(value, startIndex)
```

Parameters

value - The value to check for

startIndex - [optional] the index to search from, or 0 if not specified

Returns

true if the array includes the value, **false** otherwise

Description

Return **true** if the array includes the value, **false** otherwise

Note: This is not available in devices with low flash memory

[function Array.indexOf](#) ⇒

[View MDN documentation](#)

Call type:

(top)

```
function Array.indexOf(value, startIndex)
```

Parameters

value - The value to check for

startIndex - [optional] the index to search from, or 0 if not specified

Returns

the index of the value in the array, or -1

Description

Return the index of the value in the array, or -1

[Array.isArray](#) ⇒

[View MDN documentation](#)

Call type:

`Array.isArray(var)`

Parameters

`var` - The variable to be tested

Returns

True if var is an array, false if not.

Description

Returns true if the provided object is an array

[function Array.join](#) ⇒

[View MDN documentation](#)

Call type:

`function Array.join(separator)`

Parameters

`separator` - The separator

Returns

A String representing the Joined array

Description

Join all elements of this array together into one string, using 'separator' between them. e.g. `[1,2,3].join(' ')=='1 2 3'`

[property Array.length](#) ⇒

[View MDN documentation](#)

Call type:

`property Array.length`

Returns

The length of the array

Description

Find the length of the array

[function Array.map](#) ⇒

[View MDN documentation](#)

Call type:

`function Array.map(function, thisArg)`

Parameters

(top)

function - Function used to map one item to another

thisArg - [optional] If specified, the function is called with 'this' set to thisArg

Returns

An array containing the results

Description

Return an array which is made from the following:

```
A.map(function) =  
[function(A[0]), function(A[1]), ...]
```

[function Array.pop](#) =>

[View MDN documentation](#)

Call type:

```
function Array.pop()
```

Returns

The value that is popped off

Description

Remove and return the value on the end of this array.

This is the opposite of `[1,2,3].shift()`, which removes an element from the beginning of the array.

[function Array.push](#) =>

[View MDN documentation](#)

Call type:

```
function Array.push(arguments, ...)
```

Parameters

`arguments, ...` - One or more arguments to add

Returns

The new size of the array

Description

Push a new value onto the end of this array¹

This is the opposite of `[1,2,3].unshift(0)`, which adds one or more elements to the beginning of the array.

[function Array.reduce](#) =>

[View MDN documentation](#)

Call type:

```
function Array.reduce(callback, initialValue)
```

Parameters

`callback` - Function used to reduce the array

[\(top\)](#)

`initialValue` - if specified, the initial value to pass to the function

Returns

The value returned by the last function called

Description

Execute `previousValue=initialValue` and then

```
previousValue =  
callback(previousValue, currentValue, index, array)
```

for each element in the array, and finally return `previousValue`.

Note: This is not available in devices with low flash memory

[function Array.reverse](#) ⇒

[View MDN documentation](#)

Call type:

```
function Array.reverse()
```

Returns

The array, but reversed.

Description

Reverse all elements in this array (in place)

Note: This is not available in devices with low flash memory

[function Array.shift](#) ⇒

[View MDN documentation](#)

Call type:

```
function Array.shift()
```

Parameters

Returns

The element that was removed

Description

Remove and return the first element of the array.

This is the opposite of `[1,2,3].pop()`, which takes an element off the end.

Note: This is not available in devices with low flash memory

[function Array.slice](#) ⇒

[View MDN documentation](#)

Call type:

```
function Array.slice(start, end)
```

Parameters

[\(top\)](#)

[\(top\)](#)

[\(top\)](#)

start - Start index

end - [optional] End index

Returns

A new array

Description

Return a copy of a portion of this array (in a new array)

[function Array.some](#) ➔

[View MDN documentation](#)

Call type:

```
function Array.some(function, thisArg)
```

Parameters

function - Function to be executed

thisArg - [optional] If specified, the function is called with 'this' set to thisArg

Returns

A boolean containing the result

Description

Return 'true' if the callback returns 'true' for any of the elements in the array

[function Array.sort](#) ➔

[View MDN documentation](#)

Call type:

```
function Array.sort(var)
```

Parameters

var - A function to use to compare array elements (or undefined)

Returns

This array object

Description

Do an in-place quicksort of the array

Note: This is not available in devices with low flash memory

[function Array.splice](#) ➔

[View MDN documentation](#)

Call type:

```
function Array.splice(index, howMany, elements, ...)
```

Parameters

[\(top\)](#)

`index` - Index at which to start changing the array. If negative, will begin that many elements from the end

`howMany` - An integer indicating the number of old array elements to remove. If `howMany` is 0, no elements are removed.

`elements, ...` - One or more items to add to the array

Returns

An array containing the removed elements. If only one element is removed, an array of one element is returned.

Description

Both remove and add items to an array

[function Array.toString](#) ⇒

[View MDN documentation](#)

Call type:

```
function Array.toString(radix)
```

Parameters

`radix` - unused

Returns

A String representing the array

Description

Convert the Array to a string

[function Array.unshift](#) ⇒

[View MDN documentation](#)

Call type:

```
function Array.unshift(elements, ...)
```

Parameters

`elements, ...` - One or more items to add to the beginning of the array

Returns

The new array length

Description

Add one or more items to the start of the array, and return its new length.

This is the opposite of `[1,2,3].push(4)`, which puts one or more elements on the end.

Note: This is not available in devices with low flash memory

[\(top\)](#)

[ArrayBuffer Class](#)

This is the built-in JavaScript class for array buffers.

[\(top\)](#)

If you want to access arrays of differing types of data you may also find [DataView](#) useful.

Methods and Fields

- [constructor ArrayBuffer\(byteLength\)](#)
- [property ArrayBuffer.byteLength](#)

[constructor ArrayBuffer](#) ⇒

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
new ArrayBuffer(byteLength)
```

Parameters

byteLength - The length in Bytes

Returns

An ArrayBuffer object

Description

Create an Array Buffer object

[property ArrayBuffer.byteLength](#) ⇒

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
property ArrayBuffer.byteLength
```

Returns

The Length in bytes

Description

The length, in bytes, of the [ArrayBuffer](#)

[ArrayBufferView Class](#)

This is the built-in JavaScript class that is the prototype for:

[\(top\)](#)

- [Uint8Array](#)
- [UintClamped8Array](#)
- [Int8Array](#)
- [Uint16Array](#)
- [Int16Array](#)
- [Uint24Array](#) (Espruino-specific - not standard JS)
- [Uint32Array](#)
- [Int32Array](#)
- [Float32Array](#)
- [Float64Array](#)

If you want to access arrays of differing types of data you may also find [DataView](#) useful.

Methods and Fields

- [property ArrayBufferView.buffer](#)
- [property ArrayBufferView.byteLength](#)
- [property ArrayBufferView.byteOffset](#)
- [function ArrayBufferView.fill\(value, start, end\)](#)
- [function ArrayBufferView.filter\(function, thisArg\)](#)
- [function ArrayBufferView.find\(function\)](#)
- [function ArrayBufferView.findIndex\(function\)](#)
- [function ArrayBufferView.forEach\(function, thisArg\)](#)
- [function ArrayBufferView.includes\(value, startIndex\)](#)
- [function ArrayBufferView.indexOf\(value, startIndex\)](#)
- [function ArrayBufferView.join\(separator\)](#)
- [function ArrayBufferView.map\(function, thisArg\)](#)
- [function ArrayBufferView.reduce\(callback, initialValue\)](#)
- [function ArrayBufferView.reverse\(\)](#)
- [function ArrayBufferView.set\(arr, offset\)](#)
- [function ArrayBufferView.slice\(start, end\)](#)
- [function ArrayBufferView.sort\(var\)](#)
- [function ArrayBufferView.subarray\(begin, end\)](#)

[property ArrayBufferView.buffer](#) ⇒

Call type:

[\(top\)](#)

```
property ArrayBufferView.buffer
```

Returns

An ArrayBuffer object

Description

The buffer this view references

[property ArrayBufferView.byteLength](#) ⇒

Call type:

[\(top\)](#)

```
property ArrayBufferView.byteLength
```

Returns

The Length

Description

The length, in bytes, of the [ArrayBufferView](#)

[property ArrayBufferView.byteOffset](#) ⇒

Call type:

```
property ArrayBufferView.byteOffset
```

Returns

The byte Offset

Description

The offset, in bytes, to the first byte of the view within the backing [ArrayBuffer](#)

[function ArrayBufferView.fill](#) ⇒

Call type:

```
function ArrayBufferView.fill(value, start, end)
```

Parameters

value - The value to fill the array with

start - Optional. The index to start from (or 0). If start is negative, it is treated as length+start where length is the length of the array

end - Optional. The index to end at (or the array length). If end is negative, it is treated as length+end.

Returns

This array

Description

Fill this array with the given value, for every index \geq start and $<$ end

Note: This is not available in devices with low flash memory

[function ArrayBufferView.filter](#) ⇒

Call type:

```
function ArrayBufferView.filter(function, thisArg)
```

Parameters

function - Function to be executed

thisArg - [optional] If specified, the function is called with 'this' set to thisArg

Returns

An array containing the results

Description

Return an array which contains only those elements for which the callback function returns 'true'

Note: This is not available in devices with low flash memory

[function ArrayBufferView.find](#) ⇒

Call type:

(top)

```
function ArrayBufferView.find(function)
```

Parameters

function - Function to be executed

Returns

The array element where **function** returns **true**, or **undefined**

Description

Return the array element where **function** returns **true**, or **undefined** if it doesn't returns **true** for any element.

Note: This is not available in devices with low flash memory

[function ArrayBufferView.findIndex](#) ⇒

Call type:

[\(top\)](#)

```
function ArrayBufferView.findIndex(function)
```

Parameters

function - Function to be executed

Returns

The array element's index where **function** returns **true**, or **-1**

Description

Return the array element's index where **function** returns **true**, or **-1** if it doesn't returns **true** for any element.

Note: This is not available in devices with low flash memory

[function ArrayBufferView.forEach](#) ⇒

Call type:

[\(top\)](#)

```
function ArrayBufferView.forEach(function, thisArg)
```

Parameters

function - Function to be executed

thisArg - [optional] If specified, the function is called with 'this' set to thisArg

Description

Executes a provided function once per array element.

[function ArrayBufferView.includes](#) ⇒

Call type:

[\(top\)](#)

```
function ArrayBufferView.includes(value, startIndex)
```

Parameters

value - The value to check for

startIndex - [optional] the index to search from, or **0** if not specified

Returns

true if the array includes the value, **false** otherwise

Description

Return **true** if the array includes the value, **false** otherwise

Note: This is not available in devices with low flash memory

[function ArrayBufferView.indexOf](#) ⇒

Call type:

```
function ArrayBufferView.indexOf(value, startIndex)
```

Parameters

value - The value to check for

startIndex - [optional] the index to search from, or 0 if not specified

Returns

the index of the value in the array, or -1

Description

Return the index of the value in the array, or -1

[function ArrayBufferView.join](#) ⇒

Call type:

```
function ArrayBufferView.join(separator)
```

Parameters

separator - The separator

Returns

A String representing the Joined array

Description

Join all elements of this array together into one string, using 'separator' between them. e.g. `[1,2,3].join(' ')=='1 2 3'`

[function ArrayBufferView.map](#) ⇒

Call type:

```
function ArrayBufferView.map(function, thisArg)
```

Parameters

function - Function used to map one item to another

thisArg - [optional] If specified, the function is called with 'this' set to thisArg

Returns

An array containing the results

[\(top\)](#)

Description

Return an array which is made from the following:

```
A.map(function) =  
[function(A[0]), function(A[1]), ...]
```

Note: This returns an [ArrayBuffer](#) of the same type it was called on. To get an [Array](#), use [Array.map](#), e.g. `[].map.call(myArray, x=>x+1)`

[function ArrayBufferView.reduce](#) ⇒

Call type:

[\(top\)](#)

```
function ArrayBufferView.reduce(callback, initialValue)
```

Parameters

`callback` - Function used to reduce the array

`initialValue` - if specified, the initial value to pass to the function

Returns

The value returned by the last function called

Description

Execute `previousValue=initialValue` and then

```
previousValue =  
callback(previousValue, currentValue, index, array)
```

for each element in the array, and finally return `previousValue`.

Note: This is not available in devices with low flash memory

[function ArrayBufferView.reverse](#) ⇒

Call type:

[\(top\)](#)

```
function ArrayBufferView.reverse()
```

Returns

This array

Description

Reverse the contents of this [ArrayBufferView](#) in-place

Note: This is not available in devices with low flash memory

[function ArrayBufferView.set](#) ⇒

Call type:

[\(top\)](#)

```
function ArrayBufferView.set(arr, offset)
```

Parameters

`arr` - Floating point index to access

`offset` - [optional] The offset in this array at which to write the values

Description

Copy the contents of `array` into this one, mapping `this[x+offset]=array[x];`

[function ArrayBufferView.slice](#) ⇒

Call type:

[\(top\)](#)

```
function ArrayBufferView.slice(start, end)
```

Parameters

start - Start index

end - [optional] End index

Returns

A new array

Description

Return a copy of a portion of this array (in a new array).

Note: This currently returns a normal [Array](#), not an [ArrayBuffer](#)

Note: This is not available in devices with low flash memory

[function ArrayBufferView.sort](#) ⇒

Call type:

[\(top\)](#)

```
function ArrayBufferView.sort(var)
```

Parameters

var - A function to use to compare array elements (or undefined)

Returns

This array object

Description

Do an in-place quicksort of the array

Note: This is not available in devices with low flash memory

[function ArrayBufferView.subarray](#) ⇒

Call type:

[\(top\)](#)

```
function ArrayBufferView.subarray(begin, end)
```

Parameters

begin - Element to begin at, inclusive. If negative, this is from the end of the array. The entire array is included if this isn't specified

end - Element to end at, exclusive. If negative, it is relative to the end of the array. If not specified the whole array is included

Returns

An [ArrayBufferView](#) of the same type as this one, referencing the same data

Description

Returns a smaller part of this array which references the same data (it doesn't copy it).

Note: This is not available in devices with low flash memory

[Bangle Class](#)

Class containing utility functions for the [Bangle.js Smart Watch](#)

[\(top\)](#)

Methods and Fields

- [event Bangle.accel\(xyz\)](#)
- [Bangle.accelRd\(reg, cnt\)](#)
- [Bangle.accelWr\(reg, data\)](#)
- [event Bangle.aiGesture\(gesture, weights\)](#)
- [Bangle.appRect](#)
- [Bangle.barometerRd\(reg, cnt\)](#)
- [Bangle.barometerWr\(reg, data\)](#)
- [Bangle.beep\(time, freq\)](#)
- [Bangle.buzz\(time, strength\)](#)
- [event Bangle.charging\(charging\)](#)
- [Bangle.compassRd\(reg, cnt\)](#)
- [Bangle.compassWr\(reg, data\)](#)
- [Bangle.debug\(\)](#)
- [event Bangle.drag\(event\)](#)
- [Bangle.drawWidgets\(\)](#)
- [event Bangle.faceUp\(up\)](#)
- [Bangle.factoryReset\(\)](#)
- [event Bangle.gesture\(xyz\)](#)
- [Bangle.getAccel\(\)](#)
- [Bangle.getCompass\(\)](#)
- [Bangle.getGPSFix\(\)](#)
- [Bangle.getHealthStatus\(range\)](#)
- [Bangle.getLCDMode\(\)](#)
- [Bangle.getLogo\(\)](#)
- [Bangle.getOptions\(\)](#)
- [Bangle.getPressure\(\)](#)
- [Bangle.getStepCount\(\)](#)
- [event Bangle.GPS\(fix\)](#)
- [event Bangle.GPS-raw\(nmea, dataLoss\)](#)
- [event Bangle.health\(info\)](#)
- [event Bangle.HRM\(hrm\)](#)
- [event Bangle.HRM-raw\(hrm\)](#)
- [Bangle.hrmRd\(reg, cnt\)](#)
- [Bangle.hrmWr\(reg, data\)](#)
- [Bangle.ioWr\(mask, isOn\)](#)
- [Bangle.isBarometerOn\(\)](#)
- [Bangle.isCharging\(\)](#)
- [Bangle.isCompassOn\(\)](#)
- [Bangle.isGPSOn\(\)](#)
- [Bangle.isHRMOn\(\)](#)
- [Bangle.isLCDOn\(\)](#)
- [Bangle.isLocked\(\)](#)
- [event Bangle.lcdPower\(on\)](#)
- [Bangle.lcdWr\(cmd, data\)](#)
- [Bangle.load\(file\)](#)
- [Bangle.loadWidgets\(\)](#)
- [event Bangle.lock\(on\)](#)
- [event Bangle.mag\(xyz\)](#)
- [event Bangle.midnight\(\)](#)
- [Bangle.off\(\)](#)
- [event Bangle.pressure\(e\)](#)
- [Bangle.project\(latlong\)](#)
- [Bangle.resetCompass\(\)](#)
- [Bangle.setBarometerPower\(isOn, appID\)](#)
- [Bangle.setCompassPower\(isOn, appID\)](#)
- [Bangle.setGPSPower\(isOn, appID\)](#)
- [Bangle.setHRMPower\(isOn, appID\)](#)
- [Bangle.setLCDBrightness\(brightness\)](#)
- [Bangle.setLCDMode\(mode\)](#)
- [Bangle.setLCDOffset\(y\)](#)
- [Bangle.setLCDOverlay\(img, x, y\)](#)
- [Bangle.setLCDPower\(isOn\)](#)
- [Bangle.setLCDTimeout\(isOn\)](#)
- [Bangle.setLocked\(isLocked\)](#)
- [Bangle.setOptions\(options\)](#)
- [Bangle.setPollInterval\(interval\)](#)
- [Bangle.setStepCount\(count\)](#)
- [Bangle.setUI\(type, callback\)](#)
- [Bangle.showClock\(\)](#)

- [Bangle.showLauncher\(\)](#)
- [Bangle.softOff\(\)](#)
- [event Bangle.step\(up\)](#)
- [event Bangle.stroke\(event\)](#)
- [event Bangle.swipe\(directionLR, directionUD\)](#)
- [event Bangle.tap\(data\)](#)
- [event Bangle.touch\(button, xy\)](#)
- [event Bangle.twist\(\)](#)

[event Bangle.accel](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('accel', function(xyz) { ... });
```

Parameters

xyz -

Description

Accelerometer data available with `{x,y,z,diff,mag}` object as a parameter.

- x is X axis (left-right) in g
- y is Y axis (up-down) in g
- z is Z axis (in-out) in g
- diff is difference between this and the last reading in g
- mag is the magnitude of the acceleration in g

You can also retrieve the most recent reading with [Bangle.getAccel\(\)](#).

Note: This is only available in Bangle.js smartwatches

[Bangle.accelRd](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.accelRd(reg, cnt)
```

Parameters

reg -

cnt - If specified, returns an array of the given length (max 128). If not (or 0) it returns a number

Returns

See description above

Description

Reads a register from the accelerometer

Note: On Espruino 2v06 and before this function only returns a number (cnt is ignored).

Note: This is only available in Bangle.js smartwatches

[Bangle.accelWr](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.accelWr(reg, data)
```

Parameters

reg -

data -

Description

Writes a register on the accelerometer

Note: This is only available in Bangle.js smartwatches

[event Bangle.aiGesture](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('aiGesture', function(gesture, weights) { ... });
```

Parameters

`gesture` - The name of the gesture (if '.tfnames' exists, or the index. 'undefined' if not matching)

`weights` - An array of floating point values output by the model

Description

Emitted when a 'gesture' (fast movement) is detected, and a Tensorflow model is in storage in the ".tfmodel" file.

If a ".tfnames" file is specified as a comma-separated list of names, it will be used to decode `gesture` from a number into a string.

Note: This is only available in Bangle.js smartwatches

[Bangle.appRect](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.appRect
```

Returns

An object of the form {`x`,`y`,`w`,`h`,`x2`,`y2`}

Description

Returns the rectangle on the screen that is currently reserved for the app.

Note: This is only available in Bangle.js smartwatches

[Bangle.barometerRd](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.barometerRd(reg, cnt)
```

Parameters

`reg` -

`cnt` - If specified, returns an array of the given length (max 128). If not (or 0) it returns a number

Returns

See description above

Description

Reads a register from the barometer IC

Note: This is only available in DTNO1_F5 and Bangle.js 2 smartwatches and DICKENS

[Bangle.barometerWr](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.barometerWr(reg, data)
```

Parameters

reg -

data -

Description

Writes a register on the barometer IC

Note: This is only available in DTNO1_F5 and Bangle.js 2 smartwatches and DICKENS

[Bangle.beep](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.beep(time, freq)
```

Parameters

time - [optional] Time in ms (default 200)

freq - [optional] Frequency in hz (default 4000)

Returns

A promise, completed when beep is finished

Description

Use the piezo speaker to Beep for a certain time period and frequency

Note: This is only available in Bangle.js smartwatches

[Bangle.buzz](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.buzz(time, strength)
```

Parameters

time - [optional] Time in ms (default 200)

strength - [optional] Power of vibration from 0 to 1 (Default 1)

Returns

A promise, completed when vibration is finished

Description

Use the vibration motor to buzz for a certain time period

Note: This is only available in Bangle.js smartwatches

[event Bangle.charging](#) ⇒

Call type:

```
Bangle.on('charging', function(charging) { ... });
```

[\(top\)](#)

Parameters

charging - **true** if charging

Description

Is the battery charging or not?

Note: This is only available in Bangle.js smartwatches

[Bangle.compassRd](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.compassRd(reg, cnt)
```

Parameters

reg -

cnt - If specified, returns an array of the given length (max 128). If not (or 0) it returns a number

Returns

See description above

Description

Read a register on the Magnetometer/Compass

Note: This is only available in Bangle.js smartwatches

[Bangle.compassWr](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.compassWr(reg, data)
```

Parameters

reg -

data -

Description

Writes a register on the Magnetometer/Compass

Note: This is only available in Bangle.js smartwatches

[Bangle.dbg](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.dbg()
```

Returns

See description above

Description

Reads debug info. Exposes the current values of accHistoryIdx, accGestureCount, accIdleCount, pollInterval and others.

Please see the declaration of this function for more information (click the ==> link above [this description](#))

Note: This is only available in Bangle.js smartwatches

[event Bangle.drag](#) =>

Call type:

[\(top\)](#)

```
Bangle.on('drag', function(event) { ... });
```

Parameters

event - Object of form `{x,y,dx,dy,b}` containing touch coordinates, difference in touch coordinates, and an integer `b` containing number of touch points (currently 1 or 0)

Description

Emitted when the touchscreen is dragged or released

The touchscreen extends past the edge of the screen and while `x` and `y` coordinates are arranged such that they align with the LCD's pixels, if your finger goes towards the edge of the screen, `x` and `y` could end up larger than 175 (the screen's maximum pixel coordinates) or smaller than 0. Coordinates from the touch event are clipped.

Note: This is only available in Bangle.js smartwatches

[Bangle.drawWidgets](#) =>

Call type:

[\(top\)](#)

```
Bangle.drawWidgets()
```

Description

Draw any onscreen widgets that were loaded with [Bangle.loadWidgets\(\)](#).

Widgets should redraw themselves when something changes - you'll only need to call `drawWidgets` if you decide to clear the entire screen with `g.clear()`.

Note: This is only available in Bangle.js smartwatches

Note: This is only available in Bangle.js smartwatches with Bangle.js 2 smartwatches

[event Bangle.faceUp](#) =>

Call type:

[\(top\)](#)

```
Bangle.on('faceUp', function(up) { ... });
```

Parameters

`up` - `true` if face-up

Description

Has the watch been moved so that it is face-up, or not face up?

Note: This is only available in Bangle.js smartwatches

[Bangle.factoryReset](#) =>

Call type:

[\(top\)](#)

```
Bangle.factoryReset()
```

Description

Erase all storage and reload it with the default contents.

This is only available on Bangle.js 2.0. On Bangle.js 1.0 you need to use **Install Default Apps** under the **More...** tab of <http://banglejs.com/apps>

Note: This is only available in Bangle.js 2 smartwatches and EMULATED

[event Bangle.gesture](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('gesture', function(xyz) { ... });
```

Parameters

`xyz` - An Int8Array of XYZXYZXYZ data

Description

Emitted when a 'gesture' (fast movement) is detected

Note: This is only available in Bangle.js smartwatches

[Bangle.getAccel](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.getAccel()
```

Returns

An object containing accelerometer readings (as below)

Description

Get the most recent accelerometer reading. Data is in the same format as the `Bangle.on('accel', event)`.

- `x` is X axis (left-right) in g
- `y` is Y axis (up-down) in g
- `z` is Z axis (in-out) in g
- `diff` is difference between this and the last reading in g (calculated by comparing vectors, not magnitudes)
- `td` is the elapsed
- `mag` is the magnitude of the acceleration in g

Note: This is only available in Bangle.js smartwatches

[Bangle.getCompass](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.getCompass()
```

Returns

An object containing magnetometer readings (as below)

Description

Get the most recent Magnetometer/Compass reading. Data is in the same format as the `Bangle.on('mag', event)`.

Returns an `{x,y,z,dx,dy,dz,heading}` object

- `x/y/z` raw x,y,z magnetometer readings
- `dx/dy/dz` readings based on calibration since magnetometer turned on
- `heading` in degrees based on calibrated readings (will be NaN if magnetometer hasn't been rotated around 360 degrees).

Note: In 2v15 firmware and earlier the heading is inverted (360-heading). There's a fix in the bootloader which will apply a fix for those headings, but old apps may still expect an inverted value.

To get this event you must turn the compass on with `Bangle.setCompassPower(1)`.

Note: This is only available in Bangle.js smartwatches

[Bangle.getGPSFix](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.getGPSFix()
```

Returns

A GPS fix object with `{lat, lon, ...}`

Description

Get the last available GPS fix info (or `undefined` if GPS is off).

The fix info received is the same as you'd get from the [Bangle.GPS](#) event.

Note: This is only available in Bangle.js smartwatches

[Bangle.getHealthStatus](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.getHealthStatus(range)
```

Parameters

`range` - What time period to return data for, see below:

Returns

Returns an object containing various health info

Description

`range` is one of:

- `undefined` or '`10min`' - health data so far in this 10 minute block (eg. 9:00.00 - 9:09.59)
- '`last`' - health data during the last 10 minute block
- '`day`' - the health data so far for the day

`getHealthStatus` returns an object containing:

- `movement` is the 32 bit sum of all `acc.diff` readings since power on (and rolls over). It is the difference in accelerometer values as $g * 8192$
- `steps` is the number of steps during this period
- `bpm` the best BPM reading from HRM sensor during this period
- `bpmConfidence` best BPM confidence (0-100%) during this period

Note: This is only available in Bangle.js smartwatches

[Bangle.getLCDMode](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.getLCDMode()
```

Returns

The LCD mode as a String

Description

The current LCD mode.

See [Bangle.setLCDMode](#) for examples.

Note: This is only available in Bangle.js smartwatches

[Bangle.getLogo](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.getLogo()
```

Returns

An image to be used with `g.drawImage` (as a String)

Description

- On platforms with an LCD of >=8bpp this is 222 x 104 x 2 bits
- Otherwise it's 119 x 56 x 1 bits

Note: This is only available in Bangle.js smartwatches

[Bangle.getOptions](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.getOptions()
```

Returns

The current state of all options

Description

Return the current state of options as set by [Bangle.setOptions](#)

Note: This is only available in Bangle.js smartwatches

[Bangle.getPressure](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.getPressure()
```

Returns

A promise that will be resolved with `{temperature, pressure, altitude}`

Description

Read temperature, pressure and altitude data. A promise is returned which will be resolved with `{temperature, pressure, altitude}`.

If the Barometer has been turned on with [Bangle.setBarometerPower](#) then this will return almost immediately with the reading. If the Barometer is off, conversions take between 500-750ms.

Altitude assumes a sea-level pressure of 1013.25 hPa

```
Bangle.getPressure().then(d=>{
  console.log(d);
  // {temperature, pressure, altitude}
});
```

Note: This is only available in DTNO1_F5 and Bangle.js 2 smartwatches and DICKENS

[Bangle.getStepCount](#) ⇒

Call type:

```
Bangle.getStepCount()
```

[\(top\)](#)

Returns

The number of steps recorded by the step counter

Description

Returns the current amount of steps recorded by the step counter

Note: This is only available in Bangle.js smartwatches

[event Bangle.GPS](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('GPS', function(fix) { ... });
```

Parameters

fix - An object with fix info (see below)

Description

GPS data, as an object. Contains:

```
{ "lat": number,      // Latitude in degrees
  "lon": number,      // Longitude in degrees
  "alt": number,       // altitude in M
  "speed": number,     // Speed in kph
  "course": number,    // Course in degrees
  "time": Date,        // Current Time (or undefined if not known)
  "satellites": 7,     // Number of satellites
  "fix": 1,             // NMEA Fix state - 0 is no fix
  "hdop": number,      // Horizontal Dilution of Precision
}
```

If a value such as `lat` is not known because there is no fix, it'll be [NaN](#).

`hdop` is a value from the GPS receiver that gives a rough idea of accuracy of lat/lon based on the geometry of the satellites in range. Multiply by 5 to get a value in meters. This is just a ballpark estimation and should not be considered remotely accurate.

To get this event you must turn the GPS on with `Bangle.setGPSPower(1)`.

Note: This is only available in Bangle.js smartwatches

[event Bangle.GPS-raw](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('GPS-raw', function(nmea, dataLoss) { ... });
```

Parameters

`nmea` - A string containing the raw NMEA data from the GPS

`dataLoss` - This is set to true if some lines of GPS data have previously been lost (eg because system was too busy to queue up a GPS-raw event)

Description

Raw NMEA GPS / u-blox data messages received as a string

To get this event you must turn the GPS on with `Bangle.setGPSPower(1)`.

Note: This is only available in Bangle.js smartwatches

[event Bangle.health](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('health', function(info) { ... });
```

Parameters

info - An object containing the last 10 minutes health data

Description

See [Bangle.getHealthStatus\(\)](#) for more information. This is used for health tracking to allow Bangle.js to record historical exercise data.

Note: This is only available in Bangle.js smartwatches

[event Bangle.HRM](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('HRM', function(hrm) { ... });
```

Parameters

hrm - An object with heart rate info (see below)

Description

Heart rate data, as an object. Contains:

```
{ "bpm": number,           // Beats per minute
  "confidence": number,    // 0-100 percentage confidence in the heart rate
  "raw": Uint8Array,        // raw samples from heart rate monitor
}
```

To get this event you must turn the heart rate monitor on with [Bangle.setHRMPower\(1\)](#).

Note: This is only available in Bangle.js smartwatches

[event Bangle.HRM-raw](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('HRM-raw', function(hrm) { ... });
```

Parameters

hrm - A object containing instant readings from the heart rate sensor

Description

Called when heart rate sensor data is available - see [Bangle.setHRMPower\(1\)](#).

hrm is of the form:

```
{ "raw": -1,           // raw value from sensor
  "filt": -1,          // bandpass-filtered raw value from sensor
  "bpm": 88.9,         // last BPM value measured
  "confidence": 0 // confidence in the BPM value
}
```

Note: This is only available in Bangle.js smartwatches

[Bangle.hrmRd](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.hrmRd(reg, cnt)
```

Parameters

reg -

`cnt` - If specified, returns an array of the given length (max 128). If not (or 0) it returns a number

Returns

See description above

Description

Read a register on the Heart rate monitor

Note: This is only available in Bangle.js smartwatches

[Bangle.hrmWr](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.hrmWr(reg, data)
```

Parameters

`reg` -

`data` -

Description

Writes a register on the Heart rate monitor

Note: This is only available in Bangle.js 2 smartwatches

[Bangle.ioWr](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.ioWr(mask, isOn)
```

Parameters

`mask` -

`isOn` -

Description

Changes a pin state on the IO expander

Note: This is only available in Bangle.js 1 smartwatches

[Bangle.isBarometerOn](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.isBarometerOn()
```

Returns

Is the Barometer on?

Description

Is the Barometer powered?

Set power with `Bangle.setBarometerPower(...);`

Note: This is only available in DTNO1_F5 and Bangle.js 2 smartwatches and DICKENS

[Bangle.isCharging](#) ⇒

Call type:

`Bangle.isCharging()`

Returns

Is the battery charging or not?

[Bangle.isCompassOn](#) ⇒

Call type:

`Bangle.isCompassOn()`

Returns

Is the Compass on?

Description

Is the compass powered?

Set power with `Bangle.setCompassPower(...);`

Note: This is only available in Bangle.js smartwatches

[Bangle.isGPSOn](#) ⇒

Call type:

`Bangle.isGPSOn()`

Returns

Is the GPS on?

Description

Is the GPS powered?

Set power with `Bangle.setGPSPower(...);`

Note: This is only available in Bangle.js smartwatches

[Bangle.isHRMOn](#) ⇒

Call type:

`Bangle.isHRMOn()`

Returns

Is HRM on?

Description

Is the Heart rate monitor powered?

Set power with `Bangle.setHRMPower(...);`

Note: This is only available in Bangle.js smartwatches

[\(top\)](#)

[\(top\)](#)

[\(top\)](#)

[Bangle.isLCDOn](#) ⇒

Call type:

```
Bangle.isLCDOn()
```

Returns

Is the display on or not?

Description

Also see the [Bangle.lcdPower](#) event

Note: This is only available in Bangle.js smartwatches

[\(top\)](#)

[Bangle.isLocked](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.isLocked()
```

Returns

Is the screen locked or not?

Description

Also see the [Bangle.lock](#) event

Note: This is only available in Bangle.js smartwatches

[event Bangle.lcdPower](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('lcdPower', function(on) { ... });
```

Parameters

on - **true** if screen is on

Description

Has the screen been turned on or off? Can be used to stop tasks that are no longer useful if nothing is displayed. Also see [Bangle.isLCDOn\(\)](#).

Note: This is only available in Bangle.js smartwatches

[Bangle.lcdWr](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.lcdWr(cmd, data)
```

Parameters

cmd -

data -

Description

Writes a command directly to the ST7735 LCD controller

Note: This is only available in Bangle.js smartwatches

[Bangle.load](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.load(file)
```

Parameters

`file` - [optional] A string containing the file name for the app to be loaded

Description

This behaves the same as the global `load()` function, but if fast loading is possible (`Bangle.setUI` was called with a `remove` handler) then instead of a complete reload, the `remove` handler will be called and the new app will be loaded straight after with `eval`.

This should only be used if the app being loaded also uses widgets (eg it contains a `Bangle.loadWidgets()` call).

`load()` is slower, but safer. As such, care should be taken when using `Bangle.load()` with `Bangle.setUI({..., remove:...})` as if your remove handler doesn't completely clean up after your app, memory leaks or other issues could occur - see `Bangle.setUI` for more information.

Note: This is only available in Bangle.js smartwatches

[Bangle.loadWidgets](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.loadWidgets()
```

Description

Load all widgets from flash Storage. Call this once at the beginning of your application if you want any on-screen widgets to be loaded.

They will be loaded into a global `WIDGETS` array, and can be rendered with `Bangle.drawWidgets`.

Note: This is only available in Bangle.js smartwatches

[event Bangle.lock](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('lock', function(on) { ... });
```

Parameters

`on` - `true` if screen is locked, `false` if it is unlocked and touchscreen/buttons will work

Description

Has the screen been locked? Also see `Bangle.isLocked()`.

Note: This is only available in Bangle.js smartwatches

[event Bangle.mag](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('mag', function(xyz) { ... });
```

Parameters

`xyz` -

Description

Magnetometer/Compass data available with `{x,y,z,dx,dy,dz,heading}` object as a parameter

- `x/y/z` raw x,y,z magnetometer readings
- `dx/dy/dz` readings based on calibration since magnetometer turned on
- `heading` in degrees based on calibrated readings (will be NaN if magnetometer hasn't been rotated around 360 degrees).

Note: In 2v15 firmware and earlier the heading is inverted (360-heading). There's a fix in the bootloader which will apply a fix for those headings, but old apps may still expect an inverted value.

To get this event you must turn the compass on with `Bangle.setCompassPower(1)`.

You can also retrieve the most recent reading with `Bangle.getCompass()`.

Note: This is only available in Bangle.js smartwatches

[event Bangle.midnight](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('midnight', function() { ... });
```

Description

Emitted at midnight (at the point the day health info is reset to 0).

Can be used for housekeeping tasks that don't want to be run during the day.

Note: This is only available in Bangle.js smartwatches

[Bangle.off](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.off()
```

Description

Turn Bangle.js off. It can only be woken by pressing BTN1.

Note: This is only available in Bangle.js smartwatches

[event Bangle.pressure](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.on('pressure', function(e) { ... });
```

Parameters

e - An object containing `{temperature,pressure,altitude}`

Description

When `Bangle.setBarometerPower(true)` is called, this event is fired containing barometer readings.

Same format as `Bangle.getPressure()`.

Note: This is only available in Bangle.js smartwatches

[Bangle.project](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.project(latlong)
```

Parameters

latlong - {lat:..., lon:...}

Returns

{x:..., y:...}

Description

Perform a Spherical [Web Mercator projection](#) of latitude and longitude into x and y coordinates, which are roughly equivalent to meters from {lat:0,lon:0}.

This is the formula used for most online mapping and is a good way to compare GPS coordinates to work out the distance between them.

Note: This is only available in Bangle.js smartwatches

[Bangle.resetCompass](#) ⇒

Call type:

[\(top\)](#)

`Bangle.resetCompass()`

Parameters

Description

Resets the compass minimum/maximum values. Can be used if the compass isn't providing a reliable heading any more.

Note: This is only available in Bangle.js smartwatches

[Bangle.setBarometerPower](#) ⇒

Call type:

[\(top\)](#)

`Bangle.setBarometerPower(isOn, appID)`

Parameters

isOn - True if the barometer IC should be on, false if not

appID - A string with the app's name in, used to ensure one app can't turn off something another app is using

Returns

Is the Barometer on?

Description

Set the power to the barometer IC. Once enabled, [Bangle.pressure](#) events are fired each time a new barometer reading is available.

When on, the barometer draws roughly 50uA

Note: This is only available in DTNO1_F5 and Bangle.js 2 smartwatches and DICKENS

[Bangle.setCompassPower](#) ⇒

Call type:

[\(top\)](#)

`Bangle.setCompassPower(isOn, appID)`

Parameters

isOn - True if the Compass should be on, false if not

appID - A string with the app's name in, used to ensure one app can't turn off something another app is using

Returns

Is the Compass on?

Description

Set the power to the Compass

When on, data is output via the `mag` event on [Bangle](#):

```
Bangle.setCompassPower(true, "myapp");
Bangle.on('mag', print);
```

When on, the compass draws roughly 2mA

Note: This is only available in Bangle.js smartwatches

[Bangle.setGPSPower](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.setGPSPower(isOn, appID)
```

Parameters

`ison` - True if the GPS should be on, false if not

`appID` - A string with the app's name in, used to ensure one app can't turn off something another app is using

Returns

Is the GPS on?

Description

Set the power to the GPS.

When on, data is output via the `GPS` event on [Bangle](#):

```
Bangle.setGPSPower(true, "myapp");
Bangle.on('GPS', print);
```

When on, the GPS draws roughly 20mA

Note: This is only available in Bangle.js smartwatches

[Bangle.setHRMPower](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.setHRMPower(isOn, appID)
```

Parameters

`ison` - True if the heart rate monitor should be on, false if not

`appID` - A string with the app's name in, used to ensure one app can't turn off something another app is using

Returns

Is HRM on?

Description

Set the power to the Heart rate monitor

When on, data is output via the `HRM` event on [Bangle](#):

```
Bangle.setHRMPower(true, "myapp");
Bangle.on('HRM', print);
```

When on, the Heart rate monitor draws roughly 5mA

Note: This is only available in Bangle.js smartwatches

[Bangle.setLCDBrightness](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.setLCDBrightness(brightness)
```

Parameters

brightness - The brightness of Bangle.js's display - from 0(off) to 1(on full)

Description

This function can be used to adjust the brightness of Bangle.js's display, and hence prolong its battery life.

Due to hardware design constraints, software PWM has to be used which means that the display may flicker slightly when Bluetooth is active and the display is not at full power.

Power consumption

- 0 = 7mA
- 0.1 = 12mA
- 0.2 = 18mA
- 0.5 = 28mA
- 0.9 = 40mA (switching overhead)
- 1 = 40mA

Note: This is only available in Bangle.js smartwatches

[Bangle.setLCDMode](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.setLCDMode(mode)
```

Parameters

mode - The LCD mode (See below)

Description

This function can be used to change the way graphics is handled on Bangle.js.

Available options for [Bangle.setLCDMode](#) are:

- [`Bangle.setLCDMode\(\)`](#) or [`Bangle.setLCDMode\("direct"\)`](#) (the default) - The drawable area is 240x240 16 bit. Unbuffered, so draw calls take effect immediately. Terminal and vertical scrolling work (horizontal scrolling doesn't).
- [`Bangle.setLCDMode\("doublebuffered"\)`](#) - The drawable area is 240x160 16 bit, terminal and scrolling will not work. `g.flip()` must be called for draw operations to take effect.
- [`Bangle.setLCDMode\("120x120"\)`](#) - The drawable area is 120x120 8 bit, `g.getPixel`, terminal, and full scrolling work. Uses an offscreen buffer stored on Bangle.js, `g.flip()` must be called for draw operations to take effect.
- [`Bangle.setLCDMode\("80x80"\)`](#) - The drawable area is 80x80 8 bit, `g.getPixel`, terminal, and full scrolling work. Uses an offscreen buffer stored on Bangle.js, `g.flip()` must be called for draw operations to take effect.

You can also call [`Bangle.setLCDMode\(\)`](#) to return to normal, unbuffered "direct" mode.

Note: This is only available in Bangle.js smartwatches

[Bangle.setLCDOffset](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.setLCDOffset(y)
```

Parameters

y - The amount of pixels to shift the LCD up or down

Description

This can be used to move the displayed memory area up or down temporarily. It's used for displaying notifications while keeping the main display contents intact.

Note: This is only available in Bangle.js smartwatches

[Bangle.setLCDOverlay](#) ⇒

Call type:

([top](#))

```
Bangle.setLCDOverlay(img, x, y)
```

Parameters

img - An image

x - The X offset the graphics instance should be overlaid on the screen with

y - The Y offset the graphics instance should be overlaid on the screen with

Description

Overlay an image or graphics instance on top of the contents of the graphics buffer.

This only works on Bangle.js 2 because Bangle.js 1 doesn't have an offscreen buffer accessible from the CPU.

```
// display an alarm clock icon on the screen
var img = require("heatshrink").decompress(atob(`1ss4UBvvv///ovBlMyqoAdv/vAw1V//1qtFAQX/BINXDoPVq/9DAP
/AYIKDrWq0oREAYPW1QAB1IWCBOQxAbQWq04WCAQp6BQeqA4P1AQPq1WggEK1WrBAIkBBQJsCBYO///fBQOoPAcqCwP3BQnwgECCwP9
GwIKCngWC14sB7QKCh4CBCwN/64KDgfACwWn6vWgWysBCwOpwtWJgYsCgGqytVBQYsCLY0lqtqwAsFEINVrR4BFgggBBQosDEINWIQ
YsDEIQ3DFgyhCG4msSYeVFgnrFhMvoAgsEkE/FhEggYWCfgIhDkEACwQKBEIYKBCwSGFBQJxCQwyhBBQTkDqohCBQhCCEIJ1DXwrKE
BQoWHBQdacaCwugJoi4CCwgKECwJ9CJgIKDq+qBYUqlWtBQf+BYIAC3/VBQX/tQKDz/9BQY5BAAVV/4WCBOQjcbKwVf+oHBv4wCAAYhB`));
Bangle.setLCDOverlay(img,66,66);
```

Or use a [Graphics](#) instance:

```
var ovr = Graphics.createArrayBuffer(100,100,1,{msb:true}); // 1bpp
ovr.drawLine(0,0,100,100);
ovr.drawRect(0,0,99,99);
Bangle.setLCDOverlay(ovr,38,38);
```

Although [Graphics](#) can be specified directly, it can often make more sense to create an Image from the [Graphics](#) instance, as this gives you access to color palettes and transparent colors. For instance this will draw a colored overlay with rounded corners:

```
var ovr = Graphics.createArrayBuffer(100,100,2,{msb:true});
ovr.setColor(1).fillRect({x:0,y:0,w:99,h:99,r:8});
ovr.setColor(3).fillRect({x:2,y:2,w:95,h:95,r:7});
ovr.setColor(2).setFont("Vector:30").setFontAlign(0,0).drawString("Hi",50,50);
Bangle.setLCDOverlay({
  width:ovr.getWidth(), height:ovr.getHeight(),
  bpp:2, transparent:0,
  palette:new Uint16Array([0,0,g.toColor("#F00"),g.toColor("#FFF")]),
  buffer:ovr.buffer
},38,38);
```

Note: This is only available in Bangle.js 2 smartwatches

[Bangle.setLCDPower](#) ⇒

Call type:

([top](#))

```
Bangle.setLCDPower(isOn)
```

Parameters

ison - True if the LCD should be on, false if not

Description

This function can be used to turn Bangle.js's LCD off or on.

This function resets the Bangle's 'activity timer' (like pressing a button or the screen would) so after a time period of inactivity set by [Bangle.setLCDTimeout](#) the screen will turn off.

If you want to keep the screen on permanently (until apps are changed) you can do:

```
Bangle.setLCDTimeout(0); // turn off the timeout  
Bangle.setLCDPower(1); // keep screen on
```

When on full, the LCD draws roughly 40mA. You can adjust When brightness using [Bangle.setLCDBrightness](#).

Note: This is only available in Bangle.js smartwatches

[Bangle.setLCDTimeout](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.setLCDTimeout(isOn)
```

Parameters

isOn - The timeout of the display in seconds, or 0/[undefined](#) to turn power saving off. Default is 10 seconds.

Description

This function can be used to turn Bangle.js's LCD power saving on or off.

With power saving off, the display will remain in the state you set it with [Bangle.setLCDPower](#).

With power saving on, the display will turn on if a button is pressed, the watch is turned face up, or the screen is updated (see [Bangle.setOptions](#) for configuration). It'll turn off automatically after the given timeout.

Note: This function also sets the Backlight and Lock timeout (the time at which the touchscreen/buttons start being ignored). To set both separately, use [Bangle.setOptions](#)

Note: This is only available in Bangle.js smartwatches

[Bangle.setLocked](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.setLocked(isLocked)
```

Parameters

isLocked - [true](#) if the Bangle is locked (no user input allowed)

Description

This function can be used to lock or unlock Bangle.js (e.g. whether buttons and touchscreen work or not)

Note: This is only available in Bangle.js smartwatches

[Bangle.setOptions](#) ⇒

Call type:

[\(top\)](#)

```
Bangle.setOptions(options)
```

Parameters

options -

Description

Set internal options used for gestures, etc...

- wakeOnBTN1 should the LCD turn on when BTN1 is pressed? default = [true](#)
- wakeOnBTN2 (Bangle.js 1) should the LCD turn on when BTN2 is pressed? default = [true](#)
- wakeOnBTN3 (Bangle.js 1) should the LCD turn on when BTN3 is pressed? default = [true](#)
- wakeOnFaceUp should the LCD turn on when the watch is turned face up? default = [false](#)
- wakeOnTouch should the LCD turn on when the touchscreen is pressed? default = [false](#)

- `wakeOnTwist` should the LCD turn on when the watch is twisted? default = `true`
- `twistThreshold` How much acceleration to register a twist of the watch strap? Can be negative for opposite direction. default = `800`
- `twistMaxY` Maximum acceleration in Y to trigger a twist (low Y means watch is facing the right way up). default = `-800`
- `twistTimeout` How little time (in ms) must a twist take from low->high acceleration? default = `1000`
- `gestureStartThresh` how big a difference before we consider a gesture started? default = `sqr(800)`
- `gestureEndThresh` how small a difference before we consider a gesture ended? default = `sqr(2000)`
- `gestureInactiveCount` how many samples do we keep after a gesture has ended? default = `4`
- `gestureMinLength` how many samples must a gesture have before we notify about it? default = `10`
- `powerSave` after a minute of not being moved, Bangle.js will change the accelerometer poll interval down to 800ms (10x accelerometer samples). On movement it'll be raised to the default 80ms. If `Bangle.setPollInterval` is used this is disabled, and for it to work the poll interval must be either 80ms or 800ms. default = `true`. Setting `powerSave:false` will disable this automatic power saving, but will **not** change the poll interval from its current value. If you desire a specific interval (e.g. the default 80ms) you must set it manually with `Bangle.setPollInterval(80)` after setting `powerSave:false`.
- `lockTimeout` how many milliseconds before the screen locks
- `lcdPowerTimeout` how many milliseconds before the screen turns off
- `backlightTimeout` how many milliseconds before the screen's backlight turns off
- `btnLoadTimeout` how many milliseconds does the home button have to be pressed for before the clock is reloaded? 1500ms default, or 0 means never.
- `hrmPollInterval` set the requested poll interval (in milliseconds) for the heart rate monitor. On Bangle.js 2 only 10,20,40,80,160,200 ms are supported, and polling rate may not be exact. The algorithm's filtering is tuned for 20-40ms poll intervals, so higher/lower intervals may effect the reliability of the BPM reading.
- `hrmSportMode` - on the newest Bangle.js 2 builds with the proprietary heart rate algorithm, this is the sport mode passed to the algorithm. See `libs/misc/vc31_binary/algo.h` for more info. -1 = auto, 0 = normal (default), 1 = running, 2 = ...
- `seaLevelPressure` (Bangle.js 2) Normally 1013.25 millibars - this is used for calculating altitude with the pressure sensor

Where accelerations are used they are in internal units, where `8192 = 1g`

Note: This is only available in Bangle.js smartwatches

[Bangle.setPollInterval](#) ⇒

Call type:

[\(top\)](#)

`Bangle.setPollInterval(interval)`

Parameters

`interval` - Polling interval in milliseconds (Default is 80ms - 12.5Hz to match accelerometer)

Description

Set how often the watch should poll for new acceleration/gyro data and kick the Watchdog timer. It isn't recommended that you make this interval much larger than 1000ms, but values up to 4000ms are allowed.

Calling this will set `Bangle.setOptions({powerSave: false})` - disabling the dynamic adjustment of poll interval to save battery power when Bangle.js is stationary.

Note: This is only available in Bangle.js smartwatches

[Bangle.setStepCount](#) ⇒

Call type:

[\(top\)](#)

`Bangle.setStepCount(count)`

Parameters

`count` - The value with which to reload the step counter

Description

Sets the current value of the step counter

Note: This is only available in Bangle.js smartwatches

[Bangle.setUI](#) ⇒

Call type:

[\(top\)](#)

`Bangle.setUI(type, callback)`

Parameters

`type` - The type of UI input: 'updown', 'leftright', 'clock', 'clockupdown' or undefined to cancel. Can also be an object (see below)

`callback` - A function with one argument which is the direction

Description

This puts Bangle.js into the specified UI input mode, and calls the callback provided when there is user input.

Currently supported interface types are:

- 'updown' - UI input with upwards motion `cb(-1)`, downwards motion `cb(1)`, and select `cb()`
 - Bangle.js 1 uses BTN1/3 for up/down and BTN2 for select
 - Bangle.js 2 uses touchscreen swipe up/down and tap
- 'leftright' - UI input with left motion `cb(-1)`, right motion `cb(1)`, and select `cb()`
 - Bangle.js 1 uses BTN1/3 for left/right and BTN2 for select
 - Bangle.js 2 uses touchscreen swipe left/right and tap/BTN1 for select
- 'clock' - called for clocks. Sets `Bangle.CLOCK=1` and allows a button to start the launcher
 - Bangle.js 1 BTN2 starts the launcher
 - Bangle.js 2 BTN1 starts the launcher
- 'clockupdown' - called for clocks. Sets `Bangle.CLOCK=1`, allows a button to start the launcher, but also provides up/down functionality
 - Bangle.js 1 BTN2 starts the launcher, BTN1/BTN3 call `cb(-1)` and `cb(1)`
 - Bangle.js 2 BTN1 starts the launcher, touchscreen tap in top/bottom right hand side calls `cb(-1)` and `cb(1)`
- `{mode: "custom", ...}` allows you to specify custom handlers for different interactions. See below.
- `undefined` removes all user interaction code

While you could use `setWatch/etc` manually, the benefit here is that you don't end up with multiple `setwatch` instances, and the actual input method (touch, or buttons) is implemented dependent on the watch (Bangle.js 1 or 2)

Note: You can override this function in boot code to change the interaction mode with the watch. For instance you could make all clocks start the launcher with a swipe by using:

```
(function() {
  var sui = Bangle.setUI;
  Bangle.setUI = function(mode, cb) {
    if (mode!="clock") return sui(mode,cb);
    sui(); // clear
    Bangle.CLOCK=1;
    Bangle.swipeHandler = Bangle.showLauncher;
    Bangle.on("swipe", Bangle.swipeHandler);
  };
})();
```

The first argument can also be an object, in which case more options can be specified:

```
Bangle.setUI({
  mode : "custom",
  back : function() {}, // optional - add a 'back' icon in top-left widget area and call this function when it is pressed , also call i
  remove : function() {}, // optional - add a handler for when the UI should be removed (eg stop any intervals/timers here)
  touch : function(n,e) {}, // optional - handler for 'touch' events
  swipe : function(dir) {}, // optional - handler for 'swipe' events
  drag : function(e) {}, // optional - handler for 'drag' events (Bangle.js 2 only)
  btn : function(n) {}, // optional - handler for 'button' events (n==1 on Bangle.js 2, n==1/2/3 depending on button for Bangle.js 1)
  clock : 0 // optional - if set the behavior of 'clock' mode is added (does not override btn if defined)
});
```

If `remove` is specified, `Bangle.showLauncher`, `Bangle.showClock`, `Bangle.load` and some apps may choose to just call the `remove` function and then load a new app without resetting Bangle.js. As a result, **if you specify 'remove' you should make sure you test that after calling `Bangle.setUI()` without arguments your app is completely unloaded**, otherwise you may end up with memory leaks or other issues when switching apps.

Note: This is only available in Bangle.js smartwatches

Note: This is only available in Bangle.js smartwatches with Bangle.js 2 smartwatches

[Bangle.showClock](#)

Call type:

[\(top\)](#)

```
Bangle.showClock()
```

Description

Load the Bangle.js clock - this has the same effect as calling `Bangle.load()`.

Note: This is only available in Bangle.js smartwatches

[Bangle.showLauncher](#)

Call type:

```
Bangle.showLauncher()
```

Description

Load the Bangle.js app launcher, which will allow the user to select an application to launch.

Note: This is only available in Bangle.js smartwatches

Bangle.softOff ⇒**Call type:**

```
Bangle.softOff()
```

Description

Turn Bangle.js (mostly) off, but keep the CPU in sleep mode until BTN1 is pressed to preserve the RTC (current time).

Note: This is only available in Bangle.js smartwatches

event Bangle.step ⇒**Call type:**

```
Bangle.on('step', function(up) { ... });
```

Parameters

up - The number of steps since Bangle.js was last reset

Description

Called whenever a step is detected by Bangle.js's pedometer.

Note: This is only available in Bangle.js smartwatches

event Bangle.stroke ⇒**Call type:**

```
Bangle.on('stroke', function(event) { ... });
```

Parameters

event - Object of form `{xy:Int8Array([x1,y1,x2,y2...])}` containing touch coordinates

Description

Emitted when the touchscreen is dragged for a large enough distance to count as a gesture.

If Bangle.strokes is defined and populated with data from [Unistroke.new](#), the `event` argument will also contain a `stroke` field containing the most closely matching stroke name.

For example:

```
Bangle.strokes = {
  up : Unistroke.new(new Uint8Array([57, 151, ... 158, 137])),
  alpha : Unistroke.new(new Uint8Array([161, 55, ... 159, 161])),
};

Bangle.on('stroke', o=>{
  print(o.stroke);
  g.clear(1).drawPoly(o.xy);
});
// Might print something like
{
  "xy": new Uint8Array([149, 50, ... 107, 136]),
```

[\(top\)](#)

```
        "stroke": "alpha"
    }
```

Note: This is only available in Bangle.js 2 smartwatches

event Bangle.swipe ⇒

Call type:

[\(top\)](#)

```
Bangle.on('swipe', function(directionLR, directionUD) { ... });
```

Parameters

directionLR - -1 for left, 1 for right, 0 for up/down

directionUD - -1 for up, 1 for down, 0 for left/right (Bangle.js 2 only)

Description

Emitted when a swipe on the touchscreen is detected (a movement from left->right, right->left, down->up or up->down)

Bangle.js 1 is only capable of detecting left/right swipes as it only contains a 2 zone touchscreen.

Note: This is only available in Bangle.js smartwatches

event Bangle.tap ⇒

Call type:

[\(top\)](#)

```
Bangle.on('tap', function(data) { ... });
```

Parameters

data - {dir, double, x, y, z}

Description

If the watch is tapped, this event contains information on the way it was tapped.

dir reports the side of the watch that was tapped (not the direction it was tapped in).

```
{
  dir : "left/right/top/bottom/front/back",
  double : true/false // was this a double-tap?
  x : -2 .. 2, // the axis of the tap
  y : -2 .. 2, // the axis of the tap
  z : -2 .. 2 // the axis of the tap
```

Note: This is only available in Bangle.js smartwatches

event Bangle.touch ⇒

Call type:

[\(top\)](#)

```
Bangle.on('touch', function(button, xy) { ... });
```

Parameters

button - 1 for left, 2 for right

xy - Object of form {x,y,type} containing touch coordinates (if the device supports full touch). Clipped to 0..175 (LCD pixel coordinates) on firmware 2v13 and later. type is only available on Bangle.js 2 and is an integer, either 0 for swift touches or 2 for longer ones.

Description

Emitted when the touchscreen is pressed

Note: This is only available in Bangle.js smartwatches

[event Bangle.twist](#) ⇒

Call type:

```
Bangle.on('twist', function() { ... });
```

Description

This event happens when the watch has been twisted around it's axis - for instance as if it was rotated so someone could look at the time.

To tweak when this happens, see the `twist*` options in [Bangle.setOptions\(\)](#).

Note: This is only available in Bangle.js smartwatches

[\(top\)](#)

[BluetoothDevice Class](#)

A Web Bluetooth-style device - you can request one using `NRF.requestDevice(address)`

[\(top\)](#)

For example:

```
var gatt;
NRF.requestDevice({ filters: [{ name: 'Puck.js abcd' }] }).then(function(device) {
  console.log("found device");
  return device.gatt.connect();
}).then(function(g) {
  gatt = g;
  console.log("connected");
  return gatt.startBonding();
}).then(function() {
  console.log("bonded", gatt.getSecurityStatus());
  gatt.disconnect();
}).catch(function(e) {
  console.log("ERROR",e);
});
```

Methods and Fields

- [property BluetoothDevice.gatt](#)
- [event BluetoothDevice.gattserverdisconnected\(reason\)](#)
- [event BluetoothDevice.passkey\(passkey\)](#)
- [event BluetoothDevice.passkeyRequest\(\)](#)
- [property BluetoothDevice.rssi](#)
- [function BluetoothDevice.sendPasskey\(passkey\)](#)

[property BluetoothDevice.gatt](#) ⇒

Call type:

[\(top\)](#)

```
property BluetoothDevice.gatt
```

Returns

A [BluetoothRemoteGATTServer](#) for this device

[event BluetoothDevice.gattserverdisconnected](#) ⇒

Call type:

[\(top\)](#)

```
BluetoothDevice.on('gattserverdisconnected', function(reason) { ... });
```

Parameters

`reason` - The reason code reported back by the BLE stack - see Nordic's `ble_hci.h` file for more information

Description

Called when the device gets disconnected.

To connect and then print `Disconnected` when the device is disconnected, just do the following:

```
var gatt;
NRF.connect("aa:bb:cc:dd:ee:ff").then(function(gatt) {
  gatt.device.on('gattserverdisconnected', function(reason) {
    console.log("Disconnected ",reason);
  });
});
```

Or:

```
var gatt;
NRF.requestDevice(...).then(function(device) {
  device.on('gattserverdisconnected', function(reason) {
    console.log("Disconnected ",reason);
  });
});
```

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

[event BluetoothDevice.passkey](#) ⇒

Call type:

[\(top\)](#)

```
BluetoothDevice.on('passkey', function(passkey) { ... });
```

Parameters

passkey - A 6 character numeric String to be displayed

Description

Called when the device pairs and sends a passkey that Espruino should display.

For this to be used, you'll have to specify that there's a display using [NRF.setSecurity](#)

This is not part of the Web Bluetooth Specification. It has been added specifically for Espruino.

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

[event BluetoothDevice.passkeyRequest](#) ⇒

Call type:

[\(top\)](#)

```
BluetoothDevice.on('passkeyRequest', function() { ... });
```

Description

Called when the device pairs, displays a passkey, and wants Espruino to tell it what the passkey was.

Respond with [BluetoothDevice.sendPasskey\(\)](#) with a 6 character string containing only 0..9.

For this to be used, you'll have to specify that there's a keyboard using [NRF.setSecurity](#)

This is not part of the Web Bluetooth Specification. It has been added specifically for Espruino.

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

[property BluetoothDevice.rssi](#) ⇒

Call type:

[\(top\)](#)

```
property BluetoothDevice.rssi
```

Returns

The last received RSSI (signal strength) for this device

[function BluetoothDevice.sendPasskey](#) ⇒

Call type:

[\(top\)](#)

```
function BluetoothDevice.sendPasskey(passkey)
```

Parameters

passkey - A 6 character numeric String to be returned to the device

Description

To be used as a response when the event [BluetoothDevice.sendPasskey](#) has been received.

This is not part of the Web Bluetooth Specification. It has been added specifically for Espruino.

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

[BluetoothRemoteGattCharacteristic Class](#)

Web Bluetooth-style GATT characteristic - get this using `BluetoothRemoteGattService.getCharacteristic(s)`

[\(top\)](#)

<https://webbluetoothcg.github.io/web-bluetooth/#bluetoothremotegattcharacteristic>

Methods and Fields

- [event BluetoothRemoteGattCharacteristic.characteristicvaluechanged\(\)](#)
- [function BluetoothRemoteGattCharacteristic.readValue\(\)](#)
- [property BluetoothRemoteGattCharacteristic.service](#)
- [function BluetoothRemoteGattCharacteristic.startNotifications\(\)](#)
- [function BluetoothRemoteGattCharacteristic.stopNotifications\(\)](#)
- [function BluetoothRemoteGattCharacteristic.writeValue\(data\)](#)

[event BluetoothRemoteGattCharacteristic.characteristicvaluechanged](#) ⇒

Call type:

[\(top\)](#)

```
BluetoothRemoteGattCharacteristic.on('characteristicvaluechanged', function() { ... });
```

Description

Called when a characteristic's value changes, after `BluetoothRemoteGattCharacteristic.startNotifications` has been called.

```
...
  return service.getCharacteristic("characteristic_uuid");
}).then(function(c) {
  c.on('characteristicvaluechanged', function(event) {
    console.log("=> " + event.target.value);
  });
  return c.startNotifications();
}).then(...
```

The first argument is of the form

```
{target :  
  BluetoothRemoteGattCharacteristic}
```

, and `BluetoothRemoteGattCharacteristic.value` will then contain the new value (as a DataView).

Note: This is only available in devices with Bluetooth LE capability

[function BluetoothRemoteGattCharacteristic.readValue](#) ⇒

Call type:

[\(top\)](#)

```
function BluetoothRemoteGattCharacteristic.readValue()
```

Returns

A [Promise](#) that is resolved (or rejected) with a [DataView](#) when the characteristic is read

Description

Read a characteristic's value, return a promise containing a [DataView](#)

```
var device;  
NRF.connect(device_address).then(function(d) {  
  device = d;  
  return d.getPrimaryService("service_uuid");  
}).then(function(s) {  
  console.log("Service ", s);  
  return s.getCharacteristic("characteristic_uuid");  
}).then(function(c) {  
  return c.readValue();  
}).then(function(d) {  
  console.log("Got:", JSON.stringify(d.buffer));  
  device.disconnect();  
}).catch(function() {
```

```
    console.log("Something's broken.");
});
```

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) and ESP32 boards

[property BluetoothRemoteGattCharacteristic.service](#) ⇒

Call type:

(top)

```
property BluetoothRemoteGattCharacteristic.service
```

Returns

The [BluetoothRemoteGattService](#) this Service came from

[function BluetoothRemoteGattCharacteristic.startNotifications](#) ⇒

Call type:

(top)

```
function BluetoothRemoteGattCharacteristic.startNotifications()
```

Returns

A [Promise](#) that is resolved (or rejected) with data when notifications have been added

Description

Starts notifications - whenever this characteristic's value changes, a `characteristicvaluechanged` event is fired and `characteristic.value` will then contain the new value as a [DataView](#).

```
var device;
NRF.connect(device_address).then(function(d) {
  device = d;
  return d.getPrimaryService("service_uuid");
}).then(function(s) {
  console.log("Service ",s);
  return s.getCharacteristic("characteristic_uuid");
}).then(function(c) {
  c.on('characteristicvaluechanged', function(event) {
    console.log("-> ",event.target.value); // this is a DataView
  });
  return c.startNotifications();
}).then(function(d) {
  console.log("Waiting for notifications");
}).catch(function() {
  console.log("Something's broken.");
});
```

For example, to listen to the output of another Puck.js's Nordic Serial port service, you can use:

```
var gatt;
NRF.connect("pu:ck:js:ad:dr:es random").then(function(g) {
  gatt = g;
  return gatt.getPrimaryService("6e400001-b5a3-f393-e0a9-e50e24dcca9e");
}).then(function(service) {
  return service.getCharacteristic("6e400003-b5a3-f393-e0a9-e50e24dcca9e");
}).then(function(characteristic) {
  characteristic.on('characteristicvaluechanged', function(event) {
    console.log("RX: "+JSON.stringify(event.target.value.buffer));
  });
  return characteristic.startNotifications();
}).then(function() {
  console.log("Done!");
});
```

Note: This is only available in devices with Bluetooth LE capability

[function BluetoothRemoteGattCharacteristic.stopNotifications](#) ⇒

Call type:

(top)

```
function BluetoothRemoteGattCharacteristic.stopNotifications()
```

Returns

A [promise](#) that is resolved (or rejected) with data when notifications have been removed

Description

Stop notifications (that were requested with [BluetoothRemoteGATTCharacteristic.startNotifications](#))

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

[function BluetoothRemoteGATTCharacteristic.writeValue](#) ⇒

Call type:

[\(top\)](#)

```
function BluetoothRemoteGATTCharacteristic.writeValue(data)
```

Parameters

data - The data to write

Returns

A [promise](#) that is resolved (or rejected) when the characteristic is written

Description

Write a characteristic's value

```
var device;
NRF.connect(device_address).then(function(d) {
  device = d;
  return d.getPrimaryService("service_uuid");
}).then(function(s) {
  console.log("Service ",s);
  return s.getCharacteristic("characteristic_uuid");
}).then(function(c) {
  return c.writeValue("Hello");
}).then(function(d) {
  device.disconnect();
}).catch(function() {
  console.log("Something's broken.");
});
```

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) and ESP32 boards

[BluetoothRemoteGATTServer Class](#)

Web Bluetooth-style GATT server - get this using `NRF.connect(address)` or `NRF.requestDevice(options)` and `response.gatt.connect`

[\(top\)](#)

<https://webbluetoothcg.github.io/web-bluetooth/#bluetoothremotegattserver>

Methods and Fields

- [function BluetoothRemoteGATTServer.connect\(options\)](#)
- [property BluetoothRemoteGATTServer.connected](#)
- [function BluetoothRemoteGATTServer.disconnect\(\)](#)
- [function BluetoothRemoteGATTServer.getPrimaryService\(service\)](#)
- [function BluetoothRemoteGATTServer.getPrimaryServices\(\)](#)
- [function BluetoothRemoteGATTServer.getSecurityStatus\(\)](#)
- [property BluetoothRemoteGATTServer.handle](#)
- [function BluetoothRemoteGATTServer.setRSSIHandler\(callback\)](#)
- [function BluetoothRemoteGATTServer.startBonding\(forceRePair\)](#)

[function BluetoothRemoteGATTServer.connect](#) ⇒

Call type:

[\(top\)](#)

```
function BluetoothRemoteGATTServer.connect(options)
```

Parameters

`options` - [optional] (Espruino-specific) An object of connection options (see below)

Returns

A [Promise](#) that is resolved (or rejected) when the connection is complete

Description

Connect to a BLE device - returns a promise, the argument of which is the [BluetoothRemoteGATTServer](#) connection.

See [NRF.requestDevice](#) for usage examples.

`options` is an optional object containing:

```
{
  minInterval // min connection interval in milliseconds, 7.5 ms to 4 s
  maxInterval // max connection interval in milliseconds, 7.5 ms to 4 s
}
```

By default the interval is 20-200ms (or 500-1000ms if `NRF.setLowPowerConnection(true)` was called). During connection Espruino negotiates with the other device to find a common interval that can be used.

For instance calling:

```
NRF.requestDevice({ filters: [{ namePrefix: 'Pixel.js' }] }).then(function(device) {
  return device.gatt.connect({minInterval:7.5, maxInterval:7.5});
}).then(function(g) {
```

will force the connection to use the fastest connection interval possible (as long as the device at the other end supports it).

Note: The Web Bluetooth spec states that if a device hasn't advertised its name, when connected to a device the central (in this case Espruino) should automatically retrieve the name from the corresponding characteristic (`0x2a00` on service `0x1800`). Espruino does not automatically do this.

Note: This is only available in NRF52 devices (like Puck.js, Pixel.js, Bangle.js and MDBT42Q) and ESP32 boards

[property BluetoothRemoteGATTServer.connected](#) ⇒

Call type:

[\(top\)](#)

```
property BluetoothRemoteGATTServer.connected
```

Returns

Whether the device is connected or not

[function BluetoothRemoteGATTServer.disconnect](#) ⇒

Call type:

[\(top\)](#)

```
function BluetoothRemoteGATTServer.disconnect()
```

Returns

A [Promise](#) that is resolved (or rejected) when the disconnection is complete (non-standard)

Description

Disconnect from a previously connected BLE device connected with [BluetoothRemoteGATTServer.connect](#) - this does not disconnect from something that has connected to the Espruino.

Note: While `.disconnect` is standard Web Bluetooth, in the spec it returns undefined not a [Promise](#) for implementation reasons. In Espruino we return a [Promise](#) to make it easier to detect when Espruino is free to connect to something else.

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) and ESP32 boards

[function BluetoothRemoteGATTServer.getPrimaryService](#) ⇒

Call type:

[\(top\)](#)

```
function BluetoothRemoteGATTServer.getPrimaryService(service)
```

Parameters

service - The service UUID

Returns

A [Promise](#) that is resolved (or rejected) when the primary service is found (the argument contains a [BluetoothRemoteGattService](#))

Description

See [NRF.connect](#) for usage examples.

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) and ESP32 boards

[function BluetoothRemoteGATTServer.getPrimaryServices](#) ⇒

Call type:

[\(top\)](#)

```
function BluetoothRemoteGATTServer.getPrimaryServices()
```

Returns

A [Promise](#) that is resolved (or rejected) when the primary services are found (the argument contains an array of [BluetoothRemoteGattService](#))

[function BluetoothRemoteGATTServer.getSecurityStatus](#) ⇒

Call type:

[\(top\)](#)

```
function BluetoothRemoteGATTServer.getSecurityStatus()
```

Returns

An object

Description

Return an object with information about the security state of the current connection:

```
{  
  connected      // The connection is active (not disconnected).  
  encrypted     // Communication on this link is encrypted.  
  mitm_protected // The encrypted communication is also protected against man-in-the-middle attacks.  
  bonded         // The peer is bonded with us  
}
```

See [BluetoothRemoteGATTServer.startBonding](#) for information about negotiating a secure connection.

This is not part of the Web Bluetooth Specification. It has been added specifically for Puck.js.

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

[property BluetoothRemoteGATTServer.handle](#) ⇒

Call type:

[\(top\)](#)

```
property BluetoothRemoteGATTServer.handle
```

Returns

The handle to this device (if it is currently connected) - the handle is an internal value used by the Bluetooth Stack

[function BluetoothRemoteGATTServer.setRSSIHandler](#) ⇒

Call type:

[\(top\)](#)

```
function BluetoothRemoteGATTServer.setRSSIHandler(callback)
```

Parameters

`callback` - The callback to call with the RSSI value, or undefined to stop

Description

Start/stop listening for RSSI values on the active GATT connection

```
// Start listening for RSSI value updates  
gattServer.setRSSIHandler(function(rssi) {  
  console.log(rssi); // prints -85 (or similar)  
});  
// Stop listening  
gattServer.setRSSIHandler();
```

RSSI is the 'Received Signal Strength Indication' in dBm

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) and ESP32 boards

[function BluetoothRemoteGATTServer.startBonding](#) ⇒

Call type:

[\(top\)](#)

```
function BluetoothRemoteGATTServer.startBonding(forceRePair)
```

Parameters

`forceRePair` - If the device is already bonded, re-pair it

Returns

A [Promise](#) that is resolved (or rejected) when the bonding is complete

Description

Start negotiating bonding (secure communications) with the connected device, and return a Promise that is completed on success or failure.

```
var gatt;  
NRF.requestDevice({ filters: [{ name: 'Puck.js abcd' }] }).then(function(device) {
```

```
console.log("found device");
return device.gatt.connect();
}).then(function(g) {
gatt = g;
console.log("connected");
return gatt.startBonding();
}).then(function() {
console.log("bonded", gatt.getSecurityStatus());
gatt.disconnect();
}).catch(function(e) {
console.log("ERROR",e);
});
```

This is not part of the Web Bluetooth Specification. It has been added specifically for Espruino.

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

[BluetoothRemoteGATTService Class](#)

Web Bluetooth-style GATT service - get this using `BluetoothRemoteGATTServer.getPrimaryService(s)`

[\(top\)](#)

<https://webbluetoothcg.github.io/web-bluetooth/#bluetoothremotegattservice>

Methods and Fields

- [`property BluetoothRemoteGATTService.device`](#)
- [`function BluetoothRemoteGATTService.getCharacteristic\(characteristic\)`](#)
- [`function BluetoothRemoteGATTService.getCharacteristics\(\)`](#)

[property BluetoothRemoteGATTService.device](#) ⇒

Call type:

[\(top\)](#)

`property BluetoothRemoteGATTService.device`

Returns

The `BluetoothDevice` this Service came from

[function BluetoothRemoteGATTService.getCharacteristic](#) ⇒

Call type:

[\(top\)](#)

`function BluetoothRemoteGATTService.getCharacteristic(characteristic)`

Parameters

`characteristic` - The characteristic UUID

Returns

A `Promise` that is resolved (or rejected) when the characteristic is found (the argument contains a `BluetoothRemoteGattCharacteristic`)

Description

See `NRF.connect` for usage examples.

Note: This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) and ESP32 boards

[function BluetoothRemoteGATTService.getCharacteristics](#) ⇒

Call type:

[\(top\)](#)

`function BluetoothRemoteGATTService.getCharacteristics()`

Returns

A `Promise` that is resolved (or rejected) when the characteristic is found (the argument contains an array of `BluetoothRemoteGattCharacteristic`)

[Boolean Class](#)

Methods and Fields

[\(top\)](#)

- [constructor Boolean\(value\)](#)

[constructor Boolean](#) =>

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
new Boolean(value)
```

Parameters

value - A single value to be converted to a number

Returns

A Boolean object

Description

Creates a boolean

[CC3000 Library](#)

Methods and Fields

[\(top\)](#)

- [`require\("CC3000"\).connect\(spi, cs, en, irq\)`](#)

[CC3000.connect](#) ⇒

Call type:

[\(top\)](#)

```
require("CC3000").connect(spi, cs, en, irq)
```

Parameters

`spi` - Device to use for SPI (or undefined to use the default). SPI should be 1,000,000 baud, and set to 'mode 1'

`cs` - The pin to use for Chip Select

`en` - The pin to use for Enable

`irq` - The pin to use for Interrupts

Returns

A WLAN Object

Description

Initialise the CC3000 and return a WLAN object

[console Class](#)

An Object that contains functions for writing to the interactive console

[\(top\)](#)

Methods and Fields

- [console.log\(text,...\)](#)

[console.log](#) ⇒

Call type:

[\(top\)](#)

```
console.log(text, ...)
```

Parameters

`text, ...` - One or more arguments to print

Description

Print the supplied string(s) to the console

Note: If you're connected to a computer (not a wall adaptor) via USB but **you are not running a terminal app** then when you print data Espruino may pause execution and wait until the computer requests the data it is trying to print.

[crypto Library](#)

Cryptographic functions

[\(top\)](#)

Note: This library is currently only included in builds for boards where there is space. For other boards there is `crypto.js` which implements SHA1 in JS.

Methods and Fields

- [`require\("crypto"\).AES`](#)
- [`require\("crypto"\).PBKDF2\(passphrase, salt, options\)`](#)
- [`require\("crypto"\).SHA1\(message\)`](#)
- [`require\("crypto"\).SHA224\(message\)`](#)
- [`require\("crypto"\).SHA256\(message\)`](#)
- [`require\("crypto"\).SHA384\(message\)`](#)
- [`require\("crypto"\).SHA512\(message\)`](#)

[crypto.AES](#) →

Call type:

[\(top\)](#)

```
require("crypto").AES
```

Returns

See description above

Description

Class containing AES encryption/decryption

Note: This is only available in devices that support AES (Espruino Pico, Espruino WiFi or Linux)

[crypto.PBKDF2](#) →

Call type:

[\(top\)](#)

```
require("crypto").PBKDF2(passphrase, salt, options)
```

Parameters

`passphrase` - Passphrase

`salt` - Salt for turning passphrase into a key

`options` - Object of Options, { `keySize: 8` (`in 32 bit words`), `iterations: 10`, `hasher: 'SHA1'/'SHA224'/'SHA256'/'SHA384'/'SHA512'` }

Returns

Returns an ArrayBuffer

Description

Password-Based Key Derivation Function 2 algorithm, using SHA512

Note: This is only available in devices with TLS and SSL support (Espruino Pico and Espruino WiFi only)

[crypto.SHA1](#) →

Call type:

[\(top\)](#)

```
require("crypto").SHA1(message)
```

Parameters

`message` - The message to apply the hash to

Returns

Returns a 20 byte ArrayBuffer

Description

Performs a SHA1 hash and returns the result as a 20 byte ArrayBuffer.

Note: On some boards (currently only Espruino Original) there isn't space for a fully unrolled SHA1 implementation so a slower all-JS implementation is used instead.

Note: This is only available in devices that support Crypto Functionality (Espruino Pico, Original, Espruino WiFi, Espruino BLE devices, Linux or ESP8266)

[crypto.SHA224](#) ⇒

Call type:

[\(top\)](#)

```
require("crypto").SHA224(message)
```

Parameters

`message` - The message to apply the hash to

Returns

Returns a 20 byte ArrayBuffer

Description

Performs a SHA224 hash and returns the result as a 28 byte ArrayBuffer

Note: This is only available in devices that support SHA256 (Espruino Pico, Espruino WiFi, Espruino BLE devices or Linux)

[crypto.SHA256](#) ⇒

Call type:

[\(top\)](#)

```
require("crypto").SHA256(message)
```

Parameters

`message` - The message to apply the hash to

Returns

Returns a 20 byte ArrayBuffer

Description

Performs a SHA256 hash and returns the result as a 32 byte ArrayBuffer

Note: This is only available in devices that support SHA256 (Espruino Pico, Espruino WiFi, Espruino BLE devices or Linux)

[crypto.SHA384](#) ⇒

Call type:

[\(top\)](#)

```
require("crypto").SHA384(message)
```

Parameters

`message` - The message to apply the hash to

Returns

Returns a 20 byte ArrayBuffer

Description

Performs a SHA384 hash and returns the result as a 48 byte ArrayBuffer

Note: This is only available in devices that support SHA512 (Espruino Pico, Espruino WiFi, Espruino BLE devices or Linux)

[crypto.SHA512](#) ⇒

Call type:

[\(top\)](#)

```
require("crypto").SHA512(message)
```

Parameters

`message` - The message to apply the hash to

Returns

Returns a 32 byte ArrayBuffer

Description

Performs a SHA512 hash and returns the result as a 64 byte ArrayBuffer

Note: This is only available in devices that support SHA512 (Espruino Pico, Espruino WiFi, Espruino BLE devices or Linux)

[DataView Class](#)

This class helps

[\(top\)](#)

Methods and Fields

- [constructor DataView\(buffer, byteOffset, byteLength\)](#)
- [function DataView.getFloat32\(byteOffset, littleEndian\)](#)
- [function DataView.getFloat64\(byteOffset, littleEndian\)](#)
- [function DataView.getInt16\(byteOffset, littleEndian\)](#)
- [function DataView.getInt32\(byteOffset, littleEndian\)](#)
- [function DataView.getInt8\(byteOffset, littleEndian\)](#)
- [function DataView.getUint16\(byteOffset, littleEndian\)](#)
- [function DataView.getUint32\(byteOffset, littleEndian\)](#)
- [function DataView.getUint8\(byteOffset, littleEndian\)](#)
- [function DataView.setFloat32\(byteOffset, value, littleEndian\)](#)
- [function DataView.setFloat64\(byteOffset, value, littleEndian\)](#)
- [function DataView.setInt16\(byteOffset, value, littleEndian\)](#)
- [function DataView.setInt32\(byteOffset, value, littleEndian\)](#)
- [function DataView.setInt8\(byteOffset, value, littleEndian\)](#)
- [function DataView.setUint16\(byteOffset, value, littleEndian\)](#)
- [function DataView.setUint32\(byteOffset, value, littleEndian\)](#)
- [function DataView.setUint8\(byteOffset, value, littleEndian\)](#)

[constructor DataView](#) ⇒

[View MDN documentation](#)

Call type:

```
new DataView(buffer, byteOffset, byteLength)
```

Parameters

buffer - The [ArrayBuffer](#) to base this on

byteOffset - [optional] The offset of this view in bytes

byteLength - [optional] The length in bytes

Returns

A [DataView](#) object

Description

Create a [DataView](#) object that can be used to access the data in an [ArrayBuffer](#).

```
var b = new ArrayBuffer(8)
var v = new DataView(b)
v.setInt16(0, "0x1234")
v.setInt8(3, "0x56")
console.log("0x"+v.getUint32(0).toString(16))
// prints 0x12340056
```

Note: This is not available in devices with low flash memory

[function DataView.getFloat32](#) ⇒

[View MDN documentation](#)

Call type:

```
function DataView.getFloat32(byteOffset, littleEndian)
```

Parameters

[\(top\)](#)

`byteOffset` - The offset in bytes to read from

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

Returns

the index of the value in the array, or -1

[function DataView.getFloat64](#) ⇒

[View MDN documentation](#)

Call type:

```
function DataView.getFloat64(byteOffset, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

Returns

the index of the value in the array, or -1

[function DataView.getInt16](#) ⇒

[View MDN documentation](#)

Call type:

```
function DataView.getInt16(byteOffset, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

Returns

the index of the value in the array, or -1

[function DataView.getInt32](#) ⇒

[View MDN documentation](#)

Call type:

```
function DataView.getInt32(byteOffset, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

Returns

the index of the value in the array, or -1

[function DataView.getInt8](#) ⇒

(top)

[View MDN documentation](#)

Call type:

```
function DataView.getInt8(byteOffset, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

Returns

the index of the value in the array, or -1

function DataView.getUint16 →

[View MDN documentation](#)

Call type:

```
function DataView.getUint16(byteOffset, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

Returns

the index of the value in the array, or -1

function DataView.getUint32 →

[View MDN documentation](#)

Call type:

```
function DataView.getUint32(byteOffset, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

Returns

the index of the value in the array, or -1

function DataView.getUint8 →

[View MDN documentation](#)

Call type:

```
function DataView.getUint8(byteOffset, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

(top)

Returns

the index of the value in the array, or -1

[function DataView.setFloat32](#) ⇒

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
function DataView.setFloat32(byteOffset, value, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`value` - The value to write

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

[function DataView.setFloat64](#) ⇒

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
function DataView.setFloat64(byteOffset, value, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`value` - The value to write

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

[function DataView.setInt16](#) ⇒

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
function DataView.setInt16(byteOffset, value, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`value` - The value to write

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

[function DataView.setInt32](#) ⇒

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
function DataView.setInt32(byteOffset, value, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`value` - The value to write

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

[function DataView.setInt8](#) ⇒

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
function DataView.setInt8(byteOffset, value, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`value` - The value to write

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

[function DataView.setUint16](#) ⇒

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
function DataView.setUint16(byteOffset, value, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`value` - The value to write

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

[function DataView.setUint32](#) ⇒

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
function DataView.setUint32(byteOffset, value, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`value` - The value to write

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

[function DataView.setUint8](#) ⇒

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
function DataView.setUint8(byteOffset, value, littleEndian)
```

Parameters

`byteOffset` - The offset in bytes to read from

`value` - The value to write

`littleEndian` - [optional] Whether to read in little endian - if false or undefined data is read as big endian

[Date Class](#)

The built-in class for handling Dates.

[\(top\)](#)

Note: By default the time zone is GMT+0, however you can change the timezone using the `E.setTimeZone(...)` function.

For example `E.setTimeZone(1)` will be GMT+0100

However if you have daylight savings time set with `E.setDST(...)` then the timezone set by `E.setTimeZone(...)` will be *ignored*.

Methods and Fields

- [`constructor Date\(args, ...\)`](#)
- [`function Date.getDate\(\)`](#)
- [`function Date.getDay\(\)`](#)
- [`function Date.getFullYear\(\)`](#)
- [`function Date.getHours\(\)`](#)
- [`function Date.getIsDST\(\)`](#)
- [`function Date.getMilliseconds\(\)`](#)
- [`function Date.getMinutes\(\)`](#)
- [`function Date.getMonth\(\)`](#)
- [`function Date.getSeconds\(\)`](#)
- [`function Date.getTime\(\)`](#)
- [`function Date.getTimezoneOffset\(\)`](#)
- [`Date.now\(\)`](#)
- [`Date.parse\(str\)`](#)
- [`function Date.setDate\(dayValue\)`](#)
- [`function Date.setFullYear\(yearValue, monthValue, dayValue\)`](#)
- [`function Date.setHours\(hoursValue, minutesValue, secondsValue, millisecondsValue\)`](#)
- [`function Date.setMilliseconds\(millisecondsValue\)`](#)
- [`function Date.setMinutes\(minutesValue, secondsValue, millisecondsValue\)`](#)
- [`function Date.setMonth\(yearValue, dayValue\)`](#)
- [`function Date.setSeconds\(secondsValue, millisecondsValue\)`](#)
- [`function Date.setTime\(timeValue\)`](#)
- [`function Date.toISOString\(\)`](#)
- [`function Date.toJSON\(\)`](#)
- [`function Date.toLocalISOString\(\)`](#)
- [`function Date.toString\(\)`](#)
- [`function Date.toUTCString\(\)`](#)
- [`function Date.valueOf\(\)`](#)

[constructor Date](#) ⇒

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
new Date(args, ...)
```

Parameters

`args, ...` - Either nothing (current time), one numeric argument (milliseconds since 1970), a date string (see [Date.parse](#)), or [year, month, day, hour, minute, second, millisecond]

Returns

A Date object

Description

Creates a date object

[function Date.getDate](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.getDate()
```

Returns

See description above

Description

Day of the month 1..31

[function Date.getDay](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.getDay()
```

Returns

See description above

Description

Day of the week (0=sunday, 1=monday, etc)

[function Date.getFullYear](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.getFullYear()
```

Returns

See description above

Description

The year, e.g. 2014

[function Date.getHours](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.getHours()
```

Returns

See description above

Description

0..23

[function Date.getIsDST](#) ⇒

[\(top\)](#)

Call type:[\(top\)](#)

```
function Date.getIsDST()
```

Returns

true if daylight savings time is in effect

Description

This returns a boolean indicating whether daylight savings time is in effect.

Note: This is not available in devices with low flash memory

[function Date.getMilliseconds](#) ⇒[View MDN documentation](#)**Call type:**[\(top\)](#)

```
function Date.getMilliseconds()
```

Returns

See description above

Description

0..999

[function Date.getMinutes](#) ⇒[View MDN documentation](#)**Call type:**[\(top\)](#)

```
function Date.getMinutes()
```

Returns

See description above

Description

0..59

[function Date.getMonth](#) ⇒[View MDN documentation](#)**Call type:**[\(top\)](#)

```
function Date.getMonth()
```

Returns

See description above

Description

Month of the year 0..11

[function Date.getSeconds](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.getSeconds()
```

Returns

See description above

Description

0..59

[function Date.getTime](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.getTime()
```

Returns

See description above

Description

Return the number of milliseconds since 1970

[function Date.getTimezoneOffset](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.getTimezoneOffset()
```

Returns

The difference, in minutes, between UTC and local time

Description

This returns the time-zone offset from UTC, in minutes.

Note: This is not available in devices with low flash memory

[Date.now](#) ⇒

[View MDN documentation](#)

Call type:

```
Date.now()
```

Returns

See description above

Description

(top)

Get the number of milliseconds elapsed since 1970 (or on embedded platforms, since startup)

[Date.parse](#) ⇒

[View MDN documentation](#)

Call type:

```
date.parse(str)
```

Parameters

str - A String

Returns

The number of milliseconds since 1970

Description

Parse a date string and return milliseconds since 1970. Data can be either '2011-10-20T14:48:00', '2011-10-20' or 'Mon, 25 Dec 1995 13:30:00 +0430'

[function Date.setDate](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.setDate(dayValue)
```

Parameters

dayValue - the day of the month, between 0 and 31

Returns

The number of milliseconds since 1970

Description

Day of the month 1..31

Note: This is not available in devices with low flash memory

[function Date.setFullYear](#) ⇒

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
function Date.setFullYear(yearValue, monthValue, dayValue)
```

Parameters

yearValue - The full year - eg. 1989

monthValue - [optional] the month, between 0 and 11

dayValue - [optional] the day, between 0 and 31

Returns

The number of milliseconds since 1970

[function Date.setHours](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.setHours(hoursValue, minutesValue, secondsValue, millisecondsValue)
```

Parameters

hoursValue - number of hours, 0..23
minutesValue - number of minutes, 0..59
secondsValue - [optional] number of seconds, 0..59
millisecondsValue - [optional] number of milliseconds, 0..999

Returns

The number of milliseconds since 1970

Description

0..23

Note: This is not available in devices with low flash memory

[function Date.setMilliseconds](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.setMilliseconds(millisecondsValue)
```

Parameters

millisecondsValue - number of milliseconds, 0..999

Returns

The number of milliseconds since 1970

[function Date.setMinutes](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.setMinutes(minutesValue, secondsValue, millisecondsValue)
```

Parameters

minutesValue - number of minutes, 0..59
secondsValue - [optional] number of seconds, 0..59
millisecondsValue - [optional] number of milliseconds, 0..999

Returns

The number of milliseconds since 1970

Description

0..59

[\(top\)](#)

Note: This is not available in devices with low flash memory

[function Date.setMonth](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.setMonth(yearValue, dayValue)
```

Parameters

yearValue - The month, between 0 and 11

dayValue - [optional] the day, between 0 and 31

Returns

The number of milliseconds since 1970

Description

Month of the year 0..11

Note: This is not available in devices with low flash memory

[function Date.setSeconds](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.setSeconds(secondsValue, millisecondsValue)
```

Parameters

secondsValue - number of seconds, 0..59

millisecondsValue - [optional] number of milliseconds, 0..999

Returns

The number of milliseconds since 1970

Description

0..59

Note: This is not available in devices with low flash memory

[function Date.setTime](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.setTime(timeValue)
```

Parameters

timeValue - the number of milliseconds since 1970

Returns

the number of milliseconds since 1970

[\(top\)](#)

Description

Set the time/date of this Date class

[function Date.toISOString](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.toISOString()
```

Returns

A String

Description

Converts to a ISO 8601 String, e.g: 2014-06-20T14:52:20.123Z

Note: This always assumes a timezone of GMT

[function Date.toJSON](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.toJSON()
```

Returns

A String

Description

Calls [Date.toISOString](#) to output this date to JSON

[function Date.toLocaleISOString](#) ⇒

Call type:

```
function Date.toLocaleISOString()
```

Returns

A String

Description

Converts to a ISO 8601 String (with timezone information), e.g: 2014-06-20T14:52:20.123-0500

Note: This is not available in devices with low flash memory

[function Date.toString](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.toString()
```

Returns

[\(top\)](#)

A String

Description

Converts to a String, e.g: `Fri Jun 20 2014 14:52:20 GMT+0000`

Note: This uses whatever timezone was set with `E.setTimeZone()` or `E.setDST()`.

[function Date.toUTCString](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.toUTCString()
```

Returns

A String

Description

Converts to a String, e.g: `Fri, 20 Jun 2014 14:52:20 GMT`

Note: This always assumes a timezone of GMT

[function Date.valueOf](#) ⇒

[View MDN documentation](#)

Call type:

```
function Date.valueOf()
```

Returns

See description above

Description

Return the number of milliseconds since 1970

[dgram Library](#)

This library allows you to create UDP/DATAGRAM servers and clients

[\(top\)](#)

In order to use this, you will need an extra module to get network connectivity.

This is designed to be a cut-down version of the [node.js library](#). Please see the [Internet](#) page for more information on how to use it.

Methods and Fields

- [require\("dgram"\).createSocket\(type, callback\)](#)

[dgram.createSocket](#) ⇒

Call type:

[\(top\)](#)

```
require("dgram").createSocket(type, callback)
```

Parameters

`type` - Socket type to create e.g. 'udp4'. Or options object { type: 'udp4', reuseAddr: true, recvBufferSize: 1024 }

`callback` - A `function(sckt)` that will be called with the socket when a connection is made. You can then call `sckt.send(...)` to send data, and `sckt.on('message', function(data) { ... })` and `sckt.on('close', function() { ... })` to deal with the response.

Returns

Returns a new `dgram.Socket` object

Description

Create a UDP socket

[dgramSocket Class](#)

An actual socket connection - allowing transmit/receive of TCP data

[\(top\)](#)

Methods and Fields

- [function dgramSocket.addMembership\(group,ip\)](#)
- [function dgramSocket.bind\(port,callback\)](#)
- [function dgramSocket.close\(\)](#)
- [event dgramSocket.close\(had_error\)](#)
- [event dgramSocket.message\(msg,rinfo\)](#)
- [function dgramSocket.send\(buffer,offset,length,args,...\)](#)

[function dgramSocket.addMembership](#) ⇒

Call type:

[\(top\)](#)

```
function dgramSocket.addMembership(group, ip)
```

Parameters

group - A string containing the group ip to join

ip - A string containing the ip to join with

[function dgramSocket.bind](#) ⇒

Call type:

[\(top\)](#)

```
function dgramSocket.bind(port, callback)
```

Parameters

port - The port to bind at

callback - A function(res) that will be called when the socket is bound. You can then call res.on('message', [function\(message, info\) { ... }](#)) and res.on('close', [function\(\) { ... }](#)) to deal with the response.

Returns

The dgramSocket instance that 'bind' was called on

[function dgramSocket.close](#) ⇒

Call type:

[\(top\)](#)

```
function dgramSocket.close()
```

Description

Close the socket

[event dgramSocket.close](#) ⇒

Call type:

[\(top\)](#)

```
dgramSocket.on('close', function\(had\_error\) { ... });
```

Parameters

`had_error` - A boolean indicating whether the connection had an error (use an error event handler to get error details).

Description

Called when the connection closes.

[event dgramSocket.message](#) ⇒

Call type:

[\(top\)](#)

```
dgramSocket.on('message', function(msg, rinfo) { ... });
```

Parameters

`msg` - A string containing the received message

`rinfo` - Sender address,port containing information

Description

The 'message' event is called when a datagram message is received. If a handler is defined with `x.on('message', function(msg) { ... })` then it will be called

[function dgramSocket.send](#) ⇒

Call type:

[\(top\)](#)

```
function dgramSocket.send(buffer, offset, length, args, ...)
```

Parameters

`buffer` - A string containing message to send

`offset` - Offset in the passed string where the message starts [optional]

`length` - Number of bytes in the message [optional]

`args, ...` - Destination port number, Destination IP address string

[E Class](#)

This is the built-in JavaScript class for Espruino utility functions.

[\(top\)](#)

Methods and Fields

- [event E.AMS\(info\)](#)
- [event E.ANCS\(info\)](#)
- [E.asm\(callspec, assemblycode, ...\)](#)
- [E.clip\(x, min, max\)](#)
- [E.compiledC\(code\)](#)
- [E.connectSDCard.spi, csPin\)](#)
- [E.convolve\(arr1, arr2, offset\)](#)
- [E.CRC32\(data\)](#)
- [E.decodeUTF8\(str, lookup, replaceFn\)](#)
- [E.defrag\(\)](#)
- [E.dumpFragmentation\(\)](#)
- [E.dumpFreeList\(\)](#)
- [E.dumpLockedVars\(\)](#)
- [E.dumpStr\(\)](#)
- [E.dumpTimers\(\)](#)
- [E.dumpVariables\(\)](#)
- [E.enableWatchdog\(timeout, isAuto\)](#)
- [event E.errorFlag\(errorFlags\)](#)
- [E.FFT\(arrReal, arrImage, inverse\)](#)
- [E.flashFatFS\(options\)](#)
- [E.getAddressOf\(v, flatAddress\)](#)
- [E.getAnalogVRef\(\)](#)
- [E.getBattery\(\)](#)
- [E.getConsole\(\)](#)
- [E.getErrorFlags\(\)](#)
- [E.getFlags\(\)](#)
- [E.getRTCPrescaler\(calibrate\)](#)
- [E.getSizeOf\(v, depth\)](#)
- [E.getTemperature\(\)](#)
- [E.HSBtoRGB\(hue, sat, bri, format\)](#)
- [E.hwRand\(\)](#)
- [event E.init\(\)](#)
- [E.kickWatchdog\(\)](#)
- [event E.kill\(\)](#)
- [E.lockConsole\(\)](#)
- [E.lookupNoCase\(haystack, needle, returnKey\)](#)
- [E.mapInPlace\(from, to, map, bits\)](#)
- [E.memoryArea\(addr, len\)](#)
- [E.memoryMap\(baseAddress, registers\)](#)
- [E.nativeCall\(addr, sig, data\)](#)
- [E.openFile\(path, mode\)](#)
- [E.pipe\(source, destination, options\)](#)
- [E.reboot\(\)](#)
- [E.reverseByte\(x\)](#)
- [E.sendUSBHID\(data\)](#)
- [E.setBootCode\(code, alwaysExec\)](#)
- [E.setClock\(options\)](#)
- [E.setConsole\(device, options\)](#)
- [E.setDST\(params, ...\)](#)
- [E.setFlags\(flags\)](#)
- [E.setPassword\(password\)](#)
- [E.setRTCPrescaler\(prescaler\)](#)
- [E.setTimeZone\(zone\)](#)
- [E.setUSBHID\(opts\)](#)
- [E.showAlert\(message, options\)](#)
- [E.showMenu\(menu\)](#)
- [E.showMessage\(message, options\)](#)
- [E.showPrompt\(message, options\)](#)
- [E.showScroller\(options\)](#)
- [E.strand\(v\)](#)
- [E.stopEventPropagation\(\)](#)
- [E.sum\(arr\)](#)
- [E.toArrayBuffer\(str\)](#)
- [E.toFlatString\(args, ...\)](#)
- [E.toJS\(arg\)](#)
- [E.toString\(args, ...\)](#)
- [event E.touch\(x, y, b\)](#)
- [E.toInt8Array\(args, ...\)](#)
- [E.unmountSD\(\)](#)

- [E.variance\(arr, mean\)](#)

[event E.AMS](#) ⇒

Call type:

[\(top\)](#)

```
E.on('AMS', function(info) { ... });
```

Parameters

info - An object (see below)

Description

Called when a media event arrives on an Apple iOS device Bangle.js is connected to

```
{
  id : "artist"/"album"/"title"/"duration",
  value : "Some text",
  truncated : bool // the 'value' was too big to be sent completely
}
```

Note: This is only available in Bangle.js smartwatches

[event E.ANCS](#) ⇒

Call type:

[\(top\)](#)

```
E.on('ANCS', function(info) { ... });
```

Parameters

info - An object (see below)

Description

Called when a notification arrives on an Apple iOS device Bangle.js is connected to

```
{
  event:"add",
  uid:42,
  category:4,
  categoryCnt:42,
  silent:true,
  important:false,
  preExisting:true,
  positive:false,
  negative:true
}
```

You can then get more information with [NRF.ancsGetNotificationInfo](#), for instance:

```
E.on('ANCS', event => {
  NRF.ancsGetNotificationInfo( event.uid ).then(a=>print("Notify",E.toJS(a)));
});
```

Note: This is only available in Bangle.js smartwatches

[E.asm](#) ⇒

Call type:

[\(top\)](#)

```
E.asm(callspec, assemblycode, ...)
```

Parameters

callspec - The arguments this assembly takes - e.g. `void(int)`

assemblycode, ... - One or more strings of assembler code

Description

Provide assembly to Espruino.

This function is not part of Espruino. Instead, it is detected by the Espruino IDE (or command-line tools) at upload time and is replaced with machine code and an [E.nativeCall](#) call.

See [the documentation on the Assembler](#) for more information.

Note: This is not available in devices with low flash memory

[E.clip](#) ⇒

Call type:

[\(top\)](#)

```
E.clip(x, min, max)
```

Parameters

x - A floating point value to clip

min - The smallest the value should be

max - The largest the value should be

Returns

The value of x, clipped so as not to be below min or above max.

Description

Clip a number to be between min and max (inclusive)

Note: This is not available in devices with low flash memory

[E.compiledC](#) ⇒

Call type:

[\(top\)](#)

```
E.compiledC(code)
```

Parameters

code - A Templated string of C code

Description

Provides the ability to write C code inside your JavaScript file.

This function is not part of Espruino. Instead, it is detected by the Espruino IDE (or command-line tools) at upload time, is sent to our web service to be compiled, and is replaced with machine code and an [E.nativeCall](#) call.

See [the documentation on Inline C](#) for more information and examples.

Note: This is not available in devices with low flash memory

[E.connectSDCard](#) ⇒

Call type:

[\(top\)](#)

```
E.connectSDCard(spi, csPin)
```

Parameters

spi - The SPI object to use for communication

csPin - The pin to use for Chip Select

Description

Setup the filesystem so that subsequent calls to `E.openFile` and `require('fs').*` will use an SD card on the supplied SPI device and pin.

It can even work using software SPI - for instance:

```
// DI/CMD = C7
// DO/DAT0 = C8
// CK/CLK = C9
// CD/CS/DAT3 = C6
var spi = new SPI();
spi.setup({mosi:C7, miso:C8, sck:C9});
E.connectSDCard(spi, C6);
console.log(require("fs").readdirSync());
```

See [the page on File IO](#) for more information.

Note: We'd strongly suggest you add a pullup resistor from CD/CS pin to 3.3v. It is good practise to avoid accidental writes before Espruino is initialised, and some cards will not work reliably without one.

Note: If you want to remove an SD card after you have started using it, you *must* call `E.unmountSD()`, or you may cause damage to the card.

Note: This is not available in devices with low flash memory

[E.convolve](#) ⇒

Call type:

[\(top\)](#)

```
E.convolve(arr1, arr2, offset)
```

Parameters

`arr1` - An array to convolve

`arr2` - An array to convolve

`offset` - The mean value of the array

Returns

The variance of the given buffer

Description

Convolve arr1 with arr2. This is equivalent to

```
v=0;for (i in arr1) v+=arr1[i] *  
arr2[(i+offset) % arr2.length]
```

Note: This is not available in devices with low flash memory

[E.CRC32](#) ⇒

Call type:

[\(top\)](#)

```
E.CRC32(data)
```

Parameters

`data` - Iterable data to perform CRC32 on (each element treated as a byte)

Returns

The CRC of the supplied data

Description

Perform a standard 32 bit CRC (Cyclic redundancy check) on the supplied data (one byte at a time) and return the result as an unsigned integer.

Note: This is not available in devices with low flash memory

[E.decodeUTF8](#) ⇒

Call type:[\(top\)](#)

```
E.decodeUTF8(str, lookup, replaceFn)
```

Parameters

`str` - A string of UTF8-encoded data

`lookup` - An array containing a mapping of character code -> replacement string

`replaceFn` - If not in `lookup`, `replaceFn(charCode)` is called and the result used if it's a function, *or* if it's a string, the string value is used

Returns

A string containing all UTF8 sequences flattened to 8 bits

Description

Decode a UTF8 string.

- Any decoded character less than 256 gets passed straight through
- Otherwise if `lookup` is an array and an item with that char code exists in `lookup` then that is used
- Otherwise if `lookup` is an object and an item with that char code (as lowercase hex) exists in `lookup` then that is used
- Otherwise `replaceFn(charCode)` is called and the result used if `replaceFn` is a function
- If `replaceFn` is a string, that is used
- Or finally if nothing else matches, the character is ignored

For instance:

```
let unicodeRemap = {
  0x20ac:"\u0080", // Euro symbol
  0x2026:"\u0085", // Ellipsis
};

E.decodeUTF8("UTF-8 Euro: \u00e2\u0082\u00ac", unicodeRemap, '[?]')
== "UTF-8 Euro: \u0080"
```

Note: This is not available in devices with low flash memory

[E.defrag](#) ⇒**Call type:**[\(top\)](#)

```
E.defrag()
```

Description

BETA: defragment memory!

Note: This is not available in devices with low flash memory

[E.dumpFragmentation](#) ⇒**Call type:**[\(top\)](#)

```
E.dumpFragmentation()
```

Description

Show fragmentation.

- is free space
- # is a normal variable
- L is a locked variable (address used, cannot be moved)
- = represents data in a Flat String (must be contiguous)

Note: This is not available in devices with low flash memory

[E.dumpFreeList](#) ⇒**Call type:**[\(top\)](#)

`E.dumpFreeList()`

Description

Dump any locked variables that aren't referenced from `global` - for debugging memory leaks only.

Note: This is not available in release builds

[E.dumpLockedVars](#) ⇒

Call type:

[\(top\)](#)

`E.dumpLockedVars()`

Description

Dump any locked variables that aren't referenced from `global` - for debugging memory leaks only.

Note: This does a linear scan over memory, finding variables that are currently locked. In some cases it may show variables like `Unknown` 66 which happen when *part* of a string has ended up placed in memory ahead of the String that it's part of. See <https://github.com/espruino/Espduino/issues/2345>

Note: This is not available in release builds

[E.dumpStr](#) ⇒

Call type:

[\(top\)](#)

`E.dumpStr()`

Returns

A String

Description

Get the current interpreter state in a text form such that it can be copied to a new device

Note: This is not available in devices with low flash memory

[E.dumpTimers](#) ⇒

Call type:

[\(top\)](#)

`E.dumpTimers()`

Description

Output the current list of Utility Timer Tasks - for debugging only

Note: This is not available in devices with low flash memory

[E.dumpVariables](#) ⇒

Call type:

[\(top\)](#)

`E.dumpVariables()`

Description

Dumps a comma-separated list of all allocated variables along with the variables they link to. Can be used to visualise where memory is used.

Note: This is not available in devices with low flash memory

[E.enableWatchdog](#) ⇒

Call type:

[\(top\)](#)

```
E.enableWatchdog(timeout, isAuto)
```

Parameters

`timeout` - The timeout in seconds before a watchdog reset

`isAuto` - If undefined or true, the watchdog is kicked automatically. If not, you must call [E.kickWatchdog\(\)](#) yourself

Description

Enable the watchdog timer. This will reset Espruino if it isn't able to return to the idle loop within the timeout.

If `isAuto` is false, you must call [E.kickWatchdog\(\)](#) yourself every so often or the chip will reset.

```
E.enableWatchdog(0.5); // automatic mode
while(1); // Espruino will reboot because it has not been idle for 0.5 sec

E.enableWatchdog(1, false);
setInterval(function() {
  if (everything_ok)
    E.kickWatchdog();
}, 500);
// Espruino will now reset if everything_ok is false,
// or if the interval fails to be called
```

NOTE: This is only implemented on STM32, nRF5x and ESP32 devices (all official Espruino boards).

NOTE: On STM32 (Pico, WiFi, Original) with `setDeepSleep(1)` you need to explicitly wake Espruino up with an interval of less than the watchdog timeout or the watchdog will fire and the board will reboot. You can do this with `setInterval("", time_in_milliseconds)`. **NOTE:** On ESP32, the timeout will be rounded to the nearest second.

Note: This is not available in devices with low flash memory

[event E.errorFlag](#) ⇒

Call type:

[\(top\)](#)

```
E.on('errorFlag', function(errorFlags) { ... });
```

Parameters

`errorFlags` - An array of new error flags, as would be returned by [E.getErrorFlags\(\)](#). Error flags that were present before won't be reported.

Description

This event is called when an error is created by Espruino itself (rather than JS code) which changes the state of the error flags reported by [E.getErrorFlags\(\)](#).

This could be low memory, full buffers, UART overflow, etc. [E.getErrorFlags\(\)](#) has a full description of each type of error.

This event will only be emitted when error flag is set. If the error flag was already set nothing will be emitted. To clear error flags so that you do get a callback each time a flag is set, call [E.getErrorFlags\(\)](#).

[E.FFT](#) ⇒

Call type:

[\(top\)](#)

```
E.FFT(arrReal, arrImage, inverse)
```

Parameters

`arrReal` - An array of real values

`arrImage` - An array of imaginary values (or if undefined, all values will be taken to be 0)

`inverse` - Set this to true if you want an inverse FFT - otherwise leave as 0

Description

Performs a Fast Fourier Transform (FFT) in 32 bit floats on the supplied data and writes it back into the original arrays. Note that if only one array is supplied, the data written back is the modulus of the complex result `sqrt(r*r+i*i)`.

In order to perform the FFT, there has to be enough room on the stack to allocate two arrays of 32 bit floating point numbers - this will limit the maximum size of FFT possible to around 1024 items on most platforms.

Note: on the Original Espruino board, FFTs are performed in 64bit arithmetic as there isn't space to include the 32 bit maths routines (2x more RAM is required).

Note: This is not available in devices with low flash memory

[E.flashFatFS](#) ⇒

Call type:

[\(top\)](#)

```
E.flashFatFS(options)
```

Parameters

options - [optional] An object { `addr : int=0x300000`, `sectors : int=256`, `format : bool=false` }
addr: start address in flash
sectors: number of sectors to use
format: Format the media

Returns

True on success, or false on failure

Description

Change the parameters used for the flash filesystem. The default address is the last 1Mb of 4Mb Flash, 0x300000, with total size of 1Mb.

Before first use the media needs to be formatted.

```
fs=require("fs");
try {
  fs.readdirSync();
} catch (e) { //Uncaught Error: Unable to mount media : NO_FILESYSTEM
  console.log('Formatting FS - only need to do once');
  E.flashFatFS({ format: true });
}
fs.writeFileSync("bang.txt", "This is the way the world ends\nnot with a bang but a whimper.\n");
fs.readdirSync();
```

This will create a drive of 100 * 4096 bytes at 0x300000. Be careful with the selection of flash addresses as you can overwrite firmware! You only need to format once, as each will erase the content.

```
E.flashFatFS({ addr:0x300000,sectors:100,format:true });
```

Note: This is only available in devices with filesystem in Flash support enabled (ESP32 only)

[E.getAddressOf](#) ⇒

Call type:

[\(top\)](#)

```
E.getAddressOf(v, flatAddress)
```

Parameters

v - A variable to get the address of

flatAddress - (boolean) If `true` and a Flat String or Flat ArrayBuffer is supplied, return the address of the data inside it - otherwise 0. If `false` (the default) return the address of the JsVar itself.

Returns

The address of the given variable

Description

Return the address in memory of the given variable. This can then be used with `peek` and `poke` functions. However, changing data in JS variables directly (`flatAddress=false`) will most likely result in a crash.

This function exists to allow embedded targets to set up peripherals such as DMA so that they write directly to JS variables.

See <http://www.espruino.com/Internals> for more information

Note: This is not available in devices with low flash memory

[E.getAnalogVRef](#) ⇒

Call type:

[\(top\)](#)

```
E.getAnalogVRef()
```

Returns

The voltage (in Volts) that a reading of 1 from [analogRead](#) actually represents - usually around 3.3v

Description

Check the internal voltage reference. To work out an actual voltage of an input pin, you can use `analogRead(pin)*E.getAnalogVRef()`

Note: This value is calculated by reading the voltage on an internal voltage reference with the ADC. It will be slightly noisy, so if you need this for accurate measurements we'd recommend that you call this function several times and average the results.

While this is implemented on Espruino boards, it may not be implemented on other devices. If so it'll return NaN.

Note: This is not available in devices with low flash memory

[E.getBattery](#) ⇒

Call type:

[\(top\)](#)

```
E.getBattery()
```

Returns

A percentage between 0 and 100

Description

In devices that come with batteries, this function returns the battery charge percentage as an integer between 0 and 100.

Note: this is an estimation only, based on battery voltage. The temperature of the battery (as well as the load being drawn from it at the time [E.getBattery](#) is called) will affect the readings.

Note: This is only available in Puck.js devices and Pixl.js boards and Bangle.js smartwatches

[E.getConsole](#) ⇒

Call type:

[\(top\)](#)

```
E.getConsole()
```

Returns

The current console device as a string, or just `null` if the console is null

Description

Returns the current console device - see [E.setConsole](#) for more information.

[E.getErrorFlags](#) ⇒

Call type:

[\(top\)](#)

```
E.getErrorFlags()
```

Returns

An array of error flags

Description

Get and reset the error flags. Returns an array that can contain:

'**FIFO_FULL**': The receive FIFO filled up and data was lost. This could be state transitions for setWatch, or received characters.

'**BUFFER_FULL**': A buffer for a stream filled up and characters were lost. This can happen to any stream - Serial,HTTP,etc.

'**CALLBACK**': A callback (`setWatch, setInterval, on('data', ...)`) caused an error and so was removed.

'**LOW_MEMORY**': Memory is running low - Espruino had to run a garbage collection pass or remove some of the command history

'**MEMORY**': Espruino ran out of memory and was unable to allocate some data that it needed.

'**UART_OVERFLOW**': A UART received data but it was not read in time and was lost

Note: This is not available in devices with low flash memory

[E.getFlags](#) ⇒

Call type:

[\(top\)](#)

```
E.getFlags()
```

Returns

An object containing flag names and their values

Description

Get Espruino's interpreter flags that control the way it handles your JavaScript code.

- `deepSleep` - Allow deep sleep modes (also set by `setDeepSleep`)
- `pretokenise` - When adding functions, pre-minify them and tokenise reserved words
- `unsafeFlash` - Some platforms stop writes/erases to interpreter memory to stop you bricking the device accidentally - this removes that protection
- `unsyncFiles` - When writing files, *don't* flush all data to the SD card after each command (the default is *to* flush). This is much faster, but can cause filesystem damage if power is lost without the filesystem unmounted.

[E.getRTCPrescaler](#) ⇒

Call type:

[\(top\)](#)

```
E.getRTCPrescaler(calibrate)
```

Parameters

`calibrate` - If `false`, the current value. If `true`, the calculated 'correct' value

Returns

The RTC prescaler's current value

Description

Gets the RTC's current prescaler value if `calibrate` is undefined or false.

If `calibrate` is true, the low speed oscillator's speed is calibrated against the high speed oscillator (usually +/- 20 ppm) and a suggested value to be fed into `E.setRTCPrescaler(...)` is returned.

See [E.setRTCPrescaler](#) for more information.

Note: This is only available in Espruino Pico boards and Espruino WiFi boards and 'Original' Espruino boards

[E.getSizeOf](#) ⇒

Call type:

[\(top\)](#)

```
E.getSizeOf(v, depth)
```

Parameters

v - A variable to get the size of

depth - The depth that detail should be provided for. If depth<=0 or undefined, a single integer will be returned

Returns

Information about the variable size - see below

Description

Return the number of variable blocks used by the supplied variable. This is useful if you're running out of memory and you want to be able to see what is taking up most of the available space.

If depth>0 and the variable can be recursed into, an array listing all property names (including internal Espruino names) and their sizes is returned. If depth>1 there is also a more field that inspects the objects' children's children.

For instance `E.getSizeOf(function(a,b) { })` returns 5.

But `E.getSizeOf(function(a,b) { }, 1)` returns:

```
[  
  {  
    "name": "a",  
    "size": 1 },  
  {  
    "name": "b",  
    "size": 1 },  
  {  
    "name": "\xFFcod",  
    "size": 2 }  
]
```

In this case setting depth to 2 will make no difference as there are no more children to traverse.

See <http://www.espruino.com/Internals> for more information

Note: This is not available in devices with low flash memory

[E.getTemperature](#) ⇒

Call type:

[\(top\)](#)

```
E.getTemperature()
```

Returns

The temperature in degrees C

Description

Use the microcontroller's internal thermistor to work out the temperature.

On Puck.js v2.0 this will use the on-board PCT2075TP temperature sensor, but on other devices it may not be desperately well calibrated.

While this is implemented on Espruino boards, it may not be implemented on other devices. If so it'll return NaN.

Note: This is not entirely accurate and varies by a few degrees from chip to chip. It measures the **die temperature**, so when connected to USB it could be reading 10 over degrees C above ambient temperature. When running from battery with `setDeepSleep(true)` it is much more accurate though.

[E.HSBtoRGB](#) ⇒

Call type:

[\(top\)](#)

```
E.HSBtoRGB(hue, sat, bri, format)
```

Parameters

`hue` - The hue, as a value between 0 and 1
`sat` - The saturation, as a value between 0 and 1
`bri` - The brightness, as a value between 0 and 1
`format` - If `true` or 1, return an array of [R,G,B] values between 0 and 255. If 16, return a 16 bit number. `undefined/24` is the same as normal (returning a 24 bit number)

Returns

A 24 bit number containing bytes representing red, green, and blue `0xBBGGRR`. Or if `asArray` is true, an array [R, G, B]

Description

Convert hue, saturation and brightness to red, green and blue (packed into an integer if `asArray==false` or an array if `asArray==true`).

This replaces `Graphics.setColorHSB` and `Graphics.setBgColorHSB`. On devices with 24 bit colour it can be used as: `Graphics.setColor(E.HSBtoRGB(h, s, b))`, or on devices with 26 bit colour use `Graphics.setColor(E.HSBtoRGB(h, s, b, 16))`

You can quickly set RGB items in an Array or Typed Array using `array.set(E.HSBtoRGB(h, s, b, true), offset)`, which can be useful with arrays used with `require("neopixel").write`.

Note: This is not available in devices with low flash memory

[E.hwRand](#) ⇒

Call type:

[\(top\)](#)

```
E.hwRand()
```

Returns

A random number

Description

Unlike 'Math.random()' which uses a pseudo-random number generator, this method reads from the internal voltage reference several times, XOR-ing and rotating to try and make a relatively random value from the noise in the signal.

Note: This is not available in devices with low flash memory

[event E.init](#) ⇒

Call type:

[\(top\)](#)

```
E.on('init', function() { ... });
```

Description

This event is called right after the board starts up, and has a similar effect to creating a function called `onInit`.

For example to write "Hello World" every time Espruino starts, use:

```
E.on('init', function() {
  console.log("Hello World!");
});
```

Note: that subsequent calls to `E.on('init')`, will add a new handler, rather than replacing the last one. This allows you to write modular code - something that was not possible with `onInit`.

[E.kickWatchdog](#) ⇒

Call type:

[\(top\)](#)

```
E.kickWatchdog()
```

Description

Kicks a Watchdog timer set up with `E.enableWatchdog(..., false)`. See [E.enableWatchdog](#) for more information.

NOTE: This is only implemented on STM32 and nRF5x devices (all official Espruino boards).

Note: This is not available in devices with low flash memory

[event E.kill](#) ⇒

Call type:

[\(top\)](#)

```
E.on('kill', function() { ... });
```

Description

This event is called just before the device shuts down for commands such as `reset()`, `load()`, `save()`, `E.reboot()`, or `Bangle.off()`.

For example to write "Bye!" just before shutting down use:

```
E.on('kill', function() {
  console.log("Bye!");
});
```

NOTE: This event is not called when the device is 'hard reset' - for example by removing power, hitting an actual reset button, or via a Watchdog timer reset.

[E.lockConsole](#) ⇒

Call type:

[\(top\)](#)

```
E.lockConsole()
```

Description

If a password has been set with `E.setPassword()`, this will lock the console so the password needs to be entered to unlock it.

[E.lookupNoCase](#) ⇒

Call type:

[\(top\)](#)

```
E.lookupNoCase(haystack, needle, returnKey)
```

Parameters

`haystack` - The Array/Object/Function to search

`needle` - The key to search for

`returnKey` - If true, return the key, else return the value itself

Returns

The value in the Object matching 'needle', or if `returnKey==true` the key's name - or undefined

Description

Search in an Object, Array, or Function

[E.mapInPlace](#) ⇒

Call type:

[\(top\)](#)

```
E.mapInPlace(from, to, map, bits)
```

Parameters

`from` - An ArrayBuffer to read elements from

`to` - An ArrayBuffer to write elements too

`map` - An array or `function(value, index)` to use to map one element to another, or `undefined` to provide no mapping

`bits` - If specified, the number of bits per element (MSB first) - otherwise use a 1:1 mapping. If negative, use LSB first.

Description

Take each element of the `from` array, look it up in `map` (or call `map(value, index)` if it is a function), and write it into the corresponding element in the `to` array.

You can use an array to map:

```
var a = new Uint8Array([1,2,3,1,2,3]);
var lut = new Uint8Array([128,129,130,131]);
E.mapInPlace(a, a, lut);
// a = [129, 130, 131, 129, 130, 131]
```

Or `undefined` to pass straight through, or a function to do a normal 'mapping':

```
var a = new Uint8Array([0x12,0x34,0x56,0x78]);
var b = new Uint8Array(8);
E.mapInPlace(a, b, undefined); // straight through
// b = [0x12,0x34,0x56,0x78,0,0,0,0]
E.mapInPlace(a, b, (value,index)=>index); // write the index in the first 4 (because a.length==4)
// b = [0,1,2,3,4,0,0,0]
E.mapInPlace(a, b, undefined, 4); // 4 bits from 8 bit input -> 2x as many outputs, msb-first
// b = [1, 2, 3, 4, 5, 6, 7, 8]
E.mapInPlace(a, b, undefined, -4); // 4 bits from 8 bit input -> 2x as many outputs, lsb-first
// b = [2, 1, 4, 3, 6, 5, 8, 7]
E.mapInPlace(a, b, a=>a+2, 4);
// b = [3, 4, 5, 6, 7, 8, 9, 10]
var b = new Uint16Array(4);
E.mapInPlace(a, b, undefined, 12); // 12 bits from 8 bit input, msb-first
// b = [0x123, 0x456, 0x780, 0]
E.mapInPlace(a, b, undefined, -12); // 12 bits from 8 bit input, lsb-first
// b = [0x412, 0x563, 0x078, 0]
```

Note: This is not available in devices with low flash memory

E.memoryArea ⇒

Call type:

[\(top\)](#)

```
E.memoryArea(addr, len)
```

Parameters

`addr` - The address of the memory area

`len` - The length (in bytes) of the memory area

Returns

A String

Description

This creates and returns a special type of string, which actually references a specific memory address. It can be used in order to use sections of Flash memory directly in Espruino (for example to execute code straight from flash memory with `eval(E.memoryArea(...))`)

Note: This is only tested on STM32-based platforms (Espruino Original and Espruino Pico) at the moment.

E.memoryMap ⇒

Call type:

[\(top\)](#)

```
E.memoryMap(baseAddress, registers)
```

Parameters

`baseAddress` - The base address (added to every address in `registers`)

`registers` - An object containing {name:address}

Returns

An object where each field is memory-mapped to a register.

Description

Create an object where every field accesses a specific 32 bit address in the microcontroller's memory. This is perfect for accessing on-chip peripherals.

```
// for NRF52 based chips
var GPIO = E.memoryMap(0x50000000,{OUT:0x504, OUTSET:0x508, OUTCLR:0x50C, IN:0x510, DIR:0x514, DIRSET:0x518, DIRCLR:0x51C});
GPIO.DIRSET = 1; // set GPIO0 to output
GPIO.OUT ^= 1; // toggle the output state of GPIO0
```

Note: This is not available in devices with low flash memory

E.nativeCall ⇒

Call type:

[\(top\)](#)

```
E.nativeCall(addr, sig, data)
```

Parameters

addr - The address in memory of the function (or offset in data if it was supplied)

sig - The signature of the call, returnType (arg1,arg2,...). Allowed types are `void, bool, int, double, Pin, JsVar`

data - (Optional) A string containing the function itself. If not supplied then 'addr' is used as an absolute address.

Returns

The native function

Description

ADVANCED: This is a great way to crash Espruino if you're not sure what you are doing

Create a native function that executes the code at the given address, e.g. `E.nativeCall(0x08012345, 'double (double,double)')(1.1, 2.2)`

If you're executing a thumb function, you'll almost certainly need to set the bottom bit of the address to 1.

Note it's not guaranteed that the call signature you provide can be used - there are limits on the number of arguments allowed.

When supplying data, if it is a 'flat string' then it will be used directly, otherwise it'll be converted to a flat string and used.

Note: This is not available in devices with low flash memory

E.openFile ⇒

Call type:

[\(top\)](#)

```
E.openFile(path, mode)
```

Parameters

path - the path to the file to open.

mode - The mode to use when opening the file. Valid values for mode are 'r' for read, 'w' for write new, 'w+' for write existing, and 'a' for append. If not specified, the default is 'r'.

Returns

A File object

Description

Open a file

E.pipe ⇒

Call type:[\(top\)](#)

```
E.pipe(source, destination, options)
```

Parameters

source - The source file/stream that will send content.

destination - The destination file/stream that will receive content from the source.

options - [optional] An object { `chunkSize : int=64`, `end : bool=true`, `complete : function` }

chunkSize : The amount of data to pipe from source to destination at a time

complete : a function to call when the pipe activity is complete

end : call the 'end' function on the destination when the source is finished

E.reboot ⇒**Call type:**[\(top\)](#)

```
E.reboot()
```

Description

Forces a hard reboot of the microcontroller - as close as possible to if the reset pin had been toggled.

Note: This is different to [reset\(\)](#), which performs a software reset of Espruino (resetting the interpreter and pin states, but not all the hardware)

E.reverseByte ⇒**Call type:**[\(top\)](#)

```
E.reverseByte(x)
```

Parameters

x - A byte value to reverse the bits of

Returns

The byte with reversed bits

Description

Reverse the 8 bits in a byte, swapping MSB and LSB.

For example, `E.reverseByte(0b10010000) == 0b00001001`.

Note that you can reverse all the bytes in an array with:

```
arr =  
arr.map(E.reverseByte)
```

Note: This is not available in devices with low flash memory

E.sendUSBHID ⇒**Call type:**[\(top\)](#)

```
E.sendUSBHID(data)
```

Parameters

data - An array of bytes to send as a USB HID packet

Returns

1 on success, 0 on failure

[E.setBootCode](#) ⇒

Call type:

[\(top\)](#)

```
E.setBootCode(code, alwaysExec)
```

Parameters

code - The code to execute (as a string)

alwaysExec - Whether to always execute the code (even after a reset)

Description

This writes JavaScript code into Espruino's flash memory, to be executed on startup. It differs from [save\(\)](#) in that [save\(\)](#) saves the whole state of the interpreter, whereas this just saves JS code that is executed at boot.

Code will be executed before [onInit\(\)](#) and [E.on\('init', ...\)](#).

If `alwaysExec` is `true`, the code will be executed even after a call to [reset\(\)](#). This is useful if you're making something that you want to program, but you want some code that is always built in (for instance setting up a display or keyboard).

To remove boot code that has been saved previously, use [E.setBootCode\(""\)](#)

Note: this removes any code that was previously saved with [save\(\)](#).

[E.setClock](#) ⇒

Call type:

[\(top\)](#)

```
E.setClock(options)
```

Parameters

options - Platform-specific options for setting clock speed

Returns

The actual frequency the clock has been set to

Description

This sets the clock frequency of Espruino's processor. It will return `0` if it is unimplemented or the clock speed cannot be changed.

Note: On pretty much all boards, UART, SPI, I2C, PWM, etc will change frequency and will need setting up again in order to work.

STM32F4

Options is of the form `{ M: int, N: int, P: int, Q: int }` - see the 'Clocks' section of the microcontroller's reference manual for what these mean.

- System clock = $8\text{Mhz} * N / (M * P)$
- USB clock (should be 48Mhz) = $8\text{Mhz} * N / (M * Q)$

Optional arguments are:

- `latency` - flash latency from 0..15
- `PCLK1` - Peripheral clock 1 divisor (default: 2)
- `PCLK2` - Peripheral clock 2 divisor (default: 4)

The Pico's default is `{M:8, N:336, P:4, Q:7, PCLK1:2, PCLK2:4}`, use

```
{M:8,  
N:336,  
P:8,  
Q:7,  
PCLK1:1,  
PCLK2:2}
```

to halve the system clock speed while keeping the peripherals running at the same speed (omitting PCLK1/2 will lead to the peripherals changing speed too).

On STM32F4 boards (e.g. Espruino Pico), the USB clock needs to be kept at 48Mhz or USB will fail to work. You'll also experience USB instability if the processor clock falls much below 48Mhz.

ESP8266

Just specify an integer value, either 80 or 160 (for 80 or 160Mhz)

Note: This is not available in devices with low flash memory

[E.setConsole](#) =>

Call type:

[\(top\)](#)

```
E.setConsole(device, options)
```

Parameters

device -

options - [optional] object of options, see below

Description

Changes the device that the JS console (otherwise known as the REPL) is attached to. If the console is on a device, that device can be used for programming Espruino.

Rather than calling [Serial.setConsole](#) you can call `E.setConsole("DeviceName")`.

This is particularly useful if you just want to remove the console. `E.setConsole(null)` will make the console completely inaccessible.

device may be `"Serial1","USB","Bluetooth","Telnet","Terminal"`, any other hardware [Serial](#) device, or `null` to disable the console completely.

options is of the form:

```
{
  force : bool // default false, force the console onto this device so it does not move
              // if false, changes in connection state (e.g. USB/Bluetooth) can move
              // the console automatically.
}
```

Note: This is not available in devices with low flash memory

[E.setDST](#) =>

Call type:

[\(top\)](#)

```
E.setDST(params, ...)
```

Parameters

params, ... - An array containing the settings for DST

Description

Set the daylight savings time parameters to be used with [Date](#) objects.

The parameters are - dstOffset: The number of minutes daylight savings time adds to the clock (usually 60) - set to 0 to disable DST - timezone: The time zone, in minutes, when DST is not in effect - positive east of Greenwich - startDowNumber: The index of the day-of-week in the month when DST starts - 0 for first, 1 for second, 2 for third, 3 for fourth and 4 for last - startDow: The day-of-week for the DST start calculation - 0 for Sunday, 6 for Saturday - startMonth: The number of the month that DST starts - 0 for January, 11 for December - startDayOffset: The number of days between the selected day-of-week and the actual day that DST starts - usually 0 - startTimeOfDay: The number of minutes elapsed in the day before DST starts - endDowNumber: The index of the day-of-week in the month when DST ends - 0 for first, 1 for second, 2 for third, 3 for fourth and 4 for last - endDow: The day-of-week for the DST end calculation - 0 for Sunday, 6 for Saturday - endMonth: The number of the month that DST ends - 0 for January, 11 for December - endDayOffset: The number of days between the selected day-of-week and the actual day that DST ends - usually 0 - endTimeOfDay: The number of minutes elapsed in the day before DST ends

To determine what the dowNumber, dow, month, dayOffset, timeOfDay parameters should be, start with a sentence of the form "DST starts on the last Sunday of March (plus 0 days) at 03:00". Since it's the last Sunday, we have startDowNumber = 4, and since it's Sunday, we have startDow = 0. That it is March gives us startMonth = 2, and that the offset is zero days, we have startDayOffset = 0. The time that DST starts gives us startTimeOfDay = 3*60.

"DST ends on the Friday before the second Sunday in November at 02:00" would give us endDowNumber=1, endDow=0, endMonth=10, endDayOffset=-2 and endTimeOfDay=120.

Using Ukraine as an example, we have a time which is 2 hours ahead of GMT in winter (EET) and 3 hours in summer (EEST). DST starts at 03:00 EET on the last Sunday in March, and ends at 04:00 EEST on the last Sunday in October. So someone in Ukraine might call `E.setDST(60,120,4,0,2,0,180,4,0,9,0,240)`;

Note that when DST parameters are set (i.e. when dstOffset is not zero), [E.setTimezone\(\)](#) has no effect.

Note: This is not available in ESPR_NO_DAYLIGHT_SAVING

[E.setFlags](#) =>

Call type:

[\(top\)](#)

```
E.setFlags(flags)
```

Parameters

flags - An object containing flag names and boolean values. You need only specify the flags that you want to change.

Description

Set the Espruino interpreter flags that control the way it handles your JavaScript code.

Run [E.getFlags\(\)](#) and check its description for a list of available flags and their values.

[E.setPassword](#) ⇒

Call type:

[\(top\)](#)

```
E.setPassword(password)
```

Parameters

password - The password - max 20 chars

Description

Set a password on the console (REPL). When powered on, Espruino will then demand a password before the console can be used. If you want to lock the console immediately after this you can call [E.lockConsole\(\)](#).

To remove the password, call this function with no arguments.

Note: There is no protection against multiple password attempts, so someone could conceivably try every password in a dictionary.

Note: This password is stored in memory in plain text. If someone is able to execute arbitrary JavaScript code on the device (e.g., you use [eval](#) on input from unknown sources) or read the device's firmware then they may be able to obtain it.

[E.setRTCPrescaler](#) ⇒

Call type:

[\(top\)](#)

```
E.setRTCPrescaler(prescaler)
```

Parameters

prescaler - The amount of counts for one second of the RTC - this is a 15 bit integer value (0..32767)

Description

Sets the RTC's prescaler's maximum value. This is the counter that counts up on each oscillation of the low speed oscillator. When the prescaler counts to the value supplied, one second is deemed to have passed.

By default this is set to the oscillator's average speed as specified in the datasheet, and usually that is fine. However on early [Espruino Pico](#) boards the STM32F4's internal oscillator could vary by as much as 15% from the value in the datasheet. In that case you may want to alter this value to reflect the true RTC speed for more accurate timekeeping.

To change the RTC's prescaler value to a computed value based on comparing against the high speed oscillator, just run the following command, making sure it's done a few seconds after the board starts up:

```
E.setRTCPrescaler(E.getRTCPrescaler(true));
```

When changing the RTC prescaler, the RTC 'follower' counters are reset and it can take a second or two before readings from [getTime](#) are stable again.

To test, you can connect an input pin to a known frequency square wave and then use [setwatch](#). If you don't have a frequency source handy, you can check against the high speed oscillator:

```
// connect pin B3 to B4
analogWrite(B3, 0.5, {freq:0.5});
setWatch(function(e) {
  print(e.time - e.lastTime);
}, B4, {repeat:true});
```

Note: This is only used on official Espruino boards containing an STM32 microcontroller. Other boards (even those using an STM32) don't use the RTC and so this has no effect.

Note: This is only available in Espruino Pico boards and Espruino WiFi boards and 'Original' Espruino boards

E.setTimeZone ⇒

Call type:

[\(top\)](#)

```
E.setTimeZone(zone)
```

Parameters

zone - The time zone in hours

Description

Set the time zone to be used with [date](#) objects.

For example `E.setTimeZone(1)` will be GMT+0100

Note that `E.setTimeZone()` will have no effect when daylight savings time rules have been set with [E.setDST\(\)](#). The timezone value will be stored, but never used so long as DST settings are in effect.

Time can be set with [setTime](#).

E.setUSBHID ⇒

Call type:

[\(top\)](#)

```
E.setUSBHID(opts)
```

Parameters

opts - An object containing at least reportDescriptor, an array representing the report descriptor. Pass undefined to disable HID.

Description

USB HID will only take effect next time you unplug and re-plug your Espruino. If you're disconnecting it from power you'll have to make sure you have [save\(\)](#)d after calling this function.

Note: This is only available in devices that support USB HID (Espruino Pico and Espruino WiFi)

E.showAlert ⇒

Call type:

[\(top\)](#)

```
E.showAlert(message, options)
```

Parameters

message - A message to display. Can include newlines

options - [optional] a title for the message or an object containing options

Returns

A promise that is resolved when 'Ok' is pressed

Description

Displays a full screen prompt on the screen, with a single 'Ok' button.

When the button is pressed the promise is resolved.

```
E.showAlert("Hello").then(function() {
  print("Ok pressed");
});
// or
E.showAlert("These are\nLots of\nnLines","My Title").then(function() {
```

```
    print("Ok pressed");
});
```

To remove the window, call [E.showAlert\(\)](#) with no arguments.

Note: This is only available in Bangle.js smartwatches

[E.showMenu](#) ⇒

Call type:

```
E.showMenu(menu)
```

Parameters

menu - An object containing name->function mappings to be used in a menu

Returns

A menu object with draw, move and **select** functions

Description

Display a menu on the screen, and set up the buttons to navigate through it.

Supply an object containing menu items. When an item is selected, the function it references will be executed. For example:

```
var boolean = false;
var number = 50;
// First menu
var mainmenu = {
  "" : { title: " -- Main Menu -- " }, // options
  "LED On" : function() { LED1.set(); },
  "LED Off" : function() { LED1.reset(); },
  "Submenu" : function() { E.showMenu(submenu); },
  "A Boolean" : {
    value : boolean,
    format : v => v?"On":"Off",
    onchange : v => { boolean=v; }
  },
  "A Number" : {
    value : number,
    min:0,max:100,step:10,
    onchange : v => { number=v; }
  },
  "Exit" : function() { E.showMenu(); }, // remove the menu
};
// Submenu
var submenu = {
  "" : { title : " -- SubMenu -- ",
         back : function() { E.showMenu(mainmenu); } },
  "One" : undefined, // do nothing
  "Two" : undefined // do nothing
};
// Actually display the menu
E.showMenu(mainmenu);
```

The menu will stay onscreen and active until explicitly removed, which you can do by calling [E.showMenu\(\)](#) without arguments.

See http://www.espruino.com/graphical_menu for more detailed information.

On Bangle.js there are a few additions over the standard `graphical_menu`:

- The options object can contain:
 - `back : function() {}` - add a 'back' button, with the function called when it is pressed
 - `remove : function() {}` - add a handler function to be called when the menu is removed
 - (Bangle.js 2) `scroll : int` - an integer specifying how much the initial menu should be scrolled by
- The object returned by [E.showMenu](#) contains:
 - (Bangle.js 2) `scroller` - the object returned by [E.showScroller](#) - `scroller.scroll` returns the amount the menu is currently scrolled by
- In the object specified for editable numbers:
 - (Bangle.js 2) the `format` function is called with `format(value)` in the main menu, `format(value,1)` when in a scrollable list, or `format(value,2)` when in a popup window.

You can also specify menu items as an array (rather than an Object). This can be useful if you have menu items with the same title, or you want to push menu items onto an array:

```
var menu = [
  { title:"Something", onchange:function() { print("selected"); } },
  { title:"On or Off", value:false, onchange: v => print(v) },
  { title:"A Value", value:3, min:0, max:10, onchange: v => print(v) },
];
menu[""] = { title:"Hello" };
E.showMenu(menu);
```

[\(top\)](#)

Note: This is only available in Bangle.js smartwatches

Note: This is only available in Bangle.js smartwatches with DTNO1_F5

E.showMessage ⇒

Call type:

[\(top\)](#)

```
E.showMessage(message, options)
```

Parameters

`message` - A message to display. Can include newlines

`options` - [optional] a title for the message, or an object of options {`title:string`, `img:image_string`}

Description

A utility function for displaying a full screen message on the screen.

Draws to the screen and returns immediately.

```
E.showMessage("These are\nLots of\nLines","My Title")
```

or to display an image as well as text:

```
E.showMessage("Lots of text will wrap automatically",{
  title:"Warning",
  img:atob("FBQBAfgAf+Af/4P//D+fx/n+f5/v+f//n//5//f//n///3//5/n+P//D//wf/4B/4AH4A=")
})
```

Note: This is only available in Bangle.js smartwatches

E.showPrompt ⇒

Call type:

[\(top\)](#)

```
E.showPrompt(message, options)
```

Parameters

`message` - A message to display. Can include newlines

`options` - [optional] an object of options (see below)

Returns

A promise that is resolved when 'Ok' is pressed

Description

Displays a full screen prompt on the screen, with the buttons requested (or `yes` and `no` for defaults).

When the button is pressed the promise is resolved with the requested values (for the `yes` and `no` defaults, `true` and `false` are returned).

```
E.showPrompt("Do you like fish?").then(function(v) {
  if (v) print("Yes' chosen");
  else print("No' chosen");
}); // Or
E.showPrompt("How many fish\ndo you like?", {
  title:"Fish",
  buttons : {"One":1,"Two":2,"Three":3}
}).then(function(v) {
  print("You like "+v+" fish");
}); // Or
E.showPrompt("Continue?", {
  title:"Alert",
  img:atob("FBQBAfgAf+Af/4P//D+fx/n+f5/v+f//n//5//f//n///3//5/n+P//D//wf/4B/4AH4A=")).then(function(v) {
  if (v) print("Yes' chosen");
  else print("No' chosen");
});
```

To remove the prompt, call `E.showPrompt()` with no arguments.

The second options argument can contain:

```
{  
  title: "Hello", // optional Title  
  buttons : {"Ok":true,"Cancel":false}, // optional list of button text & return value  
  img: "image_string" // optional image string to draw  
  remove: function() {} // Bangle.js: optional function to be called when the prompt is removed  
}
```

Note: This is only available in Bangle.js smartwatches

Note: This is only available in Bangle.js smartwatches with Bangle.js 2 smartwatches

E.showScroller ⇒

Call type:

[\(top\)](#)

```
E.showScroller(options)
```

Parameters

options - An object containing { h, c, draw, **select**, back, remove } (see below)

Returns

A menu object with `draw()` and `drawItem(itemNo)` functions

Description

Display a scrollable menu on the screen, and set up the buttons/touchscreen to navigate through it and select items.

Supply an object containing:

```
{  
  h : 24, // height of each menu item in pixels  
  c : 10, // number of menu items  
  // a function to draw a menu item  
  draw : function(idx, rect) { ... }  
  // a function to call when the item is selected, touch parameter is only relevant  
  // for Bangle.js 2 and contains the coordinates touched inside the selected item  
  select : function(idx, touch) { ... }  
  // optional function to be called when 'back' is tapped  
  back : function() { ... }  
  // Bangle.js: optional function to be called when the scroller should be removed  
  remove : function() {}  
}
```

For example to display a list of numbers:

```
E.showScroller({  
  h : 40, c : 8,  
  draw : (idx, r) => {  
    g.setBgColor((idx&1)? "#666": "#999").clearRect(r.x,r.y,r.x+r.w-1,r.y+r.h-1);  
    g.setFont("6x8:2").drawString("Item Number\n"+idx,r.x+10,r.y+4);  
  },  
  select : (idx) => console.log("You selected ", idx)  
});
```

To remove the scroller, just call `E.showScroller()`.

Note: This is only available in Bangle.js smartwatches

Note: This is only available in Bangle.js smartwatches with Bangle.js 2 smartwatches

E.strand ⇒

Call type:

[\(top\)](#)

```
E.strand(v)
```

Parameters

v - The 32 bit integer seed to use for the random number generator

Description

Set the seed for the random number generator used by `Math.random()`.

Note: This is not available in devices with low flash memory

[E.stopEventPropagation](#) ⇒

Call type:

[\(top\)](#)

```
E.stopEventPropagation()
```

Description

When using events with `x.on('foo', function() { ... })` and then `x.emit('foo')` you might want to stop subsequent event handlers from being executed.

Calling this function during the execution of events will ensure that no subsequent event handlers are executed.

```
var X = {};  
// in Espruino all objects are EventEmitters  
X.on('foo', function() { print("A"); })  
X.on('foo', function() { print("B"); E.stopEventPropagation(); })  
X.on('foo', function() { print("C"); })  
X.emit('foo');  
// prints A,B but not C
```

[E.sum](#) ⇒

Call type:

[\(top\)](#)

```
E.sum(arr)
```

Parameters

`arr` - The array to sum

Returns

The sum of the given buffer

Description

Sum the contents of the given Array, String or ArrayBuffer and return the result

Note: This is not available in devices with low flash memory

[E.toArrayBuffer](#) ⇒

Call type:

[\(top\)](#)

```
E.toArrayBuffer(str)
```

Parameters

`str` - The string to convert to an ArrayBuffer

Returns

An ArrayBuffer that uses the given string

Description

Create an ArrayBuffer from the given string. This is done via a reference, not a copy - so it is very fast and memory efficient.

Note that this is an ArrayBuffer, not a Uint8Array. To get one of those, do: `new Uint8Array(E.toArrayBuffer('....'))`.

[E.toFlatString](#) ⇒

Call type:[\(top\)](#)

```
E.toFlatString(args, ...)
```

Parameters

args, ... - The arguments to convert to a Flat String

Returns

A Flat String (or undefined)

Description

Returns a Flat [String](#) representing the data in the arguments, or [undefined](#) if one can't be allocated.

This provides the same behaviour that [E.toString](#) had in Espruino before 2v18 - see [E.toString](#) for more information.

[E.toJS](#) =>**Call type:**[\(top\)](#)

```
E.toJS(arg)
```

Parameters

arg - The JS variable to convert to a string

Returns

A String

Description

This performs the same basic function as [JSON.stringify](#), however [JSON.stringify](#) adds extra characters to conform to the JSON spec which aren't required if outputting JS.

[E.toJS](#) will also stringify JS functions, whereas [JSON.stringify](#) ignores them.

For example:

- `JSON.stringify({a:1,b:2}) == '{"a":1,"b":2}'`
- `E.toJS({a:1,b:2}) == '{a:1,b:2}'`

Note: Strings generated with [E.toJS](#) can't be reliably parsed by [JSON.parse](#) - however they are valid JS so will work with [eval](#) (but this has security implications if you don't trust the source of the string).

On the desktop [JSON5 parsers](#) will parse the strings produced by [E.toJS](#) without trouble.

[E.toString](#) =>**Call type:**[\(top\)](#)

```
E.toString(args, ...)
```

Parameters

args, ... - The arguments to convert to a String

Returns

A String

Description

Returns a [String](#) representing the data in the arguments.

This creates a string from the given arguments in the same way as `E.toUint8Array`. If each argument is:

- A String or an Array, each element is traversed and added as an 8 bit character
- `{data : ..., count : N}` causes `data` to be repeated `count` times
- `{callback : fn}` calls the function and adds the result
- Anything else is converted to a character directly.

In the case where there's one argument which is an 8 bit typed array backed by a flat string of the same length, the backing string will be returned without doing a copy or other allocation. The same applies if there's a single argument which is itself a flat string.

```JS E.toString(0,1,2,"Hi",3) "\0\1\2Hi\3" E.toString(1,2,{data:[3,4], count:4},5,6) "\1\2\3\4\3\4\3\4\3\4\5\6"

`E.toString(1,2,{callback : () => "Hello World"},5,6) ="\1\2Hello World\5\6"` ```

**Note:** Prior to Espruino 2v18 `E.toString` would always return a flat string, or would return `undefined` if one couldn't be allocated. Now, it will return a normal (fragmented) String if a contiguous chunk of memory cannot be allocated. You can still check if the returned value is a Flat string using `E.getAddressOf(str, true) != 0`, or can use `E.toFlatString` instead.

## [event E.touch](#) ⇒

---

Call type:

[\(top\)](#)

```
E.on('touch', function(x, y, b) { ... });
```

Parameters

x - X coordinate in display coordinates  
y - Y coordinate in display coordinates  
b - Touch count - 0 for released, 1 for pressed

Description

This event is called when a full touchscreen device on an Espruino is interacted with.

**Note:** This event is not implemented on Bangle.js because it only has a two area touchscreen.

To use the touchscreen to draw lines, you could do:

```
var last;
E.on('touch', t=>{
 if (last) g.lineTo(t.x, t.y);
 else g.moveTo(t.x, t.y);
 last = t.b;
});
```

## [E.toUint8Array](#) ⇒

---

Call type:

[\(top\)](#)

```
E.toUint8Array(args, ...)
```

Parameters

args, ... - The arguments to convert to a Uint8Array

Returns

A Uint8Array

Description

This creates a Uint8Array from the given arguments. These are handled as follows:

- `Number` -> read as an integer, using the lowest 8 bits
- `String` -> use each character's numeric value (e.g. `String.charCodeAt(...)`)
- `Array` -> Call itself on each element
- `ArrayBuffer` or Typed Array -> use the lowest 8 bits of each element
- `Object`:
  - `{data:..., count: int}` -> call itself `object.count` times, on `object.data`
  - `{callback : function}` -> call the given function, call itself on return value

For example:

```
E.toUint8Array([1,2,3])
=new Uint8Array([1, 2, 3])
E.toUint8Array([1,{data:2,count:3},3])
=new Uint8Array([1, 2, 2, 2, 3])
E.toUint8Array("Hello")
=new Uint8Array([72, 101, 108, 108, 111])
E.toUint8Array(["hi",{callback:function() { return [1,2,3] }}}]
=new Uint8Array([104, 105, 1, 2, 3])
```

## [E.unmountSD](#) ⇒

---

Call type:

[\(top\)](#)

```
E.unmountSD()
```

### Description

Unmount the SD card, so it can be removed. If you remove the SD card without calling this you may cause corruption, and you will be unable to access another SD card until you reset Espruino or call [E.unmountSD\(\)](#).

## [E.variance](#) ⇒

---

Call type:

[\(top\)](#)

```
E.variance(arr, mean)
```

### Parameters

arr - The array to work out the variance for

mean - The mean value of the array

### Returns

The variance of the given buffer

### Description

Work out the variance of the contents of the given Array, String or ArrayBuffer and return the result. This is equivalent to

```
v=0;for (i in arr)
v+=Math.pow(mean-arr[i],2)
```

**Note:** This is not available in devices with low flash memory

---

## [Error Class](#)

---

The base class for runtime errors

[\(top\)](#)

### Methods and Fields

- [constructor Error\(message\)](#)
- [function Error.toString\(\)](#)

---

### [constructor Error](#) ⇒

---

[View MDN documentation](#)

#### Call type:

`new Error(message)`

#### Parameters

`message` - [optional] An message string

#### Returns

An Error object

#### Description

Creates an Error object

---

### [function Error.toString](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

`function Error.toString()`

#### Returns

A String

---

## [ESP32 Class](#)

---

Class containing utility functions for the [ESP32](#)

[\(top\)](#)

### Methods and Fields

- [ESP32.deepSleep\(us\)](#)
- [ESP32.deepSleepExt0\(pin, level\)](#)
- [ESP32.deepSleepExt1\(pinVar, mode\)](#)
- [ESP32.enableBLE\(enable\)](#)
- [ESP32.enableWifi\(enable\)](#)
- [ESP32.getState\(\)](#)
- [ESP32.getWakeupCause\(\)](#)
- [ESP32.reboot\(\)](#)
- [ESP32.setAtten\(pin, atten\)](#)
- [ESP32.setBLE\\_Debug\(level\)](#)
- [ESP32.setOTAVoid\(isValid\)](#)

---

### [ESP32.deepSleep](#) ⇒

---

#### Call type:

[\(top\)](#)

`ESP32.deepSleep(us)`

#### Parameters

`us` - Sleep time in us

#### Description

Put device in deepsleep state for "us" microseconds.

**Note:** This is only available in ESP32 boards

---

### [ESP32.deepSleepExt0](#) ⇒

---

#### Call type:

[\(top\)](#)

`ESP32.deepSleepExt0(pin, level)`

#### Parameters

`pin` - Pin to trigger wakeup

`level` - Logic level to trigger

#### Description

Put device in deepsleep state until interrupted by pin "pin". Eligible pin numbers are restricted to those [GPIOs designated as RTC GPIOs](#).

**Note:** This is only available in ESP32 boards

---

### [ESP32.deepSleepExt1](#) ⇒

---

#### Call type:

[\(top\)](#)

`ESP32.deepSleepExt1(pinVar, mode)`

#### Parameters

`pinVar` - Array of Pins to trigger wakeup

`mode` - Trigger mode

## Description

Put device in deepsleep state until interrupted by pins in the "pinVar" array. The trigger "mode" determines the pin state which will wake up the device. Valid modes are:

- 0: `ESP_EXT1_WAKEUP_ALL_LOW` - all nominated pins must be set LOW to trigger wakeup
- 1: `ESP_EXT1_WAKEUP_ANY_HIGH` - any of nominated pins set HIGH will trigger wakeup

Eligible pin numbers are restricted to those [GPIOs designated as RTC GPIOs](#).

**Note:** This is only available in ESP32 boards

---

## [ESP32.enableBLE](#) ⇒

### Call type:

[\(top\)](#)

```
ESP32.enableBLE(enable)
```

### Parameters

`enable` - switches Bluetooth on or off

## Description

Switches Bluetooth off/on, removes saved code from Flash, resets the board, and on restart creates jsVars depending on available heap (actual additional 1800)

**Note:** This is only available in devices with Bluetooth LE capability

---

## [ESP32.enableWifi](#) ⇒

### Call type:

[\(top\)](#)

```
ESP32.enableWifi(enable)
```

### Parameters

`enable` - switches Wifi on or off

## Description

Switches Wifi off/on, removes saved code from Flash, resets the board, and on restart creates jsVars depending on available heap (actual additional 3900)

**Note:** This is only available in ESP32 boards

---

## [ESP32.getState](#) ⇒

### Call type:

[\(top\)](#)

```
ESP32.getState()
```

### Returns

The state of the ESP32

## Description

Returns an object that contains details about the state of the ESP32 with the following fields:

- `sdkVersion` - Version of the SDK.
- `freeHeap` - Amount of free heap in bytes.
- `BLE` - Status of BLE, enabled if true.
- `wifi` - Status of Wifi, enabled if true.
- `minHeap` - Minimum heap, calculated by `heap_caps_get_minimum_free_size`

**Note:** This is only available in ESP32 boards

## [ESP32.getWakeupCause](#) ⇒

---

### Call type:

`ESP32.getWakeupCause()`

### Returns

The cause of the ESP32's wakeup from sleep

### Description

Returns a variable identifying the cause of wakeup from deep sleep. Possible causes include:

- 0 : `ESP_SLEEP_WAKEUP_UNDEFINED` - reset was not caused by exit from deep sleep
- 2 : `ESP_SLEEP_WAKEUP_EXT0` - Wakeup caused by external signal using RTC\_IO
- 3 : `ESP_SLEEP_WAKEUP_EXT1` - Wakeup caused by external signal using RTC\_CNTL
- 4 : `ESP_SLEEP_WAKEUP_TIMER` - Wakeup caused by timer
- 5 : `ESP_SLEEP_WAKEUP_TOUCHPAD` - Wakeup caused by touchpad
- 6 : `ESP_SLEEP_WAKEUP_ULP` - Wakeup caused by ULP program

**Note:** This is only available in ESP32 boards

## [ESP32.reboot](#) ⇒

---

### Call type:

`ESP32.reboot()`

[\(top\)](#)

### Description

Perform a hardware reset/reboot of the ESP32.

**Note:** This is only available in ESP32 boards

## [ESP32.setAtten](#) ⇒

---

### Call type:

[\(top\)](#)

`ESP32.setAtten(pin, atten)`

### Parameters

`pin` - Pin for Analog read

`atten` - Attenuate factor

## [ESP32.setBLE\\_Debug](#) ⇒

---

### Call type:

[\(top\)](#)

`ESP32.setBLE_Debug(level)`

### Parameters

`level` - which events should be shown (GAP=1, GATTS=2, GATTC=4). Use 255 for everything

## [ESP32.setOTAVoid](#) ⇒

---

### Call type:

[\(top\)](#)

`ESP32.setOTAVoid(isValid)`

## Parameters

`isValid` - Set whether this app is valid or not. If `isValid==false` the device will reboot.

## Description

This function is useful for ESP32 [OTA Updates](#)

Normally Espruino is uploaded to the `factory` partition so this isn't so useful, but it is possible to upload Espruino to the `ota_0` partition (or `ota_1` if a different table has been added).

If this is the case, you can use this function to mark the currently running version of Espruino as good or bad. \* If set as valid, Espruino will continue running, and the fact that everything is ok is written to flash \* If set as invalid (false) Espruino will mark itself as not working properly and will reboot. The ESP32 bootloader will then start and will load any other partition it can find that is marked as ok.

**Note:** This is only available in ESP32 boards

## [ESP8266 Library](#)

---

The ESP8266 library is specific to the ESP8266 version of Espruino, i.e., running Espruino on an ESP8266 module (not to be confused with using the ESP8266 as Wifi add-on to an Espruino board). This library contains functions to handle ESP8266-specific actions. For example:

[\(top\)](#)

```
var esp8266 = require('ESP8266');
esp8266.reboot();
```

performs a hardware reset of the module.

Class containing utility functions for the [ESP8266](#)

### Methods and Fields

- [require\("ESP8266"\).crc32\(arrayOfData\)](#)
- [require\("ESP8266"\).deepSleep\(micros, option\)](#)
- [require\("ESP8266"\).dumpSocketInfo\(\)](#)
- [require\("ESP8266"\).getFreeFlash\(\)](#)
- [require\("ESP8266"\).getResetInfo\(\)](#)
- [require\("ESP8266"\).getState\(\)](#)
- [require\("ESP8266"\).logDebug\(enable\)](#)
- [require\("ESP8266"\).neopixelWrite\(pin, arrayOfData\)](#)
- [require\("ESP8266"\).ping\(ipAddr, pingCallback\)](#)
- [require\("ESP8266"\).printLog\(\)](#)
- [require\("ESP8266"\).readLog\(\)](#)
- [require\("ESP8266"\).reboot\(\)](#)
- [require\("ESP8266"\).setCPUFreq\(freq\)](#)
- [require\("ESP8266"\).setLog\(mode\)](#)

### [ESP8266.crc32](#) ⇒

---

#### Call type:

[\(top\)](#)

```
require("ESP8266").crc32(arrayOfData)
```

#### Parameters

arrayOfData - Array of data to CRC

#### Returns

32-bit CRC

### [ESP8266.deepSleep](#) ⇒

---

#### Call type:

[\(top\)](#)

```
require("ESP8266").deepSleep(micros, option)
```

#### Parameters

micros - Number of microseconds to sleep.

option - possible values are 0, 1, 2 or 4

#### Description

Put the ESP8266 into 'deep sleep' for the given number of microseconds, reducing power consumption drastically.

meaning of option values:

0 - the 108th Byte of init parameter decides whether RF calibration will be performed or not.

1 - run RF calibration after waking up. Power consumption is high.

2 - no RF calibration after waking up. Power consumption is low.

4 - no RF after waking up. Power consumption is the lowest.

**Note:** unlike normal Espruino boards' 'deep sleep' mode, ESP8266 deep sleep actually turns off the processor. After the given number of microseconds have elapsed, the ESP8266 will restart as if power had been turned off and then back on. *All contents of RAM will be lost*. Connect GPIO 16 to RST to enable wakeup.

**Special:** 0 microseconds cause sleep forever until external wakeup RST pull down occurs.

**Note:** This is only available in ESP8266 boards running Espruino

## [ESP8266.dumpSocketInfo](#) ⇒

---

**Call type:**

[\(top\)](#)

```
require("ESP8266").dumpSocketInfo()
```

**Description**

Dumps info about all sockets to the log. This is for troubleshooting the socket implementation.

**Note:** This is only available in ESP8266 boards running Espruino

## [ESP8266.getFreeFlash](#) ⇒

---

**Call type:**

[\(top\)](#)

```
require("ESP8266").getFreeFlash()
```

**Returns**

Array of objects with `addr` and `length` properties describing the free flash areas available

**Description**

**Note:** This is deprecated. Use `require("Flash").getFree()`

**Note:** This is only available in ESP8266 boards running Espruino

## [ESP8266.getResetInfo](#) ⇒

---

**Call type:**

[\(top\)](#)

```
require("ESP8266").getResetInfo()
```

**Returns**

An object with the reset cause information

**Description**

At boot time the esp8266's firmware captures the cause of the reset/reboot. This function returns this information in an object with the following fields:

- `reason`: "power on", "wdt reset", "exception", "soft wdt", "restart", "deep sleep", or "reset pin"
- `exccause`: exception cause
- `epc1, epc2, epc3`: instruction pointers
- `excvaddr`: address being accessed
- `depc`: (?)

**Note:** This is only available in ESP8266 boards running Espruino

## [ESP8266.getState](#) ⇒

---

**Call type:**

[\(top\)](#)

```
require("ESP8266").getState()
```

## Returns

The state of the ESP8266

## Description

Returns an object that contains details about the state of the ESP8266 with the following fields:

- `sdkVersion` - Version of the SDK.
- `cpuFrequency` - CPU operating frequency in Mhz.
- `freeHeap` - Amount of free heap in bytes.
- `maxCon` - Maximum number of concurrent connections.
- `flashMap` - Configured flash size&map: '512KB:256/256' .. '4MB:512/512'
- `flashKB` - Configured flash size in KB as integer
- `flashChip` - Type of flash chip as string with manufacturer & chip, ex: '0xEF 0x4016'

**Note:** This is only available in ESP8266 boards running Espruino

---

## [ESP8266.logDebug](#) ⇒

### Call type:

[\(top\)](#)

```
require("ESP8266").logDebug(enable)
```

### Parameters

`enable` - Enable or disable the debug logging.

### Description

Enable or disable the logging of debug information. A value of `true` enables debug logging while a value of `false` disables debug logging. Debug output is sent to UART1 (gpio2).

**Note:** This is only available in ESP8266 boards running Espruino

---

## [ESP8266.neopixelWrite](#) ⇒

### Call type:

[\(top\)](#)

```
require("ESP8266").neopixelWrite(pin, arrayOfData)
```

### Parameters

`pin` - Pin for output signal.

`arrayOfData` - Array of LED data.

### Description

**This function is deprecated.** Please use

```
require("neopixel").write(pin,
data)
```

instead

**Note:** This is only available in ESP8266 boards running Espruino

---

## [ESP8266.ping](#) ⇒

### Call type:

[\(top\)](#)

```
require("ESP8266").ping(ipAddr, pingCallback)
```

### Parameters

`ipAddr` - A string representation of an IP address.

`pingCallback` - Optional callback function.

## Description

**DEPRECATED** - please use `wifi.ping` instead.

Perform a network ping request. The parameter can be either a String or a numeric IP address.

**Note:** This is only available in ESP8266 boards running Espruino

## [ESP8266.printLog](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("ESP8266").printLog()
```

## Description

Prints the contents of the debug log to the console.

**Note:** This is only available in ESP8266 boards running Espruino

## [ESP8266.readLog](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("ESP8266").readLog()
```

## Description

Returns one line from the log or up to 128 characters.

**Note:** This is only available in ESP8266 boards running Espruino

## [ESP8266.reboot](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("ESP8266").reboot()
```

## Description

Perform a hardware reset/reboot of the esp8266.

**Note:** This is only available in ESP8266 boards running Espruino

## [ESP8266.setCPUFreq](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("ESP8266").setCPUFreq(freq)
```

## Parameters

`freq` - Desired frequency - either 80 or 160.

## Description

**Note:** This is deprecated. Use `E.setClock(80/160)` **Note:** Set the operating frequency of the ESP8266 processor. The default is 160Mhz.

**Warning:** changing the cpu frequency affects the timing of some I/O operations, notably of software SPI and I2C, so things may be a bit slower at 80Mhz.

**Note:** This is only available in ESP8266 boards running Espruino

**Call type:**

([top](#))

```
require("ESP8266").setLog(mode)
```

**Parameters**

mode - Debug log mode: 0=off, 1=in-memory only, 2=in-mem and uart0, 3=in-mem and uart1.

**Description**

Set the debug logging mode. It can be disabled (which frees ~1.2KB of heap), enabled in-memory only, or in-memory and output to a UART.

**Note:** This is only available in ESP8266 boards running Espruino

---

## [Ethernet Class](#)

---

An instantiation of an Ethernet network adaptor

[\(top\)](#)

### Methods and Fields

- [function Ethernet.getHostname\(callback\)](#)
- [function Ethernet.getIP\(options\)](#)
- [function Ethernet.getStatus\(options\)](#)
- [function Ethernet.setHostname\(hostname, callback\)](#)
- [function Ethernet.setIP\(options, callback\)](#)

---

### [function Ethernet.getHostname](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function Ethernet.getHostname(callback)
```

#### Parameters

callback - [optional] An `callback(err, hostname)` function to be called back with the status information.

#### Returns

See description above

#### Description

Returns the hostname

**Note:** This is only available in builds with support for WIZnet Ethernet modules built in

---

### [function Ethernet.getIP](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function Ethernet.getIP(options)
```

#### Parameters

options - [optional] An `callback(err, ipinfo)` function to be called back with the IP information.

#### Returns

See description above

#### Description

Get the current IP address, subnet, gateway and mac address.

**Note:** This is only available in builds with support for WIZnet Ethernet modules built in

---

### [function Ethernet.getStatus](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function Ethernet.getStatus(options)
```

#### Parameters

`options` - [optional] An `callback(err, status)` function to be called back with the status information.

## Returns

See description above

## Description

Get the current status of the ethernet device

**Note:** This is only available in builds with support for WIZnet Ethernet modules built in

## [function Ethernet.setHostname](#) ⇒

---

### Call type:

[\(top\)](#)

```
function Ethernet.setHostname(hostname, callback)
```

### Parameters

`hostname` - hostname as string

`callback` - [optional] An `callback(err)` function to be called back with null or error text.

## Returns

True on success

## Description

Set hostname used during the DHCP request. Minimum 8 and maximum 12 characters, best set before calling `eth.setIP()`. Default is WIZnet010203, 010203 is the default nic as part of the mac.

**Note:** This is only available in builds with support for WIZnet Ethernet modules built in

## [function Ethernet.setIP](#) ⇒

---

### Call type:

[\(top\)](#)

```
function Ethernet.setIP(options, callback)
```

### Parameters

`options` - Object containing IP address options { `ip : '1.2.3.4'`, `subnet : '....'`, `gateway: '....'`, `dns:'....'`, `mac:':::::'` }, or do not supply an object in order to force DHCP.

`callback` - [optional] An `callback(err)` function to invoke when ip is set. `err==null` on success, or a string on failure.

## Returns

True on success

## Description

Set the current IP address or get an IP from DHCP (if no options object is specified)

If 'mac' is specified as an option, it must be a string of the form "00:01:02:03:04:05" The default mac is 00:08:DC:01:02:03.

**Note:** This is only available in builds with support for WIZnet Ethernet modules built in

---

## [File Class](#)

---

This is the File object - it allows you to stream data to and from files (As opposed to the `require('fs').readFile(..)` style functions that read an entire file).

[\(top\)](#)

To create a File object, you must type

```
var fd =
E.openFile('filepath', 'mode')
```

- see [E.openFile](#) for more information.

**Note:** If you want to remove an SD card after you have started using it, you *must* call [E.unmountSD\(\)](#) or you may cause damage to the card.

### Methods and Fields

- [function File.close\(\)](#)
- [function File.pipe\(destination, options\)](#)
- [function File.read\(length\)](#)
- [function File.seek\(nBytes\)](#)
- [function File.skip\(nBytes\)](#)
- [function File.write\(buffer\)](#)

---

### [function File.close](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function File.close()
```

#### Description

Close an open file.

---

### [function File.pipe](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function File.pipe(destination, options)
```

#### Parameters

`destination` - The destination file/stream that will receive content from the source.

`options` - [optional] An object { `chunkSize : int=32`, `end : bool=true`, `complete : function` }

`chunkSize` : The amount of data to pipe from source to destination at a time

`complete` : a function to call when the pipe activity is complete

`end` : call the 'end' function on the destination when the source is finished

#### Description

Pipe this file to a stream (an object with a 'write' method)

**Note:** This is not available in devices with low flash memory

---

### [function File.read](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function File.read(length)
```

#### Parameters

`length` - is an integer specifying the number of bytes to read.

## Returns

A string containing the characters that were read

## Description

Read data in a file in byte size chunks

---

## [function File.seek](#) ⇒

### Call type:

[\(top\)](#)

```
function File.seek(nBytes)
```

### Parameters

nBytes - is an integer specifying the number of bytes to skip forwards.

## Description

Seek to a certain position in the file

---

## [function File.skip](#) ⇒

### Call type:

[\(top\)](#)

```
function File.skip(nBytes)
```

### Parameters

nBytes - is a positive integer specifying the number of bytes to skip forwards.

## Description

Skip the specified number of bytes forward in the file

---

## [function File.write](#) ⇒

### Call type:

[\(top\)](#)

```
function File.write(buffer)
```

### Parameters

buffer - A string containing the bytes to write

## Returns

the number of bytes written

## Description

Write data to a file.

**Note:** By default this function flushes all changes to the SD card, which makes it slow (but also safe!). You can use `E.setFlags({unsyncFiles:1})` to disable this behaviour and really speed up writes - but then you must be sure to close all files you are writing before power is lost or you will cause damage to your SD card's filesystem.

## [Flash Library](#)

---

This module allows you to read and write the nonvolatile flash memory of your device.

[\(top\)](#)

Also see the [Storage](#) library, which provides a safer file-like interface to nonvolatile storage.

It should be used with extreme caution, as it is easy to overwrite parts of Flash memory belonging to Espruino or even its bootloader. If you damage the bootloader then you may need external hardware such as a USB-TTL converter to restore it. For more information on restoring the bootloader see

### **Advanced Reflashing**

in your board's reference pages.

To see which areas of memory you can and can't overwrite, look at the values reported by `process.memory()`.

**Note:** On Nordic platforms there are checks in place to help you avoid 'bricking' your device before damaging the bootloader. You can disable these with `E.setFlags({unsafeFlash:1})`

### Methods and Fields

- `require("Flash").erasePage(addr)`
- `require("Flash").getFree()`
- `require("Flash").getPage(addr)`
- `require("Flash").read(length,addr)`
- `require("Flash").write(data,addr)`

### [Flash.erasePage](#) →

---

#### Call type:

[\(top\)](#)

```
require("Flash").erasePage(addr)
```

#### Parameters

`addr` - An address in the page that is to be erased

#### Description

Erase a page of flash memory

**Note:** This is not available in devices with low flash memory

### [Flash.getFree](#) →

---

#### Call type:

[\(top\)](#)

```
require("Flash").getFree()
```

#### Returns

Array of objects with `addr` and `length` properties

#### Description

This method returns an array of objects of the form `{addr : #, length : #}`, representing contiguous areas of flash memory in the chip that are not used for anything.

The memory areas returned are on page boundaries. This means that you can safely erase the page containing any address here, and you won't risk deleting part of the Espruino firmware.

**Note:** This is not available in devices with low flash memory

### [Flash.getPage](#) →

---

**Call type:**[\(top\)](#)

```
require("Flash").getPage(addr)
```

**Parameters**

addr - An address in memory

**Returns**

An object of the form { `addr : #, length : #`}, where `addr` is the start address of the page, and `length` is the length of it (in bytes). Returns undefined if no page at address

**Description**

Returns the start and length of the flash page containing the given address.

**Note:** This is not available in devices with low flash memory

---

**Flash.read** ⇒**Call type:**[\(top\)](#)

```
require("Flash").read(length, addr)
```

**Parameters**

`length` - The amount of data to read (in bytes)

`addr` - The address to start reading from

**Returns**

A Uint8Array of data

**Description**

Read flash memory from the given address

**Note:** This is not available in devices with low flash memory

---

**Flash.write** ⇒**Call type:**[\(top\)](#)

```
require("Flash").write(data, addr)
```

**Parameters**

`data` - The data to write

`addr` - The address to start writing from

**Description**

Write data into memory at the given address

In flash memory you may only turn bits that are 1 into bits that are 0. If you're writing data into an area that you have already written (so `read` doesn't return all `0xFF`) you'll need to call `erasePage` to clear the entire page.

**Note:** This is not available in devices with low flash memory

---

## [Float32Array Class](#)

---

This is the built-in JavaScript class for a typed array of 32 bit floating point values.

[\(top\)](#)

Instantiate this in order to efficiently store arrays of data (Espruino's normal arrays store data in a map, which is inefficient for non-sparse arrays).

Arrays of this type include all the methods from [ArrayBufferView](#)

### Methods and Fields

- [constructor Float32Array\(arr, byteOffset, length\)](#)

---

#### [constructor Float32Array](#) ⇒

---

[View MDN documentation](#)

##### Call type:

[\(top\)](#)

```
new Float32Array(arr, byteOffset, length)
```

##### Parameters

**arr** - The array or typed array to base this off, or an integer which is the array length

**byteOffset** - The byte offset in the ArrayBuffer (ONLY IF the first argument was an ArrayBuffer)

**length** - The length (ONLY IF the first argument was an ArrayBuffer)

##### Returns

A typed array

##### Description

Create a typed array based on the given input. Either an existing ArrayBuffer, an Integer as a Length, or a simple array. If an [ArrayBufferView](#) (e.g. [Uint8Array](#), rather than [ArrayBuffer](#)) is given, it will be completely copied rather than referenced.

---

## [Float64Array Class](#)

---

This is the built-in JavaScript class for a typed array of 64 bit floating point values.

[\(top\)](#)

Instantiate this in order to efficiently store arrays of data (Espruino's normal arrays store data in a map, which is inefficient for non-sparse arrays).

Arrays of this type include all the methods from [ArrayBufferView](#)

### Methods and Fields

- [constructor Float64Array\(arr, byteOffset, length\)](#)

---

#### [constructor Float64Array](#) ⇒

---

[View MDN documentation](#)

##### Call type:

[\(top\)](#)

```
new Float64Array(arr, byteOffset, length)
```

##### Parameters

`arr` - The array or typed array to base this off, or an integer which is the array length

`byteOffset` - The byte offset in the ArrayBuffer (ONLY IF the first argument was an ArrayBuffer)

`length` - The length (ONLY IF the first argument was an ArrayBuffer)

##### Returns

A typed array

##### Description

Create a typed array based on the given input. Either an existing ArrayBuffer, an Integer as a Length, or a simple array. If an [ArrayBufferView](#) (e.g. [Uint8Array](#), rather than [ArrayBuffer](#)) is given, it will be completely copied rather than referenced.

## [fs Library](#)

---

This library handles interfacing with a FAT32 filesystem on an SD card. The API is designed to be similar to node.js's - However Espruino does not currently support asynchronous file IO, so the functions behave like node.js's xxxxSync functions. Versions of the functions with 'Sync' after them are also provided for ([top](#)) compatibility.

To use this, you must type `var fs = require('fs')` to get access to the library

See [the page on File IO](#) for more information, and for examples on wiring up an SD card if your device doesn't come with one.

**Note:** If you want to remove an SD card after you have started using it, you *must* call `E.unmountSD()`, or you may cause damage to the card.

### Methods and Fields

- [`require\("fs"\).appendFile\(path, data\)`](#)
- [`require\("fs"\).appendFileSync\(path, data\)`](#)
- [`require\("fs"\).mkdir\(path\)`](#)
- [`require\("fs"\).mkdirSync\(path\)`](#)
- [`require\("fs"\).pipe\(source, destination, options\)`](#)
- [`require\("fs"\).readdir\(path\)`](#)
- [`require\("fs"\).readdirSync\(path\)`](#)
- [`require\("fs"\).readFile\(path\)`](#)
- [`require\("fs"\).readFileSync\(path\)`](#)
- [`require\("fs"\).statSync\(path\)`](#)
- [`require\("fs"\).unlink\(path\)`](#)
- [`require\("fs"\).unlinkSync\(path\)`](#)
- [`require\("fs"\).writeFile\(path, data\)`](#)
- [`require\("fs"\).writeFileSync\(path, data\)`](#)

### [fs.appendFile](#) ⇒

---

#### Call type:

([top](#))

```
require("fs").appendFile(path, data)
```

#### Parameters

`path` - The path of the file to write

`data` - The data to write to the file

#### Returns

True on success, false on failure

#### Description

Append the data to the given file, created a new file if it doesn't exist

NOTE: Espruino does not yet support Async file IO, so this function behaves like the 'Sync' version.

### [fs.appendFileSync](#) ⇒

---

#### Call type:

([top](#))

```
require("fs").appendFileSync(path, data)
```

#### Parameters

`path` - The path of the file to write

`data` - The data to write to the file

#### Returns

True on success, false on failure

## Description

Append the data to the given file, created a new file if it doesn't exist

**Note:** This is not available in devices with low flash memory

## [fs.mkdir](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("fs").mkdir(path)
```

### Parameters

path - The name of the directory to create

### Returns

True on success, or false on failure

## Description

Create the directory

NOTE: Espruino does not yet support Async file IO, so this function behaves like the 'Sync' version.

**Note:** This is not available in devices with low flash memory

## [fs.mkdirSync](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("fs").mkdirSync(path)
```

### Parameters

path - The name of the directory to create

### Returns

True on success, or false on failure

## Description

Create the directory

**Note:** This is not available in devices with low flash memory

## [fs.pipe](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("fs").pipe(source, destination, options)
```

### Parameters

source - The source file/stream that will send content.

destination - The destination file/stream that will receive content from the source.

options - [optional] An object { **chunkSize** : **int**=64, **end** : **bool**=**true**, **complete** : **function** }  
chunkSize : The amount of data to pipe from source to destination at a time  
complete : a function to call when the pipe activity is complete  
end : call the 'end' function on the destination when the source is finished

## [fs.readdir](#) ⇒

---

**Call type:**

```
require("fs").readdir(path)
```

**Parameters**

path - The path of the directory to list. If it is not supplied, " is assumed, which will list the root directory

**Returns**

An array of filename strings (or undefined if the directory couldn't be listed)

**Description**

List all files in the supplied directory, returning them as an array of strings.

NOTE: Espruino does not yet support Async file IO, so this function behaves like the 'Sync' version.

## [fs.readdirSync](#) ⇒

---

**Call type:**

```
require("fs").readdirSync(path)
```

**Parameters**

path - The path of the directory to list. If it is not supplied, " is assumed, which will list the root directory

**Returns**

An array of filename strings (or undefined if the directory couldn't be listed)

**Description**

List all files in the supplied directory, returning them as an array of strings.

**Note:** This is not available in devices with low flash memory

## [fs.readFile](#) ⇒

---

**Call type:**

```
require("fs").readFile(path)
```

**Parameters**

path - The path of the file to read

**Returns**

A string containing the contents of the file (or undefined if the file doesn't exist)

**Description**

Read all data from a file and return as a string

NOTE: Espruino does not yet support Async file IO, so this function behaves like the 'Sync' version.

## [fs.readFileSync](#) ⇒

---

**Call type:**

[\(top\)](#)

```
require("fs").readFileSync(path)
```

## Parameters

path - The path of the file to read

## Returns

A string containing the contents of the file (or undefined if the file doesn't exist)

## Description

Read all data from a file and return as a string.

**Note:** The size of files you can load using this method is limited by the amount of available RAM. To read files a bit at a time, see the [File](#) class.

**Note:** This is not available in devices with low flash memory

---

## [fs.statSync](#) →

### Call type:

```
require("fs").statSync(path)
```

## Parameters

path - The path of the file to get information on

## Returns

An object describing the file, or undefined on failure

## Description

Return information on the given file. This returns an object with the following fields:

size: size in bytes dir: a boolean specifying if the file is a directory or not mtime: A Date structure specifying the time the file was last modified

**Note:** This is not available in devices with low flash memory

---

## [fs.unlink](#) →

### Call type:

```
require("fs").unlink(path)
```

## Parameters

path - The path of the file to delete

## Returns

True on success, or false on failure

## Description

Delete the given file

NOTE: Espruino does not yet support Async file IO, so this function behaves like the 'Sync' version.

**Note:** This is not available in devices with low flash memory

---

## [fs.unlinkSync](#) →

### Call type:

[\(top\)](#)

[\(top\)](#)

[\(top\)](#)

```
require("fs").unlinkSync(path)
```

## Parameters

`path` - The path of the file to delete

## Returns

True on success, or false on failure

## Description

Delete the given file

**Note:** This is not available in devices with low flash memory

---

## [fs.writeFile](#) →

### Call type:

[\(top\)](#)

```
require("fs").writeFile(path, data)
```

## Parameters

`path` - The path of the file to write

`data` - The data to write to the file

## Returns

True on success, false on failure

## Description

Write the data to the given file

NOTE: Espruino does not yet support Async file IO, so this function behaves like the 'Sync' version.

---

## [fs.writeFileSync](#) →

### Call type:

[\(top\)](#)

```
require("fs").writeFileSync(path, data)
```

## Parameters

`path` - The path of the file to write

`data` - The data to write to the file

## Returns

True on success, false on failure

## Description

Write the data to the given file

**Note:** This is not available in devices with low flash memory

---

## [Function Class](#)

---

This is the built-in class for Functions

[\(top\)](#)

### Methods and Fields

- [function Function.apply\(this, args\)](#)
- [function Function.bind\(this, params, ...\)](#)
- [function Function.call\(this, params, ...\)](#)
- [constructor Function\(args, ...\)](#)
- [function Function.replaceWith\(newFunc\)](#)

---

### [function Function.apply](#) ⇒

---

[View MDN documentation](#)

#### Call type:

`function Function.apply(this, args)`

#### Parameters

`this` - The value to use as the 'this' argument when executing the function

`args` - Optional Array of Arguments

#### Returns

The return value of executing this function

#### Description

This executes the function with the supplied 'this' argument and parameters

---

### [function Function.bind](#) ⇒

---

[View MDN documentation](#)

#### Call type:

`function Function.bind(this, params, ...)`

#### Parameters

`this` - The value to use as the 'this' argument when executing the function

`params, ...` - Optional Default parameters that are prepended to the call

#### Returns

The 'bound' function

#### Description

This executes the function with the supplied 'this' argument and parameters

---

### [function Function.call](#) ⇒

---

[View MDN documentation](#)

**Call type:**[\(top\)](#)

```
function Function.call(this, params, ...)
```

**Parameters**

**this** - The value to use as the 'this' argument when executing the function

**params, ...** - Optional Parameters

**Returns**

The return value of executing this function

**Description**

This executes the function with the supplied 'this' argument and parameters

---

**constructor Function** ⇒[View MDN documentation](#)**Call type:**[\(top\)](#)

```
new Function(args, ...)
```

**Parameters**

**args, ...** - Zero or more arguments (as strings), followed by a string representing the code to run

**Returns**

A Number object

**Description**

Creates a function

---

**function Function.replaceWith** ⇒**Call type:**[\(top\)](#)

```
function Function.replaceWith(newFunc)
```

**Parameters**

**newFunc** - The new function to replace this function with

**Description**

This replaces the function with the one in the argument - while keeping the old function's scope. This allows inner functions to be edited, and is used when edit() is called on an inner function.

## [Graphics Class](#)

---

This class provides Graphics operations that can be applied to a surface.

[\(top\)](#)

Use `Graphics.createXXX` to create a graphics object that renders in the way you want. See [the Graphics page](#) for more information.

**Note:** On boards that contain an LCD, there is a built-in `g` object of type `graphics`. For instance to draw a line you'd type: `g.drawLine(0,0,100,100)`

### Methods and Fields

- [function Graphics.asBMP\(\)](#)
- [function Graphics.asImage\(type\)](#)
- [function Graphics.asURL\(\)](#)
- [function Graphics.blendColor\(col\\_a,col\\_b,amt\)](#)
- [function Graphics.blit\(options\)](#)
- [property Graphics.buffer](#)
- [function Graphics.clear\(reset\)](#)
- [function Graphics.clearRect\(x1,y1,x2,y2\)](#)
- [Graphics.createArrayBuffer\(width,height,bpp,options\)](#)
- [Graphics.createCallback\(width,height,bpp,callback\)](#)
- [Graphics.createImage\(str\)](#)
- [Graphics.createSDL\(width,height,bpp\)](#)
- [function Graphics.drawCircle\(x,y,rad\)](#)
- [function Graphics.drawCircleAA\(x,y,r\)](#)
- [function Graphics.drawEllipse\(x1,y1,x2,y2\)](#)
- [function Graphics.drawImage\(image,x,y,options\)](#)
- [function Graphics.drawImages\(layers,options\)](#)
- [function Graphics.drawLine\(x1,y1,x2,y2\)](#)
- [function Graphics.drawLineAA\(x1,y1,x2,y2\)](#)
- [function Graphics.drawPoly\(poly,closed\)](#)
- [function Graphics.drawPolyAA\(poly,closed\)](#)
- [function Graphics.drawRect\(x1,y1,x2,y2\)](#)
- [function Graphics.drawString\(str,x,y,solid\)](#)
- [function Graphics.dump\(\)](#)
- [function Graphics.fillCircle\(x,y,rad\)](#)
- [function Graphics.fillEllipse\(x1,y1,x2,y2\)](#)
- [function Graphics.fillPoly\(poly\)](#)
- [function Graphics.fillPolyAA\(poly\)](#)
- [function Graphics.fillRect\(x1,y1,x2,y2\)](#)
- [function Graphics.flip\(all\)](#)
- [function Graphics.floodFill\(x,y,col\)](#)
- [function Graphics.getBgColor\(\)](#)
- [function Graphics.getBPP\(\)](#)
- [function Graphics.getColor\(\)](#)
- [function Graphics.getFont\(\)](#)
- [function Graphics.getFontHeight\(\)](#)
- [function Graphics.getFonts\(\)](#)
- [function Graphics.getHeight\(\)](#)
- [Graphics.getInstance\(\)](#)
- [function Graphics.getModified\(reset\)](#)
- [function Graphics.getPixel\(x,y\)](#)
- [function Graphics.getVectorFontPolys\(str,options\)](#)
- [function Graphics.getWidth\(\)](#)
- [function Graphics.imageMetrics\(str\)](#)
- [function Graphics.lineTo\(x,y\)](#)
- [function Graphics.moveTo\(x,y\)](#)
- [function Graphics.quadraticBezier\(arr,options\)](#)
- [function Graphics.reset\(\)](#)
- [function Graphics.scroll\(x,y\)](#)
- [function Graphics.setBgColor\(r,g,b\)](#)
- [function Graphics.setClipRect\(x1,y1,x2,y2\)](#)
- [function Graphics.setColor\(r,g,b\)](#)
- [function Graphics.setFont\(name,size\)](#)
- [function Graphics.setFont12x20\(scale\)](#)
- [function Graphics.setFont6x15\(scale\)](#)
- [function Graphics.setFontAlign\(x,y,rotation\)](#)
- [function Graphics.setFontBitmap\(\)](#)
- [function Graphics.setFontCustom\(bitmap,firstChar,width,height\)](#)
- [function Graphics.setFontVector\(size\)](#)
- [function Graphics.setPixel\(x,y,col\)](#)
- [function Graphics.setRotation\(rotation,reflect\)](#)
- [function Graphics.setTheme\(theme\)](#)
- [function Graphics.stringMetrics\(str\)](#)
- [function Graphics.stringWidth\(str\)](#)
- [property Graphics.theme](#)

- [function Graphics.toColor\(r,g,b\)](#)
- [function Graphics.transformVertices\(verts,transformation\)](#)
- [function Graphics.wrapString\(str,maxWidth\)](#)

## [function Graphics.asBMP](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.asBMP()
```

**Returns**

A String representing the Graphics as a Windows BMP file (or 'undefined' if not possible)

**Description**

Create a Windows BMP file from this Graphics instance, and return it as a String.

**Note:** This is not available in devices with low flash memory or 'Original' Espruino boards

## [function Graphics.asImage](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.asImage(type)
```

**Parameters**

`type` - The type of image to return. Either `object`/undefined to return an image object, or `string` to return an image string

**Returns**

An Image that can be used with [graphics.drawImage](#)

**Description**

Return this Graphics object as an Image that can be used with [Graphics.drawImage](#). Check out [the Graphics reference page](#) for more information on images.

Will return undefined if data can't be allocated for the image.

The image data itself will be referenced rather than copied if:

- An image `object` was requested (not `string`)
- The Graphics instance was created with [Graphics.createArrayBuffer](#)
- Is 8 bpp *OR* the `{msb:true}` option was given
- No other format options (zigzag/etc) were given

Otherwise data will be copied, which takes up more space and may be quite slow.

If the `Graphics` object contains `transparent` or `palette` fields, [as you might find in an image](#), those will be included in the generated image too.

**Note:** This is not available in devices with low flash memory

## [function Graphics.asURL](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.asURL()
```

**Returns**

A String representing the Graphics as a URL (or 'undefined' if not possible)

**Description**

Create a URL of the form `data:image/bmp;base64,...` that can be pasted into the browser.

The Espruino Web IDE can detect this data on the console and render the image inline automatically.

**Note:** This is not available in devices with low flash memory or 'Original' Espruino boards

## [function Graphics.blendColor](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.blendColor(col_a, col_b, amt)
```

**Parameters**

col\_a - Color to blend from (either a single integer color value, or a string)

col\_b - Color to blend to (either a single integer color value, or a string)

amt - The amount to blend. 0=col\_a, 1=col\_b, 0.5=halfway between (and so on)

**Returns**

The color index represented by the blended colors

**Description**

Blend between two colors, and return the result.

```
// dark yellow - halfway between red and green
var col = gblendColor("#f00","#0f0", 0.5);
// Get a color 25% brighter than the theme's background colour
var col = gblendColor(g.theme.fg,g.theme.bg, 0.75);
// then...
g.setColor(col).fillRect(10,10,100,100);
```

**Note:** This is only available in devices with Antialiasing support included (Bangle.js or Linux)

## [function Graphics.blit](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.blit(options)
```

**Parameters**

options - options - see below

**Returns**

The instance of Graphics this was called on, to allow call chaining

**Description**

Blit one area of the screen (x1,y1 w,h) to another (x2,y2 w,h)

```
g.blit({
 x1:0, y1:0,
 w:32, h:32,
 x2:100, y2:100,
 setModified : true // should we set the modified area?
});
```

**Note:** This uses repeated pixel reads and writes, so will not work on platforms that don't support pixel reads.

**Note:** This is not available in devices with low flash memory or 'Original' Espruino boards

## [property Graphics.buffer](#) ⇒

---

**Call type:**

[\(top\)](#)

```
property Graphics.buffer
```

## Returns

An ArrayBuffer (or not defined on Graphics instances not created with [Graphics.createArrayBuffer](#))

## Description

On Graphics instances with an offscreen buffer, this is an [ArrayBuffer](#) that provides access to the underlying pixel data.

```
g=Graphics.createArrayBuffer(8,8,8)
g.drawLine(0,0,7,7)
print(new Uint8Array(g.buffer))
new Uint8Array([
255, 0, 0, 0, 0, 0, 0, 0,
0, 255, 0, 0, 0, 0, 0, 0,
0, 0, 255, 0, 0, 0, 0, 0,
0, 0, 0, 255, 0, 0, 0, 0,
0, 0, 0, 0, 255, 0, 0, 0,
0, 0, 0, 0, 0, 255, 0, 0,
0, 0, 0, 0, 0, 0, 255, 0,
0, 0, 0, 0, 0, 0, 0, 255])
```

---

## [function Graphics.clear](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.clear(reset)
```

### Parameters

reset - [optional] If **true**, resets the state of Graphics to the default (eg. Color, Font, etc) as if calling [Graphics.reset](#)

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Clear the LCD with the Background Color

---

## [function Graphics.clearRect](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.clearRect(x1, y1, x2, y2)
```

### Parameters

x1 - The left X coordinate OR an object containing {x,y,x2,y2} or {x,y,w,h}

y1 - The top Y coordinate

x2 - The right X coordinate

y2 - The bottom Y coordinate

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Fill a rectangular area in the Background Color

On devices with enough memory, you can specify {x,y,x2,y2,r} as the first argument, which allows you to draw a rounded rectangle.

**Note:** This is not available in devices with low flash memory

---

## [Graphics.createArrayBuffer](#) ⇒

## Call type:

[\(top\)](#)

```
Graphics.createArrayBuffer(width, height, bpp, options)
```

## Parameters

width - Pixels wide

height - Pixels high

bpp - Number of bits per pixel

options - An object of other options. { zigzag : `true/false(default)`, vertical\_byte : `true/false(default)`, msb : `true/false(default)`, color\_order: 'rgb'(default), 'bgr',etc }  
zigzag = whether to alternate the direction of scanlines for rows  
vertical\_byte = whether to align bits in a byte vertically or not  
msb = when bits<8, store pixels most significant bit first, when bits>8, store most significant byte first  
interleavex = Pixels 0,2,4,etc are from the top half of the image, 1,3,5,etc from the bottom half. Used for P3 LED panels.  
color\_order = re-orders the colour values that are supplied via setColor

## Returns

The new Graphics object

## Description

Create a Graphics object that renders to an Array Buffer. This will have a field called 'buffer' that can get used to get at the buffer itself

## [Graphics.createCallback](#) →

---

## Call type:

[\(top\)](#)

```
Graphics.createCallback(width, height, bpp, callback)
```

## Parameters

width - Pixels wide

height - Pixels high

bpp - Number of bits per pixel

callback - A function of the form `function(x,y,col)` that is called whenever a pixel needs to be drawn, or an object with:  
`{setPixel:function(x,y,col),fillRect:function(x1,y1,x2,y2,col)}`. All arguments are already bounds checked.

## Returns

The new Graphics object

## Description

Create a Graphics object that renders by calling a JavaScript callback function to draw pixels

**Note:** This is not available in devices with low flash memory

## [Graphics.createImage](#) →

---

## Call type:

[\(top\)](#)

```
Graphics.createImage(str)
```

## Parameters

str - A String containing a newline-separated image - space/. is 0, anything else is 1

## Returns

An Image object that can be used with [Graphics.drawImage](#)

## Description

Create a simple Black and White image for use with [Graphics.drawImage](#).

Use as follows:

```
var img = Graphics.createImage(`
XXXXXXXXXX
X X
X X X
X X X
X X
XXXXXXXXXX
`);
g.drawImage(img, x,y);
var img = Graphics.createImage(`
....
.XXX.
.X.X.
.XXX.
....
`);
g.drawImage(img, x,y);
```

If the characters at the beginning and end of the string are newlines, they will be ignored. Spaces are treated as 0, and any other character is a 1

**Note:** This is not available in devices with low flash memory or 'Original' Espruino boards

---

## [Graphics.createSDL](#) =>

### Call type:

[\(top\)](#)

```
Graphics.createSDL(width, height, bpp)
```

### Parameters

width - Pixels wide

height - Pixels high

bpp - Bits per pixel (8,16,24 or 32 supported)

### Returns

The new Graphics object

## Description

Create a Graphics object that renders to SDL window (Linux-based devices only)

**Note:** This is only available in Linux with SDL support compiled in

---

## [function Graphics.drawCircle](#) =>

### Call type:

[\(top\)](#)

```
function Graphics.drawCircle(x, y, rad)
```

### Parameters

x - The X axis

y - The Y axis

rad - The circle radius

### Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Draw an unfilled circle 1px wide in the Foreground Color

**Note:** This is not available in devices with low flash memory

## [function Graphics.drawCircleAA](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.drawCircleAA(x, y, r)
```

**Parameters**

x - Centre x-coordinate

y - Centre y-coordinate

r - Radius

**Returns**

The instance of Graphics this was called on, to allow call chaining

**Description**

Draw a circle, centred at (x,y) with radius r in the current foreground color

**Note:** This is only available in devices with Antialiasing support included (Bangle.js or Linux)

## [function Graphics.drawEllipse](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.drawEllipse(x1, y1, x2, y2)
```

**Parameters**

x1 - The left X coordinate

y1 - The top Y coordinate

x2 - The right X coordinate

y2 - The bottom Y coordinate

**Returns**

The instance of Graphics this was called on, to allow call chaining

**Description**

Draw an ellipse in the Foreground Color

**Note:** This is not available in devices with low flash memory

## [function Graphics.drawImage](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.drawImage(image, x, y, options)
```

**Parameters**

image - An image to draw, either a String or an Object (see below)

x - The X offset to draw the image

y - The Y offset to draw the image

options - options for scaling,rotation,etc (see below)

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Image can be:

- An object with the following fields

```
{ width : int, height : int, bpp :
 optional int, buffer : ArrayBuffer/String, transparent: optional int,
 palette : optional Uint16Array(2/4/16) }
```

.bpp = bits per pixel (default is 1), transparent (if defined) is the colour that will be treated as transparent, and palette is a color palette that each pixel will be looked up in first

• A String where the the first few bytes are: width,height,bpp,[transparent,]image\_bytes.... If a transparent colour is specified the top bit of bpp should be set.

• An ArrayBuffer Graphics object (if bpp<8, msb:true must be set) - this is disabled on devices without much flash memory available. If a Graphics object is supplied, it can also contain transparent/palette fields as if it were an image.

Draw an image at the specified position.

- If the image is 1 bit, the graphics foreground/background colours will be used.
- If img.palette is a Uint16Array or 2/4/16 elements, color data will be looked from the supplied palette
- On Bangle.js, 2 bit images blend from background(0) to foreground(1) colours
- On Bangle.js, 4 bit images use the Apple Mac 16 color palette
- On Bangle.js, 8 bit images use the Web Safe 216 color palette
- Otherwise color data will be copied as-is. Bitmaps are rendered MSB-first

If options is supplied, drawImage will allow images to be rendered at any scale or angle. If options.rotate is set it will center images at x,y. options must be an object of the form:

```
{
 rotate : float, // the amount to rotate the image in radians (default 0)
 scale : float, // the amount to scale the image up (default 1)
 frame : int // if specified and the image has frames of data
 // after the initial frame, draw one of those frames from the image
}
```

For example:

```
// In the top left of the screen
g.drawImage(img,0,0);
// In the top left of the screen, twice as big
g.drawImage(img,0,0,{scale:2});
// In the center of the screen, twice as big, 45 degrees
g.drawImage(img, g.getWidth()/2, g.getHeight()/2,
 {scale:2, rotate:Math.PI/4});
```

## function Graphics.drawImage ⇒

---

### Call type:

[\(top\)](#)

```
function Graphics.drawImage(layers, options)
```

### Parameters

layers - An array of objects {x,y,image,scale,rotate,center} (up to 3)

options - options for rendering - see below

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Draws multiple images *at once* - which avoids flicker on unbuffered systems like Bangle.js. Maximum layer count right now is 4.

```
layers = [{
 x : int, // x start position
 y : int, // y start position
 image : string/object,
 scale : float, // scale factor, default 1
 rotate : float, // angle in radians
 center : bool // center on x,y? default is top left
 repeat : should this image be repeated (tiled?)
 nobounds : bool // if true, the bounds of the image are not used to work out the default area to draw
}
```

```
]
options = { // the area to render. Defaults to rendering just enough to cover what's requested
 x,y,
 width,height
}
```

**Note:** This is only available in Bangle.js smartwatches and Linux-based builds

## [function Graphics.drawLine](#) ⇒

---

Call type:

[\(top\)](#)

```
function Graphics.drawLine(x1, y1, x2, y2)
```

### Parameters

x1 - The left

y1 - The top

x2 - The right

y2 - The bottom

### Returns

The instance of Graphics this was called on, to allow call chaining

### Description

Draw a line between x1,y1 and x2,y2 in the current foreground color

## [function Graphics.drawLineAA](#) ⇒

---

Call type:

[\(top\)](#)

```
function Graphics.drawLineAA(x1, y1, x2, y2)
```

### Parameters

x1 - The left

y1 - The top

x2 - The right

y2 - The bottom

### Returns

The instance of Graphics this was called on, to allow call chaining

### Description

Draw a line between x1,y1 and x2,y2 in the current foreground color

**Note:** This is only available in devices with Antialiasing support included (Bangle.js or Linux)

## [function Graphics.drawPoly](#) ⇒

---

Call type:

[\(top\)](#)

```
function Graphics.drawPoly(poly, closed)
```

### Parameters

poly - An array of vertices, of the form [x1,y1,x2,y2,x3,y3,etc]

`closed` - Draw another line between the last element of the array and the first

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Draw a polyline (lines between each of the points in `poly`) in the current foreground color

**Note:** there is a limit of 64 points (128 XY elements) for polygons

**Note:** This is not available in devices with low flash memory

---

## [function Graphics.drawPolyAA](#) ⇒

### Call type:

```
function Graphics.drawPolyAA(poly, closed)
```

### Parameters

`poly` - An array of vertices, of the form `[x1,y1,x2,y2,x3,y3,etc]`

`closed` - Draw another line between the last element of the array and the first

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Draw an **antialiased** polyline (lines between each of the points in `poly`) in the current foreground color

**Note:** there is a limit of 64 points (128 XY elements) for polygons

**Note:** This is only available in devices with Antialiasing support included (Bangle.js or Linux)

---

## [function Graphics.drawRect](#) ⇒

### Call type:

```
function Graphics.drawRect(x1, y1, x2, y2)
```

### Parameters

`x1` - The left X coordinate OR an object containing `{x,y,x2,y2}` or `{x,y,w,h}`

`y1` - The top Y coordinate

`x2` - The right X coordinate

`y2` - The bottom Y coordinate

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Draw an unfilled rectangle 1px wide in the Foreground Color

---

## [function Graphics.drawString](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.drawString(str, x, y, solid)
```

## Parameters

`str` - The string  
`x` - The X position of the leftmost pixel  
`y` - The Y position of the topmost pixel  
`solid` - For bitmap fonts, should empty pixels be filled with the background color?

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Draw a string of text in the current font.

```
g.drawString("Hello World", 10, 10);
```

Images may also be embedded inside strings (e.g. to render Emoji or characters not in the current font). To do this, just add 0 then the image string ([about Images](#)). For example:

```
g.drawString("Hi \0\7\5\1\x82 D\x17\xC0");
// draws:
// # # # #
// # # #
// ### ## #
// # # # # #
// # # ### #####
```

## [function Graphics.dump](#) ⇒

---

### Call type:

[\(top\)](#)

```
function Graphics.dump()
```

## Description

Output this image as a bitmap URL of the form `data:image/bmp;base64,....`. The Espruino Web IDE will detect this on the console and will render the image inline automatically.

This is identical to `console.log(g.asURL())` - it is just a convenient function for easy debugging and producing screenshots of what is currently in the Graphics instance.

**Note:** This may not work on some bit depths of Graphics instances. It will also not work for the main Graphics instance of Bangle.js 1 as the graphics on Bangle.js 1 are stored in write-only memory.

**Note:** This is not available in devices with low flash memory or 'Original' Espruino boards

## [function Graphics.fillCircle](#) ⇒

---

### Call type:

[\(top\)](#)

```
function Graphics.fillCircle(x, y, rad)
```

## Parameters

`x` - The X axis  
`y` - The Y axis  
`rad` - The circle radius

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Draw a filled circle in the Foreground Color

**Note:** This is not available in devices with low flash memory

## [function Graphics.fillEllipse](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.fillEllipse(x1, y1, x2, y2)
```

### Parameters

x1 - The left X coordinate

y1 - The top Y coordinate

x2 - The right X coordinate

y2 - The bottom Y coordinate

### Returns

The instance of Graphics this was called on, to allow call chaining

### Description

Draw a filled ellipse in the Foreground Color

**Note:** This is not available in devices with low flash memory

## [function Graphics.fillPoly](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.fillPoly(poly)
```

### Parameters

poly - An array of vertices, of the form [x1,y1,x2,y2,x3,y3,etc]

### Returns

The instance of Graphics this was called on, to allow call chaining

### Description

Draw a filled polygon in the current foreground color.

```
g.fillPoly([
 16, 0,
 31, 31,
 26, 31,
 16, 12,
 6, 28,
 0, 27]);
```

This fills from the top left hand side of the polygon (low X, low Y) *down to but not including* the bottom right. When placed together polygons will align perfectly without overdraw - but this will not fill the same pixels as `drawPoly` (drawing a line around the edge of the polygon).

**Note:** there is a limit of 64 points (128 XY elements) for polygons

**Note:** This is not available in devices with low flash memory

## [function Graphics.fillPolyAA](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.fillPolyAA(poly)
```

## Parameters

`poly` - An array of vertices, of the form `[x1,y1,x2,y2,x3,y3,etc]`

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Draw a filled polygon in the current foreground color.

```
g.fillPolyAA([
 16, 0,
 31, 31,
 26, 31,
 16, 12,
 6, 28,
 0, 27]);
```

This fills from the top left hand side of the polygon (low X, low Y) *down to but not including* the bottom right. When placed together polygons will align perfectly without overdraw - but this will not fill the same pixels as `drawPoly` (drawing a line around the edge of the polygon).

**Note:** there is a limit of 64 points (128 XY elements) for polygons

**Note:** This is only available in devices with Antialiasing support included (Bangle.js or Linux)

---

## [function Graphics.fillRect](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.fillRect(x1, y1, x2, y2)
```

## Parameters

`x1` - The left X coordinate OR an object containing `{x,y,x2,y2}` or `{x,y,w,h}`

`y1` - The top Y coordinate

`x2` - The right X coordinate

`y2` - The bottom Y coordinate

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Fill a rectangular area in the Foreground Color

On devices with enough memory, you can specify `{x,y,x2,y2,r}` as the first argument, which allows you to draw a rounded rectangle.

---

## [function Graphics.flip](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.flip(all)
```

## Parameters

`all` - [optional] (only on some devices) If `true` then copy all pixels, not just those that have changed.

## Description

On instances of graphics that drive a display with an offscreen buffer, calling this function will copy the contents of the offscreen buffer to the screen.

Call this when you have drawn something to Graphics and you want it shown on the screen.

If a display does not have an offscreen buffer, it may not have a `g.flip()` method.

On Bangle.js 1, there are different graphics modes chosen with `Bangle.setLCDMode()`. The default mode is unbuffered and in this mode `g.flip()` does not affect the screen contents.

On some devices, this command will attempt to only update the areas of the screen that have changed in order to increase speed. If you have accessed the `Graphics.buffer` directly then you may need to use `Graphics.flip(true)` to force a full update of the screen.

## [function Graphics.floodFill](#) ⇒

---

Call type:

[\(top\)](#)

```
function Graphics.floodFill(x, y, col)
```

### Parameters

x - X coordinate to start from

y - Y coordinate to start from

col - The color to fill with (if undefined, foreground is used)

### Returns

The instance of Graphics this was called on, to allow call chaining

### Description

Flood fills the given Graphics instance out from a particular point.

**Note:** This only works on Graphics instances that support readback with `getPixel`. It is also not capable of filling over dithered patterns (eg non-solid colours on Bangle.js 2)

**Note:** This is not available in devices with low flash memory or 'Original' Espruino boards

## [function Graphics.getBgColor](#) ⇒

---

Call type:

[\(top\)](#)

```
function Graphics.getBgColor()
```

### Returns

The integer value of the colour

### Description

Get the background color to use for subsequent drawing operations

## [function Graphics.getBPP](#) ⇒

---

Call type:

[\(top\)](#)

```
function Graphics.getBPP()
```

### Returns

The bits per pixel of this Graphics instance

### Description

The number of bits per pixel of this Graphics instance

**Note:** Bangle.js 2 behaves a little differently here. The display is 3 bit, so `getBPP` returns 3 and `asBMP/asImage/etc` return 3 bit images. However in order to allow dithering, the colors returned by `Graphics.getColor` and `Graphics.theme` are actually 16 bits.

## [function Graphics.getColor](#) ⇒

---

**Call type:**

```
function Graphics.getColor()
```

**Returns**

The integer value of the colour

**Description**

Get the color to use for subsequent drawing operations

[\(top\)](#)

## [function Graphics.getFont](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.getFont()
```

**Returns**

Get the name of the current font

**Description**

Get the font by name - can be saved and used with [Graphics.setFont](#).

Normally this might return something like "4x6", but if a scale factor is specified, a colon and then the size is reported, like "4x6:2"

**Note:** For custom fonts, **Custom** is currently reported instead of the font name.

**Note:** This is not available in devices with low flash memory

## [function Graphics.getFontHeight](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.getFontHeight()
```

**Returns**

The height in pixels of the current font

**Description**

Return the height in pixels of the current font

**Note:** This is not available in devices with low flash memory

## [function Graphics.getFonts](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.getFonts()
```

**Returns**

An array of font names

**Description**

Return an array of all fonts currently in the Graphics library.

**Note:** Vector fonts are specified as `vector#` where `#` is the font height. As there are effectively infinite fonts, just `vector` is included in the list.

**Note:** This is not available in devices with low flash memory

## [function Graphics.getHeight](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.getHeight()
```

**Returns**

The height of this Graphics instance

**Description**

The height of this Graphics instance

## [Graphics.getInstance](#) ⇒

---

**Call type:**

[\(top\)](#)

```
Graphics.getInstance()
```

**Returns**

An instance of [Graphics](#) or undefined

**Description**

On devices like Pixl.js or HYSTM boards that contain a built-in display this will return an instance of the graphics class that can be used to access that display.

Internally, this is stored as a member called `gfx` inside the 'hiddenRoot'.

## [function Graphics.getModified](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.getModified(reset)
```

**Parameters**

`reset` - Whether to reset the modified area or not

**Returns**

An object {x1,y1,x2,y2} containing the modified area, or undefined if not modified

**Description**

Return the area of the Graphics canvas that has been modified, and optionally clear the modified area to 0.

For instance if `g.setPixel(10,20)` was called, this would return

```
{x1:10,
y1:20, x2:10, y2:20}
```

**Note:** This is not available in devices with low flash memory

## [function Graphics.getPixel](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.getPixel(x, y)
```

## Parameters

x - The left  
y - The top

## Returns

The color

## Description

Get a pixel's color

---

## [function Graphics.getVectorFontPolys](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.getVectorFontPolys(str, options)
```

## Parameters

str - The string  
options - [optional] {x,y,w,h} (see below)

## Returns

An array of Uint8Arrays for vector font polygons

## Description

Return the current string as a series of polygons (using the current vector font). options is as follows:

- x - X offset of font (default 0)
- y - Y offset of font (default 0)
- w - Width of font (default 256) - the actual width will likely be less than this as most characters are non-square
- h - Height of font (default 256) - the actual height will likely be less than this as most characters don't fully fill the font box

```
g.getVectorFontPolys("Hi", {x:-80,y:-128});
```

**Note:** This is not available in devices with low flash memory and not NO\_VECTOR\_FONT

---

## [function Graphics.getWidth](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.getWidth()
```

## Returns

The width of this Graphics instance

## Description

The width of this Graphics instance

---

## [function Graphics.imageMetrics](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.imageMetrics(str)
```

## Parameters

`str` - The string

## Returns

An object containing `{width, height, bpp, transparent}` for the image

## Description

Return the width and height in pixels of an image (either Graphics, Image Object, Image String or ArrayBuffer). Returns `undefined` if image couldn't be decoded.

`frames` is also included is the image contains more information than you'd expect for a single bitmap. In this case the bitmap might be an animation with multiple frames

---

## [function Graphics.lineTo](#) ⇒

---

### Call type:

[\(top\)](#)

```
function Graphics.lineTo(x, y)
```

### Parameters

`x` - X value

`y` - Y value

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Draw a line from the last position of `lineTo` or `moveTo` to this position

---

## [function Graphics.moveTo](#) ⇒

---

### Call type:

[\(top\)](#)

```
function Graphics.moveTo(x, y)
```

### Parameters

`x` - X value

`y` - Y value

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Move the cursor to a position - see `lineTo`

---

## [function Graphics.quadraticBezier](#) ⇒

---

### Call type:

[\(top\)](#)

```
function Graphics.quadraticBezier(arr, options)
```

### Parameters

`arr` - An array of three vertices, six enties in form of `[x0, y0, x1, y1, x2, y2]`

`options` - number of points to calculate

## Returns

Array with calculated points

## Description

Calculate the square area under a Bezier curve.

x0,y0: start point x1,y1: control point y2,y2: end point

Max 10 points without start point.

**Note:** This is not available in devices with low flash memory or 'Original' Espruino boards

---

## [function Graphics.reset](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.reset()
```

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Reset the state of Graphics to the defaults (e.g. Color, Font, etc) that would have been used when Graphics was initialised.

---

## [function Graphics.scroll](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.scroll(x, y)
```

## Parameters

x - X direction. >0 = to right

y - Y direction. >0 = down

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Scroll the contents of this graphics in a certain direction. The remaining area is filled with the background color.

Note: This uses repeated pixel reads and writes, so will not work on platforms that don't support pixel reads.

**Note:** This is not available in devices with low flash memory or 'Original' Espruino boards

---

## [function Graphics.setBgColor](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.setBgColor(r, g, b)
```

## Parameters

r - Red (between 0 and 1) **OR** an integer representing the color in the current bit depth and color order **OR** a hexidecimal color string of the form '#012345'

g - Green (between 0 and 1)

b - Blue (between 0 and 1)

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Set the background color to use for subsequent drawing operations.

See [Graphics.setColor](#) for more information on the mapping of r, g, and b to pixel values.

**Note:** On devices with low flash memory, r **must** be an integer representing the color in the current bit depth. It cannot be a floating point value, and g and b are ignored.

---

## [function Graphics.setClipRect](#) ⇒

---

### Call type:

[\(top\)](#)

```
function Graphics.setClipRect(x1, y1, x2, y2)
```

### Parameters

x1 - Top left X coordinate

y1 - Top left Y coordinate

x2 - Bottom right X coordinate

y2 - Bottom right Y coordinate

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

This sets the 'clip rect' that subsequent drawing operations are clipped to sit between.

These values are inclusive - e.g. g.setClipRect(1,0,5,0) will ensure that only pixel rows 1,2,3,4,5 are touched on column 0.

**Note:** For maximum flexibility on Bangle.js 1, the values here are not range checked. For normal use, X and Y should be between 0 and getWidth() - 1 / getHeight() - 1.

**Note:** The x/y values here are rotated, so that if [Graphics.setRotation](#) is used they correspond to the coordinates given to the draw functions, *not to the physical device pixels*.

**Note:** This is not available in devices with low flash memory

---

## [function Graphics.setColor](#) ⇒

---

### Call type:

[\(top\)](#)

```
function Graphics.setColor(r, g, b)
```

### Parameters

r - Red (between 0 and 1) **OR** an integer representing the color in the current bit depth and color order **OR** a hexidecimal color string of the form '#012345'

g - [optional] Green (between 0 and 1)

b - [optional] Blue (between 0 and 1)

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Set the color to use for subsequent drawing operations.

If just `r` is specified as an integer, the numeric value will be written directly into a pixel. eg. On a 24 bit `Graphics` instance you set bright blue with either `g.setColor(0,0,1)` or `g.setColor(0x0000FF)`.

A good shortcut to ensure you get white on all platforms is to use `g.setColor(-1)`

The mapping is as follows:

- 32 bit: `r,g,b` => `0xFFrrggbb`
- 24 bit: `r,g,b` => `0xrrggbb`
- 16 bit: `r,g,b` => `0brrrrrggggggbbbb` (RGB565)
- Other bpp: `r,g,b` => white if `r+g+b > 50%`, otherwise black (use `r` on its own as an integer)

If you specified `color_order` when creating the `Graphics` instance, `r,g` and `b` will be swapped as you specified.

**Note:** On devices with low flash memory, `r` **must** be an integer representing the color in the current bit depth. It cannot be a floating point value, and `g` and `b` are ignored.

## [function Graphics.setFont](#) ⇒

---

Call type:

[\(top\)](#)

```
function Graphics.setFont(name, size)
```

### Parameters

`name` - The name of the font to use (if undefined, the standard 4x6 font will be used)

`size` - The size of the font (or undefined)

### Returns

The instance of Graphics this was called on, to allow call chaining

### Description

Set the font by name. Various forms are available:

- `g.setFont("4x6")` - standard 4x6 bitmap font
- `g.setFont("Vector:12")` - vector font 12px high
- `g.setFont("4x6:2")` - 4x6 bitmap font, doubled in size
- `g.setFont("6x8:2x3")` - 6x8 bitmap font, doubled in width, tripled in height

You can also use these forms, but they are not recommended:

- `g.setFont("Vector12")` - vector font 12px high
- `g.setFont("4x6",2)` - 4x6 bitmap font, doubled in size

`g.getFont()` will return the current font as a String.

For a list of available font names, you can use `g.getFonts()`.

**Note:** This is not available in devices with low flash memory

## [function Graphics.setFont12x20](#) ⇒

---

Call type:

[\(top\)](#)

```
function Graphics.setFont12x20(scale)
```

### Parameters

`scale` - [optional] If >1 the font will be scaled up by that amount

### Returns

The instance of Graphics this was called on, to allow call chaining

### Description

Set the current font

## [function Graphics.setFont6x15](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.setFont6x15(scale)
```

### Parameters

scale - [optional] If >1 the font will be scaled up by that amount

### Returns

The instance of Graphics this was called on, to allow call chaining

### Description

Set the current font

---

## [function Graphics.setFontAlign](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.setFontAlign(x, y, rotation)
```

### Parameters

x - X alignment. -1=left (default), 0=center, 1=right

y - Y alignment. -1=top (default), 0=center, 1=bottom

rotation - Rotation of the text. 0=normal, 1=90 degrees clockwise, 2=180, 3=270

### Returns

The instance of Graphics this was called on, to allow call chaining

### Description

Set the alignment for subsequent calls to `drawString`

**Note:** This is not available in devices with low flash memory

---

## [function Graphics.setFontBitmap](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.setFontBitmap()
```

### Returns

The instance of Graphics this was called on, to allow call chaining

### Description

Make subsequent calls to `drawString` use the built-in 4x6 pixel bitmapped Font

It is recommended that you use `Graphics.setFont("4x6")` for more flexibility.

---

## [function Graphics.setFontCustom](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.setFontCustom(bitmap, firstChar, width, height)
```

## Parameters

`bitmap` - A column-first, MSB-first, 1bpp bitmap containing the font bitmap

`firstChar` - The first character in the font - usually 32 (space)

`width` - The width of each character in the font. Either an integer, or a string where each character represents the width

`height` - The height as an integer (max 255). Bits 8-15 represent the scale factor (eg. `2<<8` is twice the size). Bits 16-23 represent the BPP (0,1=1 bpp, 2=2 bpp, 4=4 bpp)

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Make subsequent calls to `drawString` use a Custom Font of the given height. See the [Fonts page](#) for more information about custom fonts and how to create them.

For examples of use, see the [font modules](#).

**Note:** while you can specify the character code of the first character with `firstChar`, the newline character 13 will always be treated as a newline and not rendered.

**Note:** This is not available in devices with low flash memory

---

## [function Graphics.setFontVector](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.setFontVector(size)
```

## Parameters

`size` - The height of the font, as an integer

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Make subsequent calls to `drawString` use a Vector Font of the given height.

It is recommended that you use `Graphics.setFont("Vector", size)` for more flexibility.

**Note:** This is not available in devices with low flash memory

---

## [function Graphics.setPixel](#) ⇒

### Call type:

[\(top\)](#)

```
function Graphics.setPixel(x, y, col)
```

## Parameters

`x` - The left

`y` - The top

`col` - The color (if `undefined`, the foreground color is used)

## Returns

The instance of Graphics this was called on, to allow call chaining

## Description

Set a pixel's color

## [function Graphics.setRotation](#) ⇒

---

**Call type:**

```
function Graphics.setRotation(rotation, reflect)
```

[\(top\)](#)

### Parameters

`rotation` - The clockwise rotation. 0 for no rotation, 1 for 90 degrees, 2 for 180, 3 for 270

`reflect` - Whether to reflect the image

### Returns

The instance of Graphics this was called on, to allow call chaining

### Description

Set the current rotation of the graphics device.

## [function Graphics.setTheme](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.setTheme(theme)
```

### Parameters

`theme` - An object of the form returned by [Graphics.theme](#)

### Returns

The instance of Graphics this was called on, to allow call chaining

### Description

Set the global colour scheme. On Bangle.js, this is reloaded from `settings.json` for each new app loaded.

See [Graphics.theme](#) for the fields that can be provided. For instance you can change the background to red using:

```
g.setTheme({bg: "#f00"});
```

**Note:** This is not available in devices with low flash memory or 'Original' Espruino boards

## [function Graphics.stringMetrics](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.stringMetrics(str)
```

### Parameters

`str` - The string

### Returns

An object containing `{width, height}` of the string

### Description

Return the width and height in pixels of a string of text in the current font

## [function Graphics.stringWidth](#) ⇒

---

**Call type:**[\(top\)](#)

```
function Graphics.stringWidth(str)
```

**Parameters**

str - The string

**Returns**

The length of the string in pixels

**Description**

Return the size in pixels of a string of text in the current font

---

[property Graphics.theme](#) ⇒**Call type:**[\(top\)](#)

```
property Graphics.theme
```

**Returns**

An object containing the current 'theme' (see below)

**Description**

Returns an object of the form:

```
{
 fg : 0xFFFF, // foreground colour
 bg : 0, // background colour
 fg2 : 0xFFFF, // accented foreground colour
 bg2 : 0x0007, // accented background colour
 fgH : 0xFFFF, // highlighted foreground colour
 bgH : 0x02F7, // highlighted background colour
 dark : true, // Is background dark (e.g. foreground should be a light colour)
}
```

These values can then be passed to `g.setColor/g.setBgColor` for example `g.setColor(g.theme.fg2)`. When the Graphics instance is reset, the background color is automatically set to `g.theme.bg` and foreground is set to `g.theme.fg`.

On Bang.js these values can be changed by writing updated values to `theme` in `settings.js` and reloading the app - or they can be changed temporarily by calling [graphics.setTheme](#)

**Note:** This is not available in devices with low flash memory or 'Original' Espruino boards

---

[function Graphics.toColor](#) ⇒**Call type:**[\(top\)](#)

```
function Graphics.toColor(r, g, b)
```

**Parameters**

r - Red (between 0 and 1) **OR** an integer representing the color in the current bit depth and color order **OR** a hexidecimal color string of the form '#rrggbb' **or** '#rgb'

g - Green (between 0 and 1)

b - Blue (between 0 and 1)

**Returns**

The color index represented by the arguments

**Description**

Work out the color value to be used in the current bit depth based on the arguments.

This is used internally by `setColor` and `setBgColor`

```
// 1 bit
g.toColor(1,1,1) => 1
// 16 bit
g.toColor(1,0,0) => 0xF800
```

**Note:** This is not available in devices with low flash memory

## [function Graphics.transformVertices](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.transformVertices(verts, transformation)
```

**Parameters**

`verts` - An array of vertices, of the form `[x1,y1,x2,y2,x3,y3,etc]`

`transformation` - The transformation to apply, either an Object or an Array (see below)

**Returns**

Array of transformed vertices

**Description**

Transformation can be:

- An object of the form

```
{
 x: float, // x offset (default 0)
 y: float, // y offset (default 0)
 scale: float, // scale factor (default 1)
 rotate: float, // angle in radians (default 0)
}
```
- A six-element array of the form `[a,b,c,d,e,f]`, which represents the 2D transformation matrix

```
a c e
b d f
0 0 1
```

Apply a transformation to an array of vertices.

**Note:** This is not available in devices with low flash memory or 'Original' Espruino boards

## [function Graphics.wrapString](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Graphics.wrapString(str, maxWidth)
```

**Parameters**

`str` - The string

`maxWidth` - The width in pixels

**Returns**

An array of lines that are all less than `maxWidth`

**Description**

Wrap a string to the given pixel width using the current font, and return the lines as an array.

To render within the screen's width you can do:

```
g.drawString(g.wrapString(text, g.getWidth()).join("\n)),
```

---

## [heatshrink Library](#)

---

Simple library for compression/decompression using [heatshrink](#), an LZSS compression tool.

(top)

Espruino uses heatshrink internally to compress RAM down to fit in Flash memory when [save\(.\)](#) is used. This just exposes that functionality.

Functions here take and return buffers of data. There is no support for streaming, so both the compressed and decompressed data must be able to fit in memory at the same time.

If you'd like a way to perform compression/decompression on desktop, check out <https://github.com/espruino/EspruinoWebTools#heatshrinkjs>

### Methods and Fields

- [require\("heatshrink"\).compress\(data\)](#)
- [require\("heatshrink"\).decompress\(data\)](#)

---

#### [heatshrink.compress](#) ⇒

---

##### Call type:

(top)

```
require("heatshrink").compress(data)
```

##### Parameters

data - The data to compress

##### Returns

Returns the result as an ArrayBuffer

##### Description

Compress the heatshrink-encoded data supplied as input.

If you'd like a way to perform compression/decompression on desktop, check out <https://github.com/espruino/EspruinoWebTools#heatshrinkjs>

**Note:** This is not available in devices with low flash memory

---

#### [heatshrink.decompress](#) ⇒

---

##### Call type:

(top)

```
require("heatshrink").decompress(data)
```

##### Parameters

data - The data to decompress

##### Returns

Returns the result as an ArrayBuffer

##### Description

Decompress the heatshrink-encoded data supplied as input.

If you'd like a way to perform compression/decompression on desktop, check out <https://github.com/espruino/EspruinoWebTools#heatshrinkjs>

**Note:** This is not available in devices with low flash memory

## [http Library](#)

---

This library allows you to create http servers and make http requests

(top)

In order to use this, you will need an extra module to get network connectivity such as the [TI CC3000](#) or [WIZnet W5500](#).

This is designed to be a cut-down version of the [node.js library](#). Please see the [Internet](#) page for more information on how to use it.

### Methods and Fields

- [require\("http"\).createServer\(callback\)](#)
- [require\("http"\).get\(options, callback\)](#)
- [require\("http"\).request\(options, callback\)](#)

### [http.createServer](#) ⇒

---

#### Call type:

(top)

```
require("http").createServer(callback)
```

#### Parameters

callback - A function(request,response) that will be called when a connection is made

#### Returns

Returns a new httpSrv object

#### Description

Create an HTTP Server

When a request to the server is made, the callback is called. In the callback you can use the methods on the response ([httpsRs](#)) to send data. You can also add `request.on('data', function() { ... })` to listen for POSTed data

### [http.get](#) ⇒

---

#### Call type:

(top)

```
require("http").get(options, callback)
```

#### Parameters

options - A simple URL, or an object containing host,port,path,method fields

callback - A function(res) that will be called when a connection is made. You can then call `res.on('data', function(data) { ... })` and `res.on('close', function() { ... })` to deal with the response.

#### Returns

Returns a new httpCRq object

#### Description

Request a webpage over HTTP - a convenience function for [http.request\(\)](#), that makes sure the HTTP command is 'GET', and that calls `end` automatically.

```
require("http").get("http://pur3.co.uk/hello.txt", function(res) {
 res.on('data', function(data) {
 console.log("HTTP> "+data);
 });
 res.on('close', function(data) {
 console.log("Connection closed");
 });
});
```

See [http.request\(\)](#) and [the Internet page](#) and ` for more usage examples.

## [http.request](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("http").request(options, callback)
```

### Parameters

options - An object containing host,port,path,method,headers fields (and also ca,key,cert if HTTPS is enabled)

callback - A function(res) that will be called when a connection is made. You can then call res.on('data', **function**(data) { ... }) and res.on('close', **function**() { ... }) to deal with the response.

### Returns

Returns a new httpCRq object

### Description

Create an HTTP Request - **end()** must be called on it to complete the operation. options is of the form:

```
var options = {
 host: 'example.com', // host name
 port: 80, // (optional) port, defaults to 80
 path: '/', // path sent to server
 method: 'GET', // HTTP command sent to server (must be uppercase 'GET', 'POST', etc)
 protocol: 'http:', // optional protocol - https: or http:
 headers: { key : value, key : value } // (optional) HTTP headers
};
var req = require("http").request(options, function(res) {
 res.on('data', function(data) {
 console.log("HTTP> "+data);
 });
 res.on('close', function(data) {
 console.log("Connection closed");
 });
});
// You can reg.write(...) here if your request requires data to be sent.
req.end(); // called to finish the HTTP request and get the response
```

You can easily pre-populate options from a URL using

```
var options =
url.parse("http://www.example.com/foo.html")
```

There's an example of using [http.request](#) for HTTP POST here

**Note:** if TLS/HTTPS is enabled, options can have ca, key and cert fields. See [tls.connect](#) for more information about these and how to use them.

---

## [httpCRq Class](#)

---

The HTTP client request, returned by [http.request\(\)](#) and [http.get\(\)](#).

[\(top\)](#)

### Methods and Fields

- [event httpCRq.drain\(\)](#)
- [function httpCRq.end\(data\)](#)
- [event httpCRq.error\(\)](#)
- [function httpCRq.write\(data\)](#)

---

### [event httpCRq.drain](#) ⇒

---

#### Call type:

[\(top\)](#)

```
httpCRq.on('drain', function() { ... });
```

#### Description

An event that is fired when the buffer is empty and it can accept more data to send.

---

### [function httpCRq.end](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function httpCRq.end(data)
```

#### Parameters

`data` - A string containing data to send

#### Description

Finish this HTTP request - optional data to append as an argument

See [socket.write](#) for more information about the data argument

---

### [event httpCRq.error](#) ⇒

---

#### Call type:

[\(top\)](#)

```
httpCRq.on('error', function() { ... });
```

#### Description

An event that is fired if there is an error making the request and the response callback has not been invoked. In this case the error event concludes the request attempt. The error event function receives an error object as parameter with a `code` field and a `message` field.

---

### [function httpCRq.write](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function httpCRq.write(data)
```

#### Parameters

`data` - A string containing data to send

## Returns

For node.js compatibility, returns the boolean false. When the send buffer is empty, a `drain` event will be sent

## Description

This function writes the `data` argument as a string. Data that is passed in (including arrays) will be converted to a string with the normal JavaScript `toString` method. For more information about sending binary data see [`Socket.write`](#)

---

## [httpCRs Class](#)

---

The HTTP client response, passed to the callback of `http.request()`, an `http.get()`.

[\(top\)](#)

### Methods and Fields

- [function httpCRs.available\(\)](#)
- [event httpCRs.close\(\)](#)
- [event httpCRs.data\(data\)](#)
- [event httpCRs.error\(\)](#)
- [property httpCRs.headers](#)
- [property httpCRs.httpVersion](#)
- [function httpCRs.pipe\(destination, options\)](#)
- [function httpCRs.read\(chars\)](#)
- [property httpCRs.statusCode](#)
- [property httpCRs.statusMessage](#)

---

### [function httpCRs.available](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function httpCRs.available()
```

#### Returns

How many bytes are available

#### Description

Return how many bytes are available to read. If there is a 'data' event handler, this will always return 0.

---

### [event httpCRs.close](#) ⇒

---

#### Call type:

[\(top\)](#)

```
httpCRs.on('close', function() { ... });
```

#### Description

Called when the connection closes with one `hadError` boolean parameter, which indicates whether an error occurred.

---

### [event httpCRs.data](#) ⇒

---

#### Call type:

[\(top\)](#)

```
httpCRs.on('data', function(data) { ... });
```

#### Parameters

`data` - A string containing one or more characters of received data

#### Description

The 'data' event is called when data is received. If a handler is defined with `x.on('data', function(data) { ... })` then it will be called, otherwise data will be stored in an internal buffer, where it can be retrieved with `x.read()`

---

### [event httpCRs.error](#) ⇒

---

**Call type:**[\(top\)](#)

```
httpCRs.on('error', function() { ... });
```

**Description**

An event that is fired if there is an error receiving the response. The error event function receives an error object as parameter with a `code` field and a `message` field. After the error event the close even will also be triggered to conclude the HTTP request/response.

---

[\*\*property httpCRs.headers\*\*](#) ⇒**Call type:**[\(top\)](#)

```
property httpCRs.headers
```

**Returns**

An object mapping header name to value

**Description**

The headers received along with the HTTP response

---

[\*\*property httpCRs.httpVersion\*\*](#) ⇒**Call type:**[\(top\)](#)

```
property httpCRs.httpVersion
```

**Returns**

Th

**Description**

The HTTP version reported back by the server - usually "1.1"

---

[\*\*function httpCRs.pipe\*\*](#) ⇒**Call type:**[\(top\)](#)

```
function httpCRs.pipe(destination, options)
```

**Parameters**

`destination` - The destination file/stream that will receive content from the source.

`options` - [optional] An object { `chunkSize` : `int=32`, `end` : `bool=true`, `complete` : `function` }  
`chunkSize` : The amount of data to pipe from source to destination at a time  
`complete` : a function to call when the pipe activity is complete  
`end` : call the 'end' function on the destination when the source is finished

**Description**

Pipe this to a stream (an object with a 'write' method)

**Note:** This is not available in devices with low flash memory

---

[\*\*function httpCRs.read\*\*](#) ⇒**Call type:**[\(top\)](#)

```
function httpCRs.read(chars)
```

## Parameters

`chars` - The number of characters to read, or undefined/0 for all available

## Returns

A string containing the required bytes.

## Description

Return a string containing characters that have been received

---

## [property httpCRs.statusCode](#) ⇒

### Call type:

[\(top\)](#)

```
property httpCRs.statusCode
```

## Returns

The status code as a String

## Description

The HTTP response's status code - usually "200" if all went well

---

## [property httpCRs.statusMessage](#) ⇒

### Call type:

[\(top\)](#)

```
property httpCRs.statusMessage
```

## Returns

An String Status Message

## Description

The HTTP response's status message - Usually "OK" if all went well

---

## [httpSRq Class](#)

---

The HTTP server request

[\(top\)](#)

### Methods and Fields

- [function httpSRq.available\(\)](#)
- [event httpSRq.close\(\)](#)
- [event httpSRq.data\(data\)](#)
- [property httpSRq.headers](#)
- [property httpSRq.method](#)
- [function httpSRq.pipe\(destination, options\)](#)
- [function httpSRq.read\(chars\)](#)
- [property httpSRq.url](#)

---

### [function httpSRq.available](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function httpSRq.available()
```

#### Returns

How many bytes are available

#### Description

Return how many bytes are available to read. If there is already a listener for data, this will always return 0.

---

### [event httpSRq.close](#) ⇒

---

#### Call type:

[\(top\)](#)

```
httpSRq.on('close', function() { ... });
```

#### Description

Called when the connection closes.

---

### [event httpSRq.data](#) ⇒

---

#### Call type:

[\(top\)](#)

```
httpSRq.on('data', function(data) { ... });
```

#### Parameters

data - A string containing one or more characters of received data

#### Description

The 'data' event is called when data is received. If a handler is defined with `x.on('data', function(data) { ... })` then it will be called, otherwise data will be stored in an internal buffer, where it can be retrieved with `x.read()`

---

### [property httpSRq.headers](#) ⇒

---

#### Call type:

[\(top\)](#)

```
property httpSRq.headers
```

## Returns

An object mapping header name to value

## Description

The headers to sent to the server with this HTTP request.

---

## [property httpSRq.method](#) ⇒

### Call type:

[\(top\)](#)

```
property httpSRq.method
```

## Returns

A string

## Description

The HTTP method used with this request. Often "GET".

---

## [function httpSRq.pipe](#) ⇒

### Call type:

[\(top\)](#)

```
function httpSRq.pipe(destination, options)
```

## Parameters

destination - The destination file/stream that will receive content from the source.

options - [optional] An object { **chunkSize** : **int**=32, **end** : **bool**=**true**, **complete** : **function** }  
chunkSize : The amount of data to pipe from source to destination at a time  
complete : a function to call when the pipe activity is complete  
end : call the 'end' function on the destination when the source is finished

## Description

Pipe this to a stream (an object with a 'write' method)

**Note:** This is not available in devices with low flash memory

---

## [function httpSRq.read](#) ⇒

### Call type:

[\(top\)](#)

```
function httpSRq.read(chars)
```

## Parameters

chars - The number of characters to read, or undefined/0 for all available

## Returns

A string containing the required bytes.

## Description

Return a string containing characters that have been received

## [property httpSRq.url](#) ⇒

---

### Call type:

`property httpSRq.url`

### Returns

A string representing the URL

### Description

The URL requested in this HTTP request, for instance:

- `"/"` - the main page
- `"/favicon.ico"` - the web page's icon

[\(top\)](#)

---

## [httpSRs Class](#)

---

The HTTP server response

[\(top\)](#)

### Methods and Fields

- [event httpSRs.close\(\)](#)
- [event httpSRs.drain\(\)](#)
- [function httpSRs.end\(data\)](#)
- [property httpSRs.headers](#)
- [function httpSRs.setHeader\(name, value\)](#)
- [function httpSRs.write\(data\)](#)
- [function httpSRs.writeHead\(statusCode, headers\)](#)

---

### [event httpSRs.close](#) ⇒

---

#### Call type:

[\(top\)](#)

```
httpSRs.on('close', function() { ... });
```

#### Description

Called when the connection closes.

---

### [event httpSRs.drain](#) ⇒

---

#### Call type:

[\(top\)](#)

```
httpSRs.on('drain', function() { ... });
```

#### Description

An event that is fired when the buffer is empty and it can accept more data to send.

---

### [function httpSRs.end](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function httpSRs.end(data)
```

#### Parameters

data - A string containing data to send

#### Description

See [socket.write](#) for more information about the data argument

---

### [property httpSRs.headers](#) ⇒

---

#### Call type:

[\(top\)](#)

```
property httpSRs.headers
```

#### Returns

An object mapping header name to value

## Description

The headers to send back along with the HTTP response.

The default contents are:

```
{
 "Connection": "close"
}
```

---

## [function httpSRs.setHeader](#) ⇒

### Call type:

[\(top\)](#)

```
function httpSRs.setHeader(name, value)
```

### Parameters

`name` - The name of the header as a String

`value` - The value of the header as a String

## Description

Set a value to send in the header of this HTTP response. This updates the [httpSRs.headers](#) property.

Any headers supplied to `writeHead` will overwrite any headers with the same name.

---

## [function httpSRs.write](#) ⇒

### Call type:

[\(top\)](#)

```
function httpSRs.write(data)
```

### Parameters

`data` - A string containing data to send

### Returns

For node.js compatibility, returns the boolean false. When the send buffer is empty, a `drain` event will be sent

## Description

This function writes the `data` argument as a string. Data that is passed in (including arrays) will be converted to a string with the normal JavaScript `toString` method. For more information about sending binary data see [Socket.write](#)

---

## [function httpSRs.writeHead](#) ⇒

### Call type:

[\(top\)](#)

```
function httpSRs.writeHead(statusCode, headers)
```

### Parameters

`statusCode` - The HTTP status code

`headers` - An object containing the headers

## Description

Send the given status code and headers. If not explicitly called this will be done automatically the first time data is written to the response.

This cannot be called twice, or after data has already been sent in the response.

---

## [httpSrv Class](#)

---

The HTTP server created by `require('http').createServer`

[\(top\)](#)

### Methods and Fields

- [function httpSrv.close\(\)](#)
- [function httpSrv.listen\(port\)](#)

---

### [function httpSrv.close](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function httpSrv.close()
```

#### Description

Stop listening for new HTTP connections

---

### [function httpSrv.listen](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function httpSrv.listen(port)
```

#### Parameters

`port` - The port to listen on

#### Returns

The HTTP server instance that 'listen' was called on

#### Description

Start listening for new HTTP connections on the given port

---

## [I2C Class](#)

---

This class allows use of the built-in I2C ports. Currently it allows I2C Master mode only.

[\(top\)](#)

All addresses are in 7 bit format. If you have an 8 bit address then you need to shift it one bit to the right.

### Instances

- [I2C1](#) The first I2C port
- [I2C2](#) The second I2C port
- [I2C3](#) The third I2C port

### Methods and Fields

- [I2C.find\(pin\)](#)
- [constructor I2C\(\)](#)
- [function I2C.readFrom\(address, quantity\)](#)
- [function I2C.setup\(options\)](#)
- [function I2C.writeTo\(address, data, ...\)](#)

---

### [I2C.find](#) ⇒

---

#### Call type:

[\(top\)](#)

```
I2C.find(pin)
```

#### Parameters

`pin` - A pin to search with

#### Returns

An object of type [I2C](#), or [undefined](#) if one couldn't be found.

#### Description

Try and find an I2C hardware device that will work on this pin (e.g. [I2C1](#))

May return undefined if no device can be found.

---

### [constructor I2C](#) ⇒

---

#### Call type:

[\(top\)](#)

```
new I2C()
```

#### Returns

An I2C object

#### Description

Create a software I2C port. This has limited functionality (no baud rate), but it can work on any pins.

Use [I2C.setup](#) to configure this port.

---

### [function I2C.readFrom](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function I2C.readFrom(address, quantity)
```

## Parameters

`address` - The 7 bit address of the device to request bytes from, or an object of the form `{address:12, stop:false}` to send this data without a STOP signal.

`quantity` - The number of bytes to request

## Returns

The data that was returned - as a Uint8Array

## Description

Request bytes from the given slave device, and return them as a Uint8Array (packed array of bytes). This is like using Arduino Wire's `requestFrom`, available and `read` functions. Sends a STOP

---

## [function I2C.setup](#) ⇒

### Call type:

[\(top\)](#)

```
function I2C.setup(options)
```

## Parameters

`options` - [optional] A structure containing extra information on initialising the I2C port  
`{scl:pin, sda:pin, bitrate:100000}`

You can find out which pins to use by looking at [your board's reference page](#) and searching for pins with the `i2c` marker. Note that 400kHz is the maximum bitrate for most parts.

## Description

Set up this I2C port

If not specified in options, the default pins are used (usually the lowest numbered pins on the lowest port that supports this peripheral)

---

## [function I2C.writeTo](#) ⇒

### Call type:

[\(top\)](#)

```
function I2C.writeTo(address, data, ...)
```

## Parameters

`address` - The 7 bit address of the device to transmit to, or an object of the form `{address:12, stop:false}` to send this data without a STOP signal.

`data, ...` - One or more items to write. May be ints, strings, arrays, or special objects (see [E.toInt8Array](#) for more info).

## Description

Transmit to the slave device with the given address. This is like Arduino's `beginTransmission`, `write`, and `endTransmission` rolled up into one.

---

## [Int16Array Class](#)

---

This is the built-in JavaScript class for a typed array of 16 bit signed integers.

[\(top\)](#)

Instantiate this in order to efficiently store arrays of data (Espruino's normal arrays store data in a map, which is inefficient for non-sparse arrays).

Arrays of this type include all the methods from [ArrayBufferView](#)

### Methods and Fields

- [constructor Int16Array\(arr,byteOffset,length\)](#)

---

### [constructor Int16Array](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

```
new Int16Array(arr, byteOffset, length)
```

#### Parameters

`arr` - The array or typed array to base this off, or an integer which is the array length

`byteOffset` - The byte offset in the ArrayBuffer (ONLY IF the first argument was an ArrayBuffer)

`length` - The length (ONLY IF the first argument was an ArrayBuffer)

#### Returns

A typed array

#### Description

Create a typed array based on the given input. Either an existing ArrayBuffer, an Integer as a Length, or a simple array. If an [ArrayBufferView](#) (e.g. [Uint8Array](#), rather than [ArrayBuffer](#)) is given, it will be completely copied rather than referenced.

---

## [Int32Array Class](#)

---

This is the built-in JavaScript class for a typed array of 32 bit signed integers.

(top)

Instantiate this in order to efficiently store arrays of data (Espruino's normal arrays store data in a map, which is inefficient for non-sparse arrays).

Arrays of this type include all the methods from [ArrayBufferView](#)

### Methods and Fields

- [constructor Int32Array\(arr,byteOffset,length\)](#)

---

### [constructor Int32Array](#) ⇒

---

[View MDN documentation](#)

#### Call type:

(top)

```
new Int32Array(arr, byteOffset, length)
```

#### Parameters

`arr` - The array or typed array to base this off, or an integer which is the array length

`byteOffset` - The byte offset in the ArrayBuffer (ONLY IF the first argument was an ArrayBuffer)

`length` - The length (ONLY IF the first argument was an ArrayBuffer)

#### Returns

A typed array

#### Description

Create a typed array based on the given input. Either an existing ArrayBuffer, an Integer as a Length, or a simple array. If an [ArrayBufferView](#) (e.g. [Uint8Array](#), rather than [ArrayBuffer](#)) is given, it will be completely copied rather than referenced.

## [Int8Array Class](#)

---

This is the built-in JavaScript class for a typed array of 8 bit signed integers.

[\(top\)](#)

Instantiate this in order to efficiently store arrays of data (Espruino's normal arrays store data in a map, which is inefficient for non-sparse arrays).

Arrays of this type include all the methods from [ArrayBufferView](#)

### Methods and Fields

- [constructor Int8Array\(arr, byteOffset, length\)](#)

### [constructor Int8Array](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

```
new Int8Array(arr, byteOffset, length)
```

#### Parameters

`arr` - The array or typed array to base this off, or an integer which is the array length

`byteOffset` - The byte offset in the ArrayBuffer (ONLY IF the first argument was an ArrayBuffer)

`length` - The length (ONLY IF the first argument was an ArrayBuffer)

#### Returns

A typed array

#### Description

Create a typed array based on the given input. Either an existing ArrayBuffer, an Integer as a Length, or a simple array. If an [ArrayBufferView](#) (e.g. [Uint8Array](#), rather than [ArrayBuffer](#)) is given, it will be completely copied rather than referenced.

---

## [InternalError Class](#)

---

The base class for internal errors

[\(top\)](#)

### Methods and Fields

- [constructor InternalError\(message\)](#)
- [function InternalError.toString\(\)](#)

---

### [constructor InternalError](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

```
new InternalError(message)
```

#### Parameters

message - [optional] An message string

#### Returns

An InternalError object

#### Description

Creates an InternalError object

---

### [function InternalError.toString](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function InternalError.toString()
```

#### Returns

A String

---

## [JSON Class](#)

---

An Object that handles conversion to and from the JSON data interchange format

[\(top\)](#)

### Methods and Fields

- [JSON.parse\(string\)](#)
- [JSON.stringify\(data, replacer, space\)](#)

#### [JSON.parse](#) ⇒

---

[View MDN documentation](#)

##### Call type:

`JSON.parse(string)`

##### Parameters

`string` - A JSON string

##### Returns

The JavaScript object created by parsing the data string

##### Description

Parse the given JSON string into a JavaScript object

NOTE: This implementation uses eval() internally, and as such it is unsafe as it can allow arbitrary JS commands to be executed.

#### [JSON.stringify](#) ⇒

---

[View MDN documentation](#)

##### Call type:

`JSON.stringify(data, replacer, space)`

##### Parameters

`data` - The data to be converted to a JSON string

`replacer` - This value is ignored

`space` - The number of spaces to use for padding, a string, or null/undefined for no whitespace

##### Returns

A JSON string

##### Description

Convert the given object into a JSON string which can subsequently be parsed with JSON.parse or eval.

**Note:** This differs from JavaScript's standard [JSON.stringify](#) in that:

- The `replacer` argument is ignored
- Typed arrays like `new Uint8Array(5)` will be dumped as if they were arrays, not as if they were objects (since it is more compact)

---

## [Math Class](#)

---

This is a standard JavaScript class that contains useful Maths routines

[\(top\)](#)

### Methods and Fields

- [Math.abs\(x\)](#)
- [Math.acos\(x\)](#)
- [Math.asin\(x\)](#)
- [Math.atan\(x\)](#)
- [Math.atan2\(y, x\)](#)
- [Math.ceil\(x\)](#)
- [Math.clip\(x, min, max\)](#)
- [Math.cos\(theta\)](#)
- [Math.E](#)
- [Math.exp\(x\)](#)
- [Math.floor\(x\)](#)
- [Math.LN10](#)
- [Math.LN2](#)
- [Math.log\(x\)](#)
- [Math.LOG10E](#)
- [Math.LOG2E](#)
- [Math.max\(args, ...\)](#)
- [Math.min\(args, ...\)](#)
- [Math.PI](#)
- [Math.pow\(x, y\)](#)
- [Math.random\(\)](#)
- [Math.round\(x\)](#)
- [Math.sign\(x\)](#)
- [Math.sin\(theta\)](#)
- [Math.sqrt\(x\)](#)
- [Math.SQRT1\\_2](#)
- [Math.SQRT2](#)
- [Math.tan\(theta\)](#)
- [Math.wrap\(x, max\)](#)

---

### [Math.abs](#) =>

---

[View MDN documentation](#)

**Call type:**

[\(top\)](#)

`Math.abs(x)`

**Parameters**

x - A floating point value

**Returns**

The absolute value of x (eg, `Math.abs(2)==2`, but also `Math.abs(-2)==2`)

---

### [Math.acos](#) =>

---

[View MDN documentation](#)

**Call type:**

[\(top\)](#)

`Math.acos(x)`

**Parameters**

x - The value to get the arc cosine of

**Returns**

The arc cosine of x, between 0 and PI

## [Math.asin](#) ⇒

---

[View MDN documentation](#)

**Call type:**

`Math.asin(x)`

**Parameters**

x - The value to get the arc sine of

**Returns**

The arc sine of x, between -PI/2 and PI/2

## [Math.atan](#) ⇒

---

[View MDN documentation](#)

**Call type:**

[\(top\)](#)

`Math.atan(x)`

**Parameters**

x - The value to get the arc tangent of

**Returns**

The arc tangent of x, between -PI/2 and PI/2

## [Math.atan2](#) ⇒

---

[View MDN documentation](#)

**Call type:**

[\(top\)](#)

`Math.atan2(y, x)`

**Parameters**

y - The Y-part of the angle to get the arc tangent of

x - The X-part of the angle to get the arc tangent of

**Returns**

The arctangent of Y/X, between -PI and PI

## [Math.ceil](#) ⇒

---

[View MDN documentation](#)

**Call type:**

[\(top\)](#)

`Math.ceil(x)`

**Parameters**

x - The value to round up

## Returns

x, rounded upwards to the nearest integer

## [Math.clip](#) ⇒

---

### Call type:

[\(top\)](#)

```
Math.clip(x, min, max)
```

### Parameters

x - A floating point value to clip

min - The smallest the value should be

max - The largest the value should be

### Returns

The value of x, clipped so as not to be below min or above max.

### Description

DEPRECATED - Please use [`E.clip\(\)`](#) instead. Clip a number to be between min and max (inclusive)

**Note:** This is not available in devices with low flash memory

## [Math.cos](#) ⇒

---

[View MDN documentation](#)

### Call type:

[\(top\)](#)

```
Math.cos(theta)
```

### Parameters

theta - The angle to get the cosine of

### Returns

The cosine of theta

## [Math.E](#) ⇒

---

[View MDN documentation](#)

### Call type:

[\(top\)](#)

```
Math.E
```

### Returns

The value of E - 2.718281828459045

## [Math.exp](#) ⇒

---

[View MDN documentation](#)

### Call type:

[\(top\)](#)

```
Math.exp(x)
```

## Parameters

x - The value raise E to the power of

## Returns

$E^x$

---

## [Math.floor](#) ⇒

[View MDN documentation](#)

## Call type:

`Math.floor(x)`

## Parameters

x - The value to round down

## Returns

x, rounded downwards to the nearest integer

---

## [Math.LN10](#) ⇒

[View MDN documentation](#)

## Call type:

`Math.LN10`

## Returns

The natural logarithm of 10 - 2.302585092994046

---

## [Math.LN2](#) ⇒

[View MDN documentation](#)

## Call type:

`Math.LN2`

## Returns

The natural logarithm of 2 - 0.6931471805599453

---

## [Math.log](#) ⇒

[View MDN documentation](#)

## Call type:

`Math.log(x)`

## Parameters

x - The value to take the logarithm (base E) root of

## Returns

[\(top\)](#)

The log (base E) of x

## [Math.LOG10E](#) ⇒

---

[View MDN documentation](#)

**Call type:**

`Math.LOG10E`

**Returns**

The base 10 logarithm of e - 0.4342944819032518

## [Math.LOG2E](#) ⇒

---

[View MDN documentation](#)

**Call type:**

`Math.LOG2E`

**Returns**

The base 2 logarithm of e - 1.4426950408889634

## [Math.max](#) ⇒

---

[View MDN documentation](#)

**Call type:**

`Math.max(args, ...)`

**Parameters**

`args, ...` - Floating point values to clip

**Returns**

The maximum of the supplied values

**Description**

Find the maximum of a series of numbers

## [Math.min](#) ⇒

---

[View MDN documentation](#)

**Call type:**

`Math.min(args, ...)`

**Parameters**

`args, ...` - Floating point values to clip

**Returns**

The minimum of the supplied values

[\(top\)](#)

## Description

Find the minimum of a series of numbers

### [Math.PI](#) ⇒

---

[View MDN documentation](#)

#### Call type:

`Math.PI`

#### Returns

The value of PI - 3.141592653589793

### [Math.pow](#) ⇒

---

[View MDN documentation](#)

#### Call type:

([top](#))

`Math.pow(x, y)`

#### Parameters

x - The value to raise to the power

y - The power x should be raised to

#### Returns

x raised to the power y ( $x^y$ )

### [Math.random](#) ⇒

---

[View MDN documentation](#)

#### Call type:

([top](#))

`Math.random()`

#### Returns

A random number between 0 and 1

### [Math.round](#) ⇒

---

[View MDN documentation](#)

#### Call type:

([top](#))

`Math.round(x)`

#### Parameters

x - The value to round

#### Returns

x, rounded to the nearest integer

## [Math.sign](#) ⇒

---

[View MDN documentation](#)

**Call type:**

`Math.sign(x)`

**Parameters**

`x` - The value to get the sign from

**Returns**

sign on `x` - -1, 1, or 0

## [Math.sin](#) ⇒

---

[View MDN documentation](#)

**Call type:**

`Math.sin(theta)`

**Parameters**

`theta` - The angle to get the sine of

**Returns**

The sine of `theta`

## [Math.sqrt](#) ⇒

---

[View MDN documentation](#)

**Call type:**

`Math.sqrt(x)`

**Parameters**

`x` - The value to take the square root of

**Returns**

The square root of `x`

## [Math.SQRT1\\_2](#) ⇒

---

[View MDN documentation](#)

**Call type:**

`Math.SQRT1_2`

**Returns**

The square root of  $1/2$  - 0.7071067811865476

## [Math.SQRT2](#) ⇒

---

(top)

[View MDN documentation](#)

## Call type:

`Math.SQRT2`

## Returns

The square root of 2 - 1.4142135623730951

---

## [Math.tan](#) ⇒

[View MDN documentation](#)

## Call type:

`Math.tan(theta)`

## Parameters

`theta` - The angle to get the tangent of

## Returns

The tangent of theta

---

## [Math.wrap](#) ⇒

## Call type:

`Math.wrap(x, max)`

## Parameters

`x` - A floating point value to wrap

`max` - The largest the value should be

## Returns

The value of `x`, wrapped so as not to be below min or above max.

## Description

DEPRECATED - This is not part of standard JavaScript libraries

Wrap a number around if it is less than 0 or greater than or equal to max. For instance you might do: `Math.wrap(angleInDegrees, 360)`

**Note:** This is not available in devices with low flash memory

(top)

---

## [Microbit Class](#)

---

Class containing [micro:bit's](#) utility functions.

[\(top\)](#)

### Methods and Fields

- [Microbit.accel\(\)](#)
- [Microbit.accelOff\(\)](#)
- [Microbit.accelOn\(\)](#)
- [Microbit.accelWr\(addr, data\)](#)
- [event Microbit.gesture\(gesture\)](#)
- [Microbit.mag\(\)](#)
- [Microbit.MIC](#)
- [Microbit.MIC\\_ENABLE](#)
- [Microbit.play\(waveform, samplesPerSecond, callback\)](#)
- [Microbit.record\(samplesPerSecond, callback, samples\)](#)
- [Microbit.SPEAKER](#)

---

### [Microbit.accel](#) ⇒

---

Call type:

[\(top\)](#)

`Microbit.accel()`

#### Returns

An Object {x,y,z} of acceleration readings in G

---

### [Microbit.accelOff](#) ⇒

---

Call type:

[\(top\)](#)

`Microbit.accelOff()`

#### Description

Turn off events from the accelerometer (started with [Microbit.accelOn](#))

**Note:** This is only available in BBC micro:bit v2 boards

---

### [Microbit.accelOn](#) ⇒

---

Call type:

[\(top\)](#)

`Microbit.accelOn()`

#### Description

Turn on the accelerometer, and create [Microbit.accel](#) and [Microbit.gesture](#) events.

**Note:** The accelerometer is currently always enabled - this code just responds to interrupts and reads

**Note:** This is only available in BBC micro:bit v2 boards

---

### [Microbit.accelWr](#) ⇒

---

Call type:

[\(top\)](#)

`Microbit.accelWr(addr, data)`

## Parameters

addr - Accelerometer address

data - Data to write

## Description

**Note:** This function is only available on the [BBC micro:bit](#) board

Write the given value to the accelerometer

**Note:** This is only available in BBC micro:bit v2 boards

---

## [event Microbit.gesture](#) ⇒

### Call type:

[\(top\)](#)

```
Microbit.on('gesture', function(gesture) { ... });
```

## Parameters

gesture - An Int8Array containing the accelerations (X,Y,Z) from the last gesture detected by the accelerometer

## Description

Called when the Micro:bit is moved in a deliberate fashion, and includes data on the detected gesture.

**Note:** This is only available in BBC micro:bit v2 boards

---

## [Microbit.mag](#) ⇒

### Call type:

[\(top\)](#)

```
Microbit.mag()
```

## Returns

An Object {x,y,z} of magnetometer readings as integers

---

## [Microbit.MIC](#) ⇒

### Call type:

[\(top\)](#)

```
Microbit.MIC
```

## Returns

See description above

## Description

The micro:bit's microphone pin

`MIC_ENABLE` should be set to 1 before using this

**Note:** This is only available in BBC micro:bit v2 boards

---

## [Microbit.MIC\\_ENABLE](#) ⇒

### Call type:

[\(top\)](#)

```
Microbit.MIC_ENABLE
```

## Returns

See description above

## Description

The micro:bit's microphone enable pin

**Note:** This is only available in BBC micro:bit v2 boards

---

## [Microbit.play](#) ⇒

### Call type:

[\(top\)](#)

```
Microbit.play(waveform, samplesPerSecond, callback)
```

### Parameters

`waveform` - An array of data to play (unsigned 8 bit)

`samplesPerSecond` - The number of samples per second for playback default is 4000

`callback` - A function to call when playback is finished

### Description

Play a waveform on the Micro:bit's speaker

**Note:** This is only available in BBC micro:bit v2 boards

---

## [Microbit.record](#) ⇒

### Call type:

[\(top\)](#)

```
Microbit.record(samplesPerSecond, callback, samples)
```

### Parameters

`samplesPerSecond` - The number of samples per second for recording - 4000 is recommended

`callback` - A function to call with the result of recording (unsigned 8 bit ArrayBuffer)

`samples` - [optional] How many samples to record (6000 default)

### Description

Records sound from the micro:bit's onboard microphone and returns the result

**Note:** This is only available in BBC micro:bit v2 boards

---

## [Microbit.SPEAKER](#) ⇒

### Call type:

[\(top\)](#)

```
Microbit.SPEAKER
```

## Returns

See description above

## Description

The micro:bit's speaker pin

**Note:** This is only available in BBC micro:bit v2 boards

---

## [Modules Class](#)

---

Built-in class that caches the modules used by the [require](#) command

[\(top\)](#)

### Methods and Fields

- [Modules.addCached\(id, sourcecode\)](#)
- [Modules.getCached\(\)](#)
- [Modules.removeAllCached\(\)](#)
- [Modules.removeCached\(id\)](#)

---

### [Modules.addCached](#) ⇒

---

#### Call type:

[\(top\)](#)

`Modules.addCached(id, sourcecode)`

#### Parameters

`id` - The module name to add

`sourcecode` - The module's sourcecode

#### Description

Add the given module to the cache

---

### [Modules.getCached](#) ⇒

---

#### Call type:

[\(top\)](#)

`Modules.getCached()`

#### Returns

An array of module names

#### Description

Return an array of module names that have been cached

---

### [Modules.removeAllCached](#) ⇒

---

#### Call type:

[\(top\)](#)

`Modules.removeAllCached()`

#### Description

Remove all cached modules

---

### [Modules.removeCached](#) ⇒

---

#### Call type:

[\(top\)](#)

`Modules.removeCached(id)`

#### Parameters

`id` - The module name to remove

### Description

Remove the given module from the list of cached modules

## [neopixel Library](#)

This library allows you to write to Neopixel/WS281x/APA10x/SK6812 LED strips

[\(top\)](#)

These use a high speed single-wire protocol which needs platform-specific implementation on some devices - hence this library to simplify things.

### Methods and Fields

- `require("neopixel").write(pin, data)`

#### [neopixel.write](#) ⇒

##### Call type:

[\(top\)](#)

```
require("neopixel").write(pin, data)
```

##### Parameters

`pin` - The Pin the LEDs are connected to

`data` - The data to write to the LED strip (must be a multiple of 3 bytes long)

##### Description

Write to a strip of NeoPixel/WS281x/APA104/APA106/SK6812-style LEDs attached to the given pin.

```
// set just one pixel, red, green, blue
require("neopixel").write(B15, [255,0,0]);

// Produce an animated rainbow over 25 LEDs
var rgb = new Uint8ClampedArray(25*3);
var pos = 0;
function getPattern() {
 pos++;
 for (var i=0;i<rgb.length;) {
 rgb[i++] = (1 + Math.sin((i+pos)*0.1324)) * 127;
 rgb[i++] = (1 + Math.sin((i+pos)*0.1654)) * 127;
 rgb[i++] = (1 + Math.sin((i+pos)*0.1)) * 127;
 }
 return rgb;
}
setInterval(function() {
 require("neopixel").write(B15, getPattern());
}, 100);
```

##### Note:

- 

Different types of LED have the data in different orders - so don't be surprised by RGB or BGR orderings!

- 

Some LED strips (SK6812) actually take 4 bytes per LED (red, green, blue and white). These are still supported but the array of data supplied must still be a multiple of 3 bytes long. Just round the size up - it won't cause any problems.

- 

On some platforms like STM32, pins capable of hardware SPI MOSI are required.

- 

On STM32, `neopixel.write` chooses a hardware SPI device to output the signal on and uses that. However in order to avoid spikes in the output, if that hardware device is *already initialised* it will not be re-initialised. This means that if the SPI device was already in use, you may have to use `SPIx.setup({baud:3200000, mosi:the_pin})` to force it to be re-setup on the pin.

- 

Espruino devices tend to have 3.3v IO, while WS2812/etc run off of 5v. Many WS2812 will only register a logic '1' at 70% of their input voltage - so if powering them off 5v you will not be able to send them data reliably. You can work around this by powering the LEDs off a lower voltage (for example 3.7v from a LiPo battery), can put the output into the `af.opendrain` state and use a pullup resistor to 5v on STM32 based boards (nRF52 are not 5v tolerant so you can't do this), or can use a level shifter to shift the voltage up into the 5v range.

---

## [net Library](#)

---

This library allows you to create TCPIP servers and clients

[\(top\)](#)

In order to use this, you will need an extra module to get network connectivity.

This is designed to be a cut-down version of the [node.js library](#). Please see the [Internet](#) page for more information on how to use it.

### Methods and Fields

- [`require\("net"\).connect\(options, callback\)`](#)
- [`require\("net"\).createServer\(callback\)`](#)

---

### [net.connect](#) ⇒

---

#### Call type:

[\(top\)](#)

```
require("net").connect(options, callback)
```

#### Parameters

`options` - An object containing host,port fields

`callback` - A `function(sckt)` that will be called with the socket when a connection is made. You can then call `sckt.write(...)` to send data, and `sckt.on('data', function(data) { ... })` and `sckt.on('close', function() { ... })` to deal with the response.

#### Returns

Returns a new `net.Socket` object

#### Description

Create a TCP socket connection

---

### [net.createServer](#) ⇒

---

#### Call type:

[\(top\)](#)

```
require("net").createServer(callback)
```

#### Parameters

`callback` - A `function(connection)` that will be called when a connection is made

#### Returns

Returns a new Server Object

#### Description

Create a Server

When a request to the server is made, the callback is called. In the callback you can use the methods on the connection to send data. You can also add `connection.on('data',function() { ... })` to listen for received data

---

## [NetworkJS Library](#)

---

Library that initialises a network device that calls into JavaScript

[\(top\)](#)

### Methods and Fields

- [require\("NetworkJS"\).create\(obj\)](#)

---

### [NetworkJS.create](#) ⇒

---

#### Call type:

[\(top\)](#)

```
require("NetworkJS").create(obj)
```

#### Parameters

obj - An object containing functions to access the network device

#### Returns

The object passed in

#### Description

Initialise the network using the callbacks given and return the first argument. For instance:

```
require("NetworkJS").create({
 create : function(host, port, socketType, options) {
 // Create a socket and return its index, host is a string, port is an integer.
 // If host isn't defined, create a server socket
 console.log("Create",host,port);
 return 1;
 },
 close : function(sckt) {
 // Close the socket. returns nothing
 },
 accept : function(sckt) {
 // Accept the connection on the server socket. Returns socket number or -1 if no connection
 return -1;
 },
 recv : function(sckt, maxLen, socketType) {
 // Receive data. Returns a string (even if empty).
 // If non-string returned, socket is then closed
 return null;//or "";
 },
 send : function(sckt, data, socketType) {
 // Send data (as string). Returns the number of bytes sent - 0 is ok.
 // Less than 0
 return data.length;
 }
});
```

socketType is an integer - 2 for UDP, or see SocketType in <https://github.com/espruino/Espruino/blob/master/libs/network/network.h> for more information.

---

## [NodeMCU Class](#)

---

This is a built-in class to allow you to use the ESP8266 NodeMCU boards' pin namings to access pins. It is only available on ESP8266-based boards.

[\(top\)](#)

### Methods and Fields

- [NodeMCU.A0](#)
- [NodeMCU.D0](#)
- [NodeMCU.D1](#)
- [NodeMCU.D10](#)
- [NodeMCU.D2](#)
- [NodeMCU.D3](#)
- [NodeMCU.D4](#)
- [NodeMCU.D5](#)
- [NodeMCU.D6](#)
- [NodeMCU.D7](#)
- [NodeMCU.D8](#)
- [NodeMCU.D9](#)

---

### [NodeMCU.A0](#) ⇒

---

#### Call type:

[\(top\)](#)

`NodeMCU.A0`

#### Returns

A Pin

---

### [NodeMCU.D0](#) ⇒

---

#### Call type:

[\(top\)](#)

`NodeMCU.D0`

#### Returns

A Pin

---

### [NodeMCU.D1](#) ⇒

---

#### Call type:

[\(top\)](#)

`NodeMCU.D1`

#### Returns

A Pin

---

### [NodeMCU.D10](#) ⇒

---

#### Call type:

[\(top\)](#)

`NodeMCU.D10`

#### Returns

A Pin

## [NodeMCU.D2](#) ⇒

---

**Call type:**

`NodeMCU.D2`

**Returns**

A Pin

[\(top\)](#)

## [NodeMCU.D3](#) ⇒

---

**Call type:**

[\(top\)](#)

`NodeMCU.D3`

**Returns**

A Pin

[\(top\)](#)

## [NodeMCU.D4](#) ⇒

---

**Call type:**

[\(top\)](#)

`NodeMCU.D4`

**Returns**

A Pin

[\(top\)](#)

## [NodeMCU.D5](#) ⇒

---

**Call type:**

[\(top\)](#)

`NodeMCU.D5`

**Returns**

A Pin

[\(top\)](#)

## [NodeMCU.D6](#) ⇒

---

**Call type:**

[\(top\)](#)

`NodeMCU.D6`

**Returns**

A Pin

[\(top\)](#)

## [NodeMCU.D7](#) ⇒

---

**Call type:**

[\(top\)](#)

`NodeMCU.D7`

**Returns**

A Pin

[\(top\)](#)

## [NodeMCU.D8](#) ⇒

---

**Call type:**

`NodeMCU.D8`

**Returns**

A Pin

[\(top\)](#)

## [NodeMCU.D9](#) ⇒

---

**Call type:**

[\(top\)](#)

`NodeMCU.D9`

**Returns**

A Pin

---

## [NRF Class](#)

---

The NRF class is for controlling functionality of the Nordic nRF51/nRF52 chips.

[\(top\)](#)

Most functionality is related to Bluetooth Low Energy, however there are also some functions related to NFC that apply to NRF52-based devices.

### Methods and Fields

- [event NRF.advertising\(isAdvertising\)](#)
- [NRF.amsCommand\(id\)](#)
- [NRF.amsGetPlayerInfo\(id\)](#)
- [NRF.amsGetTrackInfo\(id\)](#)
- [NRF.amsIsActive\(\)](#)
- [NRF.ancsAction\(uid, positive\)](#)
- [NRF.ancsGetAppInfo\(id\)](#)
- [NRF.ancsGetNotificationInfo\(uid\)](#)
- [NRF.ancsIsActive\(\)](#)
- [event NRF.bond\(status\)](#)
- [event NRF.characteristicsDiscover\(\)](#)
- [event NRF.connect\(addr\)](#)
- [NRF.connect\(mac, options\)](#)
- [event NRF.disconnect\(reason\)](#)
- [NRF.disconnect\(\)](#)
- [NRF.filterDevices\(devices, filters\)](#)
- [NRF.findDevices\(callback, options\)](#)
- [NRF.getAddress\(\)](#)
- [NRF.getAdvertisingData\(data, options\)](#)
- [NRF.getBattery\(\)](#)
- [NRF.getSecurityStatus\(\)](#)
- [event NRF.HID\(\)](#)
- [NRF.nfcAndroidApp\(app\)](#)
- [event NRF.NFCoff\(\)](#)
- [event NRF.NFCon\(\)](#)
- [NRF.nfcPair\(key\)](#)
- [NRF.nfcRaw\(payload\)](#)
- [event NRF.NFCrx\(arr\)](#)
- [NRF.nfcSend\(payload\)](#)
- [NRF.nfcStart\(payload\)](#)
- [NRF.nfcStop\(\)](#)
- [NRF.nfcURL\(url\)](#)
- [NRF.requestDevice\(options\)](#)
- [NRF.restart\(callback\)](#)
- [event NRF.security\(status\)](#)
- [NRF.sendHIDReport\(data, callback\)](#)
- [event NRF.servicesDiscover\(\)](#)
- [NRF.setAddress\(addr\)](#)
- [NRF.setAdvertising\(data, options\)](#)
- [NRF.setConnectionInterval\(interval\)](#)
- [NRF.setLowPowerConnection\(lowPower\)](#)
- [NRF.setRSSIHandler\(callback\)](#)
- [NRF.setScan\(callback, options\)](#)
- [NRF.setScanResponse\(data\)](#)
- [NRF.setSecurity\(options\)](#)
- [NRF.setServices\(data, options\)](#)
- [NRF.setTxPower\(power\)](#)
- [NRF.setWhitelist\(whitelisting\)](#)
- [NRF.sleep\(\)](#)
- [NRF.startBonding\(forceRepair\)](#)
- [NRF.updateServices\(data\)](#)
- [NRF.wake\(\)](#)

---

### [event NRF.advertising](#) ⇒

---

Call type:

[\(top\)](#)

```
NRF.on('advertising', function(isAdvertising) { ... });
```

### Parameters

isAdvertising - Whether we are advertising or not

## Description

Called when Bluetooth advertising starts or stops on Espruino

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

---

## [NRF.amsCommand](#) ⇒

### Call type:

[\(top\)](#)

```
NRF.amsCommand(id)
```

### Parameters

`id` - For example, 'play', 'pause', 'volup' or 'voldown'

## Description

Send an AMS command to an Apple Media Service device to control music playback

Command is one of play, pause, playpause, next, prev, volup, voldown, repeat, shuffle, skipforward, skipback, like, dislike, bookmark

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

---

## [NRF.amsGetPlayerInfo](#) ⇒

### Call type:

[\(top\)](#)

```
NRF.amsGetPlayerInfo(id)
```

### Parameters

`id` - Either 'name', 'playbackinfo' or 'volume'

### Returns

A [Promise](#) that is resolved (or rejected) when the connection is complete

## Description

Get Apple Media Service (AMS) info for the current media player. "playbackinfo" returns a concatenation of three comma-separated values:

- PlaybackState: a string that represents the integer value of the playback state: - PlaybackStatePaused = 0 - PlaybackStatePlaying = 1 - PlaybackStateRewinding = 2 - PlaybackStateFastForwarding = 3
- PlaybackRate: a string that represents the floating point value of the playback rate.
- ElapsedTime: a string that represents the floating point value of the elapsed time of the current track, in seconds

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

---

## [NRF.amsGetTrackInfo](#) ⇒

### Call type:

[\(top\)](#)

```
NRF.amsGetTrackInfo(id)
```

### Parameters

`id` - Either 'artist', 'album', 'title' or 'duration'

### Returns

A [Promise](#) that is resolved (or rejected) when the connection is complete

## Description

Get Apple Media Service (AMS) info for the currently-playing track

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

## [NRF.amsIsActive](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.amsIsActive()
```

**Parameters**

**Returns**

True if Apple Media Service (AMS) has been initialised and is active

**Description**

Check if Apple Media Service (AMS) is currently active on the BLE connection

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

## [NRF.ancsAction](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.ancsAction(uid, positive)
```

**Parameters**

`uid` - The UID of the notification to respond to

`positive` - `true` for positive action, `false` for negative

**Description**

Send an ANCS action for a specific Notification UID. Corresponds to posaction/negaction in the 'ANCS' event that was received

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

## [NRF.ancsGetAppInfo](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.ancsGetAppInfo(id)
```

**Parameters**

`id` - The app ID to get information for

**Returns**

A [promise](#) that is resolved (or rejected) when the connection is complete

**Description**

Get ANCS info for an app (app id is available via [NRF.ancsGetNotificationInfo](#))

Promise returns:

```
{
 "uid" : int,
 "appId" : string,
 "title" : string,
 "subtitle" : string,
 "message" : string,
 "messageSize" : string,
```

```
"date" : string,
"posAction" : string,
"negAction" : string,
"name" : string,
}
```

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

## [NRF.ancsGetNotificationInfo](#) ⇒

---

Call type:

[\(top\)](#)

```
NRF.ancsGetNotificationInfo(uid)
```

### Parameters

uid - The UID of the notification to get information for

### Returns

A [Promise](#) that is resolved (or rejected) when the connection is complete

### Description

Get ANCS info for a notification event received via [E.ANCS](#), e.g.:

```
E.on('ANCS', event => {
 NRF.ancsGetNotificationInfo(event.uid).then(a=>print("Notify",E.toJS(a)));
});
```

Returns:

```
{
 "uid" : integer,
 "appId": string,
 "title": string,
 "subtitle": string,
 "message": string,
 "messageSize": string,
 "date": string,
 "posAction": string,
 "negAction": string
}
```

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

## [NRF.ancsIsActive](#) ⇒

---

Call type:

[\(top\)](#)

```
NRF.ancsIsActive()
```

### Parameters

### Returns

True if Apple Notification Center Service (ANCS) has been initialised and is active

### Description

Check if Apple Notification Center Service (ANCS) is currently active on the BLE connection

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

## [event NRF.bond](#) ⇒

---

Call type:

[\(top\)](#)

```
NRF.on('bond', function(status) { ... });
```

## Parameters

status - One of 'request'/'start'/'success'/'fail'

## Description

Called during the bonding process to update on status

status is one of:

- "request" - Bonding has been requested in code via [NRF.startBonding](#)
- "start" - The bonding procedure has started
- "success" - The bonding procedure has succeeded ([NRF.startBonding](#)'s promise resolves)
- "fail" - The bonding procedure has failed ([NRF.startBonding](#)'s promise rejects)

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

---

## [event NRF.characteristicsDiscover](#) ⇒

### Call type:

[\(top\)](#)

```
NRF.on('characteristicsDiscover', function() { ... });
```

## Description

Called with discovered characteristics when discovery is finished

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) and ESP32 boards

---

## [event NRF.connect](#) ⇒

### Call type:

[\(top\)](#)

```
NRF.on('connect', function(addr) { ... });
```

## Parameters

addr - The address of the device that has connected

## Description

Called when a host device connects to Espruino. The first argument contains the address.

---

## [NRF.connect](#) ⇒

### Call type:

[\(top\)](#)

```
NRF.connect(mac, options)
```

## Parameters

mac - The MAC address to connect to

options - (Espruino-specific) An object of connection options (see [BluetoothRemoteGATTServer.connect](#) for full details)

## Returns

A [Promise](#) that is resolved (or rejected) when the connection is complete

## Description

Connect to a BLE device by MAC address. Returns a promise, the argument of which is the [BluetoothRemoteGATTServer](#) connection.

```
NRF.connect("aa:bb:cc:dd:ee").then(function(server) {
 // ...
});
```

This has the same effect as calling `BluetoothDevice.gatt.connect` on a `BluetoothDevice` requested using `NRF.requestDevice`. It just allows you to specify the address directly (without having to scan).

You can use it as follows - this would connect to another Puck device and turn its LED on:

```
var gatt;
NRF.connect("aa:bb:cc:dd:ee random").then(function(g) {
 gatt = g;
 return gatt.getPrimaryService("6e400001-b5a3-f393-e0a9-e50e24dcca9e");
}).then(function(service) {
 return service.getCharacteristic("6e400002-b5a3-f393-e0a9-e50e24dcca9e");
}).then(function(characteristic) {
 return characteristic.writeValue("LED1.set()\n");
}).then(function() {
 gatt.disconnect();
 console.log("Done!");
});
```

**Note:** Espruino Bluetooth devices use a type of BLE address known as 'random static', which is different to a 'public' address. To connect to an Espruino device you'll need to use an address string of the form

```
"aa:bb:cc:dd:ee
random"
```

rather than just "aa:bb:cc:dd:ee". If you scan for devices with `NRF.findDevices/NRF.setScan` then addresses are already reported in the correct format.

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) and ESP32 boards

## [event NRF.disconnect](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.on('disconnect', function(reason) { ... });
```

**Parameters**

`reason` - The reason code reported back by the BLE stack - see Nordic's [ble\\_hci.h file](#) for more information

**Description**

Called when a host device disconnects from Espruino.

The most common reason is: \* 19 - REMOTE\_USER\_TERMINATED\_CONNECTION \* 22 - LOCAL\_HOST\_TERMINATED\_CONNECTION

## [NRF.disconnect](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.disconnect()
```

**Description**

If a device is connected to Espruino, disconnect from it.

## [NRF.filterDevices](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.filterDevices(devices, filters)
```

**Parameters**

`devices` - An array of `BluetoothDevice` objects, from `NRF.findDevices` or similar

`filters` - A list of filters (as would be passed to `NRF.requestDevice`) to filter devices by

**Returns**

An array of `BluetoothDevice` objects that match the given filters

**Description**

This function can be used to quickly filter through Bluetooth devices.

For instance if you wish to scan for multiple different types of device at the same time then you could use [NRF.findDevices](#) with all the filters you're interested in. When scanning is finished you can then use [NRF.filterDevices](#) to pick out just the devices of interest.

```
// the two types of device we're interested in
var filter1 = [{serviceData:{'fe95':{}}}];
var filter2 = [{namePrefix:"Pixel.js"}];
// the following filter will return both types of device
var allFilters = filter1.concat(filter2);
// now scan for both types of device, and filter them out afterwards
NRF.findDevices(function(devices) {
 var devices1 = NRF.filterDevices(devices, filter1);
 var devices2 = NRF.filterDevices(devices, filter2);
 // ...
}, {filters : allFilters});
```

## [NRF.findDevices](#) ⇒

---

Call type:

[\(top\)](#)

```
NRF.findDevices(callback, options)
```

Parameters

callback - The callback to call with received advertising packets (as [BluetoothDevice](#)), or undefined to stop

options - [optional] A time in milliseconds to scan for (defaults to 2000), Or an optional object `{filters: ..., timeout : ..., active: bool}` (as would be passed to [NRF.requestDevice](#)) to filter devices by

Description

Utility function to return a list of BLE devices detected in range. Behind the scenes, this uses [NRF.setScan\(...\)](#) and collates the results.

```
NRF.findDevices(function(devices) {
 console.log(devices);
}, 1000);
```

prints something like:

```
[
 BluetoothDevice {
 "id" : "e7:e0:57:ad:36:a2 random",
 "rssi": -45,
 "services": ["4567"],
 "serviceData" : { "0123" : [1] },
 "manufacturer" : 1424,
 "manufacturerData" : new Uint8Array([...]).buffer,
 "data": new ArrayBuffer([...]),
 "name": "Puck.js 36a2"
 },
 BluetoothDevice {
 "id": "c0:52:3f:50:42:c9 random",
 "rssi": -65,
 "data": new ArrayBuffer([...]),
 "name": "Puck.js 8f57"
 }
]
```

For more information on the structure returned, see [NRF.setScan](#).

If you want to scan only for specific devices you can replace the timeout with an object of the form `{filters: ..., timeout : ..., active: bool}` using the filters described in [NRF.requestDevice](#). For example to search for devices with Espruino's manufacturerData:

```
NRF.findDevices(function(devices) {
 ...
}, {timeout : 2000, filters : [{ manufacturerData:{0x0590:{} } }] });
```

You could then use [BluetoothDevice.gatt.connect\(...\)](#) on the device returned to make a connection.

You can also use [NRF.connect\(...\)](#) on just the id string returned, which may be useful if you always want to connect to a specific device.

**Note:** Using [findDevices](#) turns the radio's receive mode on for 2000ms (or however long you specify). This can draw a *lot* of power (12mA or so), so you should use it sparingly or you can run your battery down quickly.

**Note:** The 'data' field contains the data of *the last packet received*. There may have been more packets. To get data for each packet individually use [NRF.setScan](#) instead.

## [NRF.getAddress](#) ⇒

---

Call type:

[\(top\)](#)

```
NRF.getAddress()
```

## Returns

MAC address - a string of the form 'aa:bb:cc:dd:ee:ff'

## Description

Get this device's default Bluetooth MAC address.

For Puck.js, the last 5 characters of this (e.g. ee:ff) are used in the device's advertised Bluetooth name.

---

## [NRF.getAdvertisingData](#) ⇒

### Call type:

[\(top\)](#)

```
NRF.getAdvertisingData(data, options)
```

## Parameters

**data** - The data to advertise as an object

**options** - [optional] An object of options

## Returns

An array containing the advertising data

## Description

This is just like [NRF.setAdvertising](#), except instead of advertising the data, it returns the packet that would be advertised as an array.

---

## [NRF.getBattery](#) ⇒

### Call type:

[\(top\)](#)

```
NRF.getBattery()
```

## Returns

Battery level in volts

## Description

Get the battery level in volts (the voltage that the NRF chip is running off of).

This is the battery level of the device itself - it has nothing to do with any device that might be connected.

---

## [NRF.getSecurityStatus](#) ⇒

### Call type:

[\(top\)](#)

```
NRF.getSecurityStatus()
```

## Returns

An object

## Description

Return an object with information about the security state of the current peripheral connection:

```
{
 connected // The connection is active (not disconnected).
 encrypted // Communication on this link is encrypted.
}
```

```
mitm_protected // The encrypted communication is also protected against man-in-the-middle attacks.
bonded // The peer is bonded with us
advertising // Are we currently advertising?
connected_addr // If connected=true, the MAC address of the currently connected device
}
```

If there is no active connection, `{connected:false}` will be returned.

See [NRF.setSecurity](#) for information about negotiating a secure connection.

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

## [event NRF.HID](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.on('HID', function() { ... });
```

### Description

Called with a single byte value when Espruino is set up as a HID device and the computer it is connected to sends a HID report back to Espruino. This is usually used for handling indications such as the Caps Lock LED.

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

## [NRF.nfcAndroidApp](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.nfcAndroidApp(app)
```

### Parameters

app - The unique identifier of the given Android App

### Description

Enables NFC with a record that will launch the given android app.

For example:

```
NRF.nfcAndroidApp("no.nordicsemi.android.nrftoolbox")
```

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) with NFC (Puck.js, Pixl.js, MDBT42Q)

## [event NRF.NFCoff](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.on('NFCoff', function() { ... });
```

### Description

Called when an NFC field is no longer detected

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) with NFC (Puck.js, Pixl.js, MDBT42Q)

## [event NRF.NFCon](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.on('NFCon', function() { ... });
```

### Description

Called when an NFC field is detected

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) with NFC (Puck.js, Pixl.js, MDBT42Q)

## [NRF.nfcPair](#) ⇒

---

### Call type:

[\(top\)](#)

```
NRF.nfcPair(key)
```

### Parameters

key - 16 byte out of band key

### Description

Enables NFC and with an out of band 16 byte pairing key.

For example the following will enable out of band pairing on BLE such that the device will pair when you tap the phone against it:

```
var bleKey = [0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0x99, 0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00];
NRF.on('security', s=>print("security", JSON.stringify(s)));
NRF.nfcPair(bleKey);
NRF.setSecurity({oob:bleKey, mitm:true});
```

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) with NFC (Puck.js, Pixl.js, MDBT42Q)

## [NRF.nfcRaw](#) ⇒

---

### Call type:

[\(top\)](#)

```
NRF.nfcRaw(payload)
```

### Parameters

payload - The NFC NDEF message to deliver to the reader

### Description

Enables NFC and starts advertising with Raw data. For example:

```
NRF.nfcRaw(new Uint8Array([193, 1, 0, 0, 0, 13, 85, 3, 101, 115, 112, 114, 117, 105, 110, 111, 46, 99, 111, 109]));
// same as NRF.nfcURL("http://espruino.com");
```

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) with NFC (Puck.js, Pixl.js, MDBT42Q)

## [event NRF.NFCrx](#) ⇒

---

### Call type:

[\(top\)](#)

```
NRF.on('NFCrx', function(arr) { ... });
```

### Parameters

arr - An ArrayBuffer containign the received data

### Description

When NFC is started with [NRF.nfcStart](#), this is fired when NFC data is received. It doesn't get called if NFC is started with [NRF.nfcURL](#) or [NRF.nfcRaw](#)

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) with NFC (Puck.js, Pixl.js, MDBT42Q)

## [NRF.nfcSend](#) ⇒

---

### Call type:

[\(top\)](#)

```
NRF.nfcSend(payload)
```

## Parameters

payload - Optional tx data

## Description

**Advanced NFC Functionality.** If you just want to advertise a URL, use [NRF.nfcURL](#) instead.

Acknowledges the last frame and optionally transmits a response. If payload is an array, then a array.length byte nfc frame is sent. If payload is a int, then a 4bit ACK/NACK is sent. **Note:** nfcSend should always be called after an **NFCrx** event.

```
NRF.nfcSend(new Uint8Array([0x01, 0x02, ...]));
// or
NRF.nfcSend(0xA);
// or
NRF.nfcSend();
```

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) with NFC (Puck.js, Pixl.js, MDBT42Q)

---

## [NRF.nfcStart](#) ⇒

### Call type:

[\(top\)](#)

```
NRF.nfcStart(payload)
```

## Parameters

payload - Optional 7 byte UID

## Returns

Internal tag memory (first 10 bytes of tag data)

## Description

**Advanced NFC Functionality.** If you just want to advertise a URL, use [NRF.nfcURL](#) instead.

Enables NFC and starts advertising. **NFCrx** events will be fired when data is received.

```
NRF.nfcStart();
```

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) with NFC (Puck.js, Pixl.js, MDBT42Q)

---

## [NRF.nfcStop](#) ⇒

### Call type:

[\(top\)](#)

```
NRF.nfcStop()
```

## Parameters

## Description

**Advanced NFC Functionality.** If you just want to advertise a URL, use [NRF.nfcURL](#) instead.

Disables NFC.

```
NRF.nfcStop();
```

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) with NFC (Puck.js, Pixl.js, MDBT42Q)

---

## [NRF.nfcURL](#) ⇒

### Call type:

[\(top\)](#)

```
NRF.nfcURL(url)
```

## Parameters

url - The URL string to expose on NFC, or undefined to disable NFC

## Description

Enables NFC and starts advertising the given URL. For example:

```
NRF.nfcURL("http://espruino.com");
```

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) with NFC (Puck.js, Pixl.js, MDBT42Q)

## NRF.requestDevice =>

---

### Call type:

[\(top\)](#)

```
NRF.requestDevice(options)
```

### Parameters

`options` - Options used to filter the device to use

### Returns

A [Promise](#) that is resolved (or rejected) when the connection is complete

## Description

Search for available devices matching the given filters. Since we have no UI here, Espruino will pick the FIRST device it finds, or it'll call `catch`.

`options` can have the following fields:

- `filters` - a list of filters that a device must match before it is returned (see below)
- `timeout` - the maximum time to scan for in milliseconds (scanning stops when a match is found. e.g. `NRF.requestDevice({ timeout:2000, filters: [ ... ] })`)
- `active` - whether to perform active scanning (requesting 'scan response' packets from any devices that are found). e.g.

```
NRF.requestDevice({ active:true,
 filters: [...] })

• phy - (NRF52840 only) use the long-range coded phy ("1mbps" default, can be "1mbps/2mbps/both/coded")
• extended - (NRF52840 only) support receiving extended-length advertising packets (default=true if phy isn't "1mbps")
```

**NOTE:** `timeout` and `active` are not part of the Web Bluetooth standard.

The following filter types are implemented:

- `services` - list of services as strings (all of which must match). 128 bit services must be in the form '01230123-0123-0123-0123-012301230123'
- `name` - exact device name
- `namePrefix` - starting characters of device name
- `id` - exact device address (`id:"e9:53:86:09:89:99 random"`) (this is Espruino-specific, and is not part of the Web Bluetooth spec)
- `serviceData` - an object containing service characteristics which must all match (`serviceData:{"1809":{}}`). Matching of actual service data is not supported yet.
- `manufacturerData` - an object containing manufacturer UUIDs which must all match (`manufacturerData:{0x0590:{}}`). Matching of actual manufacturer data is not supported yet.

```
NRF.requestDevice({ filters: [{ namePrefix: 'Puck.js' }] }).then(function(device) { ... });
// or
NRF.requestDevice({ filters: [{ services: ['1823'] }] }).then(function(device) { ... });
// or
NRF.requestDevice({ filters: [{ manufacturerData:{0x0590:{}} }] }).then(function(device) { ... });
```

As a full example, to send data to another Puck.js to turn an LED on:

```
var gatt;
NRF.requestDevice({ filters: [{ namePrefix: 'Puck.js' }] }).then(function(device) {
 return device.gatt.connect();
}).then(function(g) {
 gatt = g;
 return gatt.getPrimaryService("6e400001-b5a3-f393-e0a9-e50e24dcca9e");
}).then(function(service) {
 return service.getCharacteristic("6e400002-b5a3-f393-e0a9-e50e24dcca9e");
}).then(function(characteristic) {
 return characteristic.writeValue("LED1.set()\n");
}).then(function() {
 gatt.disconnect();
 console.log("Done!");
});
```

Or slightly more concisely, using ES6 arrow functions:

```

var gatt;
NRF.requestDevice({ filters: [{ namePrefix: 'Puck.js' }] }).then(
 device => device.gatt.connect().then(
 g => (gatt=g).getPrimaryService("6e400001-b5a3-f393-e0a9-e50e24dcca9e").then(
 service => service.getCharacteristic("ge400002-b5a3-f393-e0a9-e50e24dcca9e").then(
 characteristic => characteristic.writeValue("LED1.reset()\n").then(
 () => { gatt.disconnect(); console.log("Done!"); })
)
)
)

```

Note that you have to keep track of the `gatt` variable so that you can disconnect the Bluetooth connection when you're done.

**Note:** Using a filter in `NRF.requestDevice` filters each advertising packet individually. As soon as a matching advertisement is received, `NRF.requestDevice` resolves the promise and stops scanning. This means that if you filter based on a service UUID and a device advertises with multiple packets (or a scan response when `active:true`) only the packet matching the filter is returned - you may not get the device's name is that was in a separate packet. To aggregate multiple packets you can use `NRF.findDevices`.

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) and ESP32 boards

## [NRF.restart](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.restart(callback)
```

**Parameters**

`callback` - [optional] A function to be called while the softdevice is uninitialised. Use with caution - accessing console/bluetooth will almost certainly result in a crash.

**Description**

Restart the Bluetooth softdevice (if there is currently a BLE connection, it will queue a restart to be done when the connection closes).

You shouldn't need to call this function in normal usage. However, Nordic's BLE softdevice has some settings that cannot be reset. For example there are only a certain number of unique UUIDs. Once these are all used the only option is to restart the softdevice to clear them all out.

## [event NRF.security](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.on('security', function(status) { ... });
```

**Parameters**

`status` - An object containing `auth\_status,bonded,lv4,kdist\_own,kdist\_peer`

**Description**

Contains updates on the security of the current Bluetooth link.

See Nordic's `ble_gap_evt_auth_status_t` structure for more information.

## [NRF.sendHIDReport](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.sendHIDReport(data, callback)
```

**Parameters**

`data` - Input report data as an array

`callback` - A callback function to be called when the data is sent

**Description**

Send a USB HID report. HID must first be enabled with

```
NRF.setServices({}, {hid:
 hid_report})
```

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

## [NRF.servicesDiscover](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.on('servicesDiscover', function() { ... });
```

### Description

Called with discovered services when discovery is finished

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q) and ESP32 boards

## [NRF.setAddress](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.setAddress(addr)
```

### Parameters

addr - The address to use (as a string)

### Description

Set this device's default Bluetooth MAC address:

```
NRF.setAddress("ff:ee:dd:cc:bb:aa random");
```

Addresses take the form:

- "ff:ee:dd:cc:bb:aa" or "ff:ee:dd:cc:bb:aa public" for a public address
- "ff:ee:dd:cc:bb:aa random" for a random static address (the default for Espruino)

This may throw a `INVALID_BLE_ADDR` error if the upper two bits of the address don't match the address type.

To change the address, Espruino must restart the softdevice. It will only do so when it is disconnected from other devices.

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

## [NRF.setAdvertising](#) ⇒

---

**Call type:**

[\(top\)](#)

```
NRF.setAdvertising(data, options)
```

### Parameters

data - The service data to advertise as an object - see below for more info

options - [optional] Object of options

### Description

Change the data that Espruino advertises.

Data can be of the form `{ UUID : data_as_byte_array }`. The UUID should be a [Bluetooth Service ID](#).

For example to return battery level at 95%, do:

```
NRF.setAdvertising({
 0x180F : [95] // Service data 0x180F = 95
});
```

Or you could report the current temperature:

```
setInterval(function() {
 NRF.setAdvertising({
 0x1809 : [Math.round(E.getTemperature())]
});
```

```
});
, 30000);
```

If you specify a value for the object key, Service Data is advertised. However if you specify **undefined**, the Service UUID is advertised:

```
NRF.setAdvertising({
 0x180D : undefined // Advertise service UUID 0x180D (HRM)
});
```

Service UUIDs can also be supplied in the second argument of [NRF.setServices](#), but those go in the scan response packet.

You can also supply the raw advertising data in an array. For example to advertise as an Eddystone beacon:

```
NRF.setAdvertising([0x03, // Length of Service List
 0x03, // Param: Service List
 0xAA, 0xFE, // Eddystone ID
 0x13, // Length of Service Data
 0x16, // Service Data
 0xAA, 0xFE, // Eddystone ID
 0x10, // Frame type: URL
 0xF8, // Power
 0x03, // https://
 'g','o','o','.', 'g','l','/','B','3','J','0','O','c'],
 {interval:100});
```

(However for Eddystone we'd advise that you use the [Espruino Eddystone library](#))

**Note:** When specifying data as an array, certain advertising options such as `discoverable` and `showName` won't have any effect.

**Note:** The size of Bluetooth LE advertising packets is limited to 31 bytes. If you want to advertise more data, consider using an array for `data` (See below), or [NRF.setScanResponse](#).

You can even specify an array of arrays or objects, in which case each advertising packet will be used in turn - for instance to make your device advertise battery level and its name as well as both Eddystone and iBeacon :

```
NRF.setAdvertising([
 {0x180F : [Puck.getBatteryPercentage()]}, // normal advertising, with battery %
 require("ble_ibeacon").get(...), // ibeacon
 require("ble_eddystone").get(...), // eddystone
], {interval:300});
```

`options` is an object, which can contain:

```
{
 name: "Hello" // The name of the device
 showName: true/false // include full name, or nothing
 discoverable: true/false // general discoverable, or limited - default is limited
 connectable: true/false // whether device is connectable - default is true
 scannable: true/false // whether device can be scanned for scan response packets - default is true
 whenConnected : true/false // keep advertising when connected (nRF52 only)
 interval: 600 // Advertising interval in msec, between 20 and 10000 (default is 375ms)
 manufacturer: 0x0590 // IF sending manufacturer data, this is the manufacturer ID
 manufacturerData: [...] // IF sending manufacturer data, this is an array of data
 phy: "1mbps/2mbps/coded" // (NRF52840 only) use the long-range coded phy for transmission (1mbps default)
}
```

Setting `connectable` and `scannable` to false gives the lowest power consumption as the BLE radio doesn't have to listen after sending advertising.

**NOTE:** Non-connectable advertising can't have an advertising interval less than 100ms according to the BLE spec.

So for instance to set the name of Puck.js without advertising any other data you can just use the command:

```
NRF.setAdvertising({}, {name:"Hello"});
```

You can also specify 'manufacturer data', which is another form of advertising data. We've registered the Manufacturer ID 0x0590 (as Pur3 Ltd) for use with *Official Espruino devices* - use it to advertise whatever data you'd like, but we'd recommend using JSON.

For example by not advertising a device name you can send up to 24 bytes of JSON on Espruino's manufacturer ID:

```
var data = {a:1,b:2};
NRF.setAdvertising({}, {
 showName:false,
 manufacturer:0x0590,
 manufacturerData:JSON.stringify(data)
});
```

If you're using [EspruinoHub](#) then it will automatically decode this into the following MQTT topics:

- /ble/advertise/ma:c:\_a:dd:re:ss/espruino -> {"a":10,"b":15}
- /ble/advertise/ma:c:\_a:dd:re:ss/a -> 1
- /ble/advertise/ma:c:\_a:dd:re:ss/b -> 2

Note that **you only have 24 characters available for JSON**, so try to use the shortest field names possible and avoid floating point values that can be very long when converted to a String.

[NRF.setConnectionInterval](#) ⇒

## Call type:

[\(top\)](#)

```
NRF.setConnectionInterval(interval)
```

## Parameters

interval - The connection interval to use (see below)

## Description

When connected, Bluetooth LE devices communicate at a set interval. Lowering the interval (e.g. more packets/second) means a lower delay when sending data, higher bandwidth, but also more power consumption.

By default, when connected as a peripheral Espruino automatically adjusts the connection interval. When connected it's as fast as possible (7.5ms) but when idle for over a minute it drops to 200ms. On continued activity (>1 BLE operation) the interval is raised to 7.5ms again.

The options for interval are:

- `undefined` / `"auto"` : (default) automatically adjust connection interval
- `100` : set min and max connection interval to the same number (between 7.5ms and 4000ms)
- `{minInterval:20, maxInterval:100}` : set min and max connection interval as a range

This configuration is not remembered during a `save()` - you will have to re-set it via `onInit`.

**Note:** If connecting to another device (as Central), you can use an extra argument to `NRF.connect` or `BluetoothRemoteGATTServer.connect` to specify a connection interval.

**Note:** This overwrites any changes imposed by the deprecated `NRF.setLowPowerConnection`

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

---

## [NRF.setLowPowerConnection](#) =>

## Call type:

[\(top\)](#)

```
NRF.setLowPowerConnection(lowPower)
```

## Parameters

lowPower - Whether the connection is low power or not

## Description

**THIS IS DEPRECATED** - please use `NRF.setConnectionInterval` for peripheral and `NRF.connect(addr, options)/BluetoothRemoteGATTServer.connect(options)` for central connections.

This sets the connection parameters - these affect the transfer speed and power usage when the device is connected.

- When not low power, the connection interval is between 7.5 and 20ms
- When low power, the connection interval is between 500 and 1000ms

When low power connection is enabled, transfers of data over Bluetooth will be very slow, however power usage while connected will be drastically decreased.

This will only take effect after the connection is disconnected and re-established.

---

## [NRF.setRSSIHandler](#) =>

## Call type:

[\(top\)](#)

```
NRF.setRSSIHandler(callback)
```

## Parameters

callback - The callback to call with the RSSI value, or undefined to stop

## Description

Start/stop listening for RSSI values on the currently active connection (where This device is a peripheral and is being connected to by a 'central' device)

```
// Start scanning
NRF.setRSSIHandler(function(rssi) {
```

```
 console.log(rssi); // prints -85 (or similar)
 });
// Stop Scanning
NRF.setRSSIHandler();
```

RSSI is the 'Received Signal Strength Indication' in dBm

## [NRF.setScan](#) →

---

### Call type:

[\(top\)](#)

```
NRF.setScan(callback, options)
```

### Parameters

**callback** - The callback to call with received advertising packets, or undefined to stop

**options** - [optional] An object {filters: ...} (as would be passed to [NRF.requestDevice](#)) to filter devices by

### Description

Start/stop listening for BLE advertising packets within range. Returns a [BluetoothDevice](#) for each advertising packet. **By default this is not an active scan, so Scan Response advertising data is not included (see below)**

```
// Start scanning
packets=10;
NRF.setScan(function(d) {
 packets--;
 if (packets<=0)
 NRF.setScan(); // stop scanning
 else
 console.log(d); // print packet info
});
```

Each [BluetoothDevice](#) will look a bit like:

```
BluetoothDevice {
 "id": "aa:bb:cc:dd:ee:ff", // address
 "rssi": -89, // signal strength
 "services": ["128bit-uuid", ...], // zero or more service UUIDs
 "data": new Uint8Array([...]).buffer, // ArrayBuffer of returned data
 "serviceData" : { "0123" : [1] }, // if service data is in 'data', it's extracted here
 "manufacturer" : 0x1234, // if manufacturer data is in 'data', the 16 bit manufacturer ID is extracted here
 "manufacturerData" : new Uint8Array([...]).buffer, // if manufacturer data is in 'data', the data is extracted here as an ArrayBuffer
 "name": "DeviceName" // the advertised device name
}
```

You can also supply a set of filters (as described in [NRF.requestDevice](#)) as a second argument, which will allow you to filter the devices you get a callback for. This helps to cut down on the time spent processing JavaScript code in areas with a lot of Bluetooth advertisements. For example to find only devices with the manufacturer data `0x0590` (Espruino's ID) you could do:

```
NRF.setScan(function(d) {
 console.log(d.manufacturerData);
}, { filters: [{ manufacturerData:{0x0590:{} } }] });
```

You can also specify `active:true` in the second argument to perform active scanning (this requests scan response packets) from any devices it finds.

**Note:** Using a filter in `setScan` filters each advertising packet individually. As a result, if you filter based on a service UUID and a device advertises with multiple packets (or a scan response when `active:true`) only the packets matching the filter are returned. To aggregate multiple packets you can use [NRF.findDevices](#).

**Note:** BLE advertising packets can arrive quickly - faster than you'll be able to print them to the console. It's best only to print a few, or to use a function like `NRF.findDevices(...)` which will collate a list of available devices.

**Note:** Using `setScan` turns the radio's receive mode on constantly. This can draw a *lot* of power (12mA or so), so you should use it sparingly or you can run your battery down quickly.

## [NRF.setScanResponse](#) →

---

### Call type:

[\(top\)](#)

```
NRF.setScanResponse(data)
```

### Parameters

**data** - The data to for the scan response

### Description

The raw scan response data should be supplied as an array. For example to return "Sample" for the device name:

```
NRF.setScanResponse([0x07, // Length of Data
 0x09, // Param: Complete Local Name
 'S', 'a', 'm', 'p', 'l', 'e']);
```

**Note:** `NRF.setServices(..., {advertise:[ ... ]})` writes advertised services into the scan response - so you can't use both `advertise` and `NRF.setServices` or one will overwrite the other.

## [NRF.setSecurity](#) ⇒

---

Call type:

[\(top\)](#)

```
NRF.setSecurity(options)
```

### Parameters

`options` - An object containing security-related options (see below)

### Description

Sets the security options used when connecting/pairing. This applies to both central *and* peripheral mode.

```
NRF.setSecurity({
 display : bool // default false, can this device display a passkey
 // - sent via the `BluetoothDevice.passkey` event
 keyboard : bool // default false, can this device enter a passkey
 // - request sent via the `BluetoothDevice.passkeyRequest` event
 bond : bool // default true, Perform bonding
 mitm : bool // default false, Man In The Middle protection
 lesc : bool // default false, LE Secure Connections
 passkey : // default "", or a 6 digit passkey to use
 oob : [0..15] // if specified, Out Of Band pairing is enabled and
 // the 16 byte pairing code supplied here is used
 encryptUart : bool // default false (unless oob or passkey specified)
 // This sets the BLE UART service such that it
 // is encrypted and can only be used from a bonded connection
});
```

**NOTE:** Some combinations of arguments will cause an error. For example supplying a passkey without `display:1` is not allowed. If `display:1` is set you do not require a physical display, the user just needs to know the passkey you supplied.

For instance, to require pairing and to specify a passkey, use:

```
NRF.setSecurity({passkey:"123456", mitm:1, display:1});
```

However, while most devices will request a passkey for pairing at this point it is still possible for a device to connect without requiring one (e.g. using the 'NRF Connect' app).

To force a passkey you need to protect each characteristic you define with `NRF.setSecurity`. For instance the following code will *require* that the passkey 123456 is entered before the characteristic 9d020002-bf5f-1d1a-b52a-fe52091d5b12 can be read.

```
NRF.setSecurity({passkey:"123456", mitm:1, display:1});
NRF.setServices({
 "9d020001-bf5f-1d1a-b52a-fe52091d5b12" : {
 "9d020002-bf5f-1d1a-b52a-fe52091d5b12" : {
 // readable always
 value : "Not Secret"
 },
 "9d020003-bf5f-1d1a-b52a-fe52091d5b12" : {
 // readable only once bonded
 value : "Secret",
 readable : true,
 security: {
 read: {
 mitm: true,
 encrypted: true
 }
 }
 },
 "9d020004-bf5f-1d1a-b52a-fe52091d5b12" : {
 // readable always
 // writable only once bonded
 value : "Readable",
 readable : true,
 writable : true,
 onWrite : function(evt) {
 console.log("Wrote ", evt.data);
 },
 security: {
 write: {
 mitm: true,
 encrypted: true
 }
 }
 }
 }
});
```

```
});
```

**Note:** If passkey or oob is specified, the Nordic UART service (if enabled) will automatically be set to require encryption, but otherwise it is open.

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

## [NRF.setServices](#) ⇒

---

### Call type:

```
NRF.setServices(data, options)
```

### Parameters

**data** - The service (and characteristics) to advertise

**options** - [optional] Object containing options

### Description

Change the services and characteristics Espruino advertises.

If you want to **change** the value of a characteristic, you need to use [NRF.updateServices\(\)](#) instead

To expose some information on Characteristic ABCD on service BCDE you could do:

```
NRF.setServices({
 0xBCDE : {
 0xABCD : {
 value : "Hello",
 readable : true
 }
 }
});
```

Or to allow the 3 LEDs to be controlled by writing numbers 0 to 7 to a characteristic, you can do the following. evt.data is an ArrayBuffer.

```
NRF.setServices({
 0xBCDE : {
 0xABCD : {
 writable : true,
 onWrite : function(evt) {
 digitalWrite([LED3,LED2,LED1], evt.data[0]);
 }
 }
 }
});
```

You can supply many different options:

```
NRF.setServices({
 0xBCDE : {
 0xABCD : {
 value : "Hello", // optional
 maxLen : 5, // optional (otherwise is length of initial value)
 broadcast : false, // optional, default is false
 readable : true, // optional, default is false
 writable : true, // optional, default is false
 notify : true, // optional, default is false
 indicate : true, // optional, default is false
 description: "My Characteristic", // optional, default is null,
 security: { // optional - see NRF.setSecurity
 read: { // optional
 encrypted: false, // optional, default is false
 mitm: false, // optional, default is false
 lesc: false, // optional, default is false
 signed: false // optional, default is false
 },
 write: { // optional
 encrypted: true, // optional, default is false
 mitm: false, // optional, default is false
 lesc: false, // optional, default is false
 signed: false // optional, default is false
 }
 },
 onWrite : function(evt) { // optional
 console.log("Got ", evt.data); // an ArrayBuffer
 },
 onWriteDesc : function(evt) { // optional - called when the 'cccd' descriptor is written
 // for example this is called when notifications are requested by the client:
 console.log("Notifications enabled = ", evt.data[0]&1);
 }
 }
 }
});
```

[\(top\)](#)

```

 // more services allowed
};

Note: UUIDs can be integers between 0 and 0xFFFF, strings of the form "ABCD", or strings of the form "ABCDABCD-ABCD-ABCD-ABCD-ABCDABCDABCD"
options can be of the form:

NRF.setServices(undefined, {
 hid : new Uint8Array(...), // optional, default is undefined. Enable BLE HID support
 uart : true, // optional, default is true. Enable BLE UART support
 advertise: ['180D'] // optional, list of service UUIDs to advertise
 ancs : true, // optional, Bangle.js-only, enable Apple ANCS support for notifications
 ams : true // optional, Bangle.js-only, enable Apple AMS support for media control
});

```

To enable BLE HID, you must set hid to an array which is the BLE report descriptor. The easiest way to do this is to use the `ble_hid_controls` or `ble_hid_keyboard` modules.

**Note:** Just creating a service doesn't mean that the service will be advertised. It will only be available after a device connects. To advertise, specify the UUIDs you wish to advertise in the `advertise` field of the second `options` argument. For example this will create and advertise a heart rate service:

```

NRF.setServices({
 0x180D: { // heart_rate
 0x2A37: { // heart_rate_measurement
 notify: true,
 value : [0x06, heartrate],
 }
 }
}, { advertise: ['180D'] });

```

You may specify 128 bit UUIDs to advertise, however you may get a `DATA_SIZE` exception because there is insufficient space in the Bluetooth LE advertising packet for the 128 bit UART UUID as well as the UUID you specified. In this case you can add `uart:false` after the `advertise` element to disable the UART, however you then be unable to connect to Puck.js's console via Bluetooth.

If you absolutely require two or more 128 bit UUIDs then you will have to specify your own raw advertising data packets with [NRF.setAdvertising](#).

**Note:** The services on Espruino can only be modified when there is no device connected to it as it requires a restart of the Bluetooth stack. **iOS devices will 'cache' the list of services** so apps like NRF Connect may incorrectly display the old services even after you have modified them. To fix this, disable and re-enable Bluetooth on your iOS device, or use an Android device to run NRF Connect.

**Note:** Not all combinations of security configuration values are valid, the valid combinations are: encrypted, encrypted + mitm, lesc, signed, signed + mitm. See [NRF.setSecurity](#) for more information.

## [NRF.setTxPower](#) ⇒

---

**Call type:**

```
NRF.setTxPower(power)
```

[\(top\)](#)

**Parameters**

`power` - Transmit power. Accepted values are -40(nRF52 only), -30(nRF51 only), -20, -16, -12, -8, -4, 0, and 4 dBm. On nRF52840 (eg Bangle.js 2) 5/6/7/8 dBm are available too. Others will give an error code.

**Description**

Set the BLE radio transmit power. The default TX power is 0 dBm, and

## [NRF.setWhitelist](#) ⇒

---

**Call type:**

```
NRF.setWhitelist(whitelisting)
```

[\(top\)](#)

**Parameters**

`whitelisting` - Are we using a whitelist? (default false)

**Description**

If set to true, whenever a device bonds it will be added to the whitelist.

When set to false, the whitelist is cleared and newly bonded devices will not be added to the whitelist.

**Note:** This is remembered between [reset\(\)](#)s but isn't remembered after power-on (you'll have to add it to `onInit()`).

**Note:** This is only available in NRF52 devices (like Puck.js, Pixl.js, Bangle.js and MDBT42Q)

## [NRF.sleep](#) ⇒

---

### Call type:

[\(top\)](#)

```
NRF.sleep()
```

### Description

Disable Bluetooth advertising and disconnect from any device that connected to Puck.js as a peripheral (this won't affect any devices that Puck.js initiated connections to).

This makes Puck.js undiscoverable, so it can't be connected to.

Use [NRF.wake\(\)](#) to wake up and make Puck.js connectable again.

## [NRF.startBonding](#) ⇒

---

### Call type:

[\(top\)](#)

```
NRF.startBonding(forceRepair)
```

### Parameters

`forceRepair` - True if we should force repairing even if there is already valid pairing info

### Returns

A promise

## [NRF.updateServices](#) ⇒

---

### Call type:

[\(top\)](#)

```
NRF.updateServices(data)
```

### Parameters

`data` - The service (and characteristics) to update

### Description

Update values for the services and characteristics Espruino advertises. Only services and characteristics previously declared using [NRF.setServices](#) are affected.

To update the '0xABCD' characteristic in the '0xBCDE' service:

```
NRF.updateServices({
 0xBCDE : {
 0xABCD : {
 value : "World"
 }
 }
});
```

You can also use 128 bit UUIDs, for example "`b7920001-3c1b-4b40-869f-3c0db9be80c6`".

To define a service and characteristic and then notify connected clients of a change to it when a button is pressed:

```
NRF.setServices({
 0xBCDE : {
 0xABCD : {
 value : "Hello",
 maxLen : 20,
 notify: true
 }
 }
});
setWatch(function() {
 NRF.updateServices({
 0xBCDE : {
 0xABCD : {
 value : "World"
 }
 }
 });
});
```

```
 value : "World!",
 notify: true
 }
});
}, BTN, { repeat:true, edge:"rising", debounce: 50 });

```

This only works if the characteristic was created with `notify: true` using [NRF.setServices](#), otherwise the characteristic will be updated but no notification will be sent.

Also note that `maxLen` was specified. If it wasn't then the maximum length of the characteristic would have been 5 - the length of "`Hello`".

To indicate (i.e. notify with ACK) connected clients of a change to the '0xABCD' characteristic in the '0xBCDE' service:

```
NRF.updateServices({
 0xBCDE : {
 0xABCD : {
 value : "World",
 indicate: true
 }
 }
});
```

This only works if the characteristic was created with `indicate: true` using [NRF.setServices](#), otherwise the characteristic will be updated but no notification will be sent.

**Note:** See [NRF.setServices](#) for more information

## [NRF.wake](#) =>

---

### Call type:

[\(top\)](#)

```
NRF.wake()
```

### Description

Enable Bluetooth advertising (this is enabled by default), which allows other devices to discover and connect to Puck.js.

Use [NRF.sleep\(\)](#) to disable advertising.

---

## [Nucleo Class](#)

---

This is the built-in class for the Arduino-style pin namings on ST Nucleo boards

[\(top\)](#)

### Methods and Fields

- [Nucleo.A0](#)
- [Nucleo.A1](#)
- [Nucleo.A2](#)
- [Nucleo.A3](#)
- [Nucleo.A4](#)
- [Nucleo.A5](#)
- [Nucleo.D0](#)
- [Nucleo.D1](#)
- [Nucleo.D10](#)
- [Nucleo.D11](#)
- [Nucleo.D12](#)
- [Nucleo.D13](#)
- [Nucleo.D14](#)
- [Nucleo.D15](#)
- [Nucleo.D2](#)
- [Nucleo.D3](#)
- [Nucleo.D4](#)
- [Nucleo.D5](#)
- [Nucleo.D6](#)
- [Nucleo.D7](#)
- [Nucleo.D8](#)
- [Nucleo.D9](#)

---

### [Nucleo.A0](#) =>

---

#### Call type:

[\(top\)](#)

`Nucleo.A0`

#### Returns

A Pin

---

### [Nucleo.A1](#) =>

---

#### Call type:

[\(top\)](#)

`Nucleo.A1`

#### Returns

A Pin

---

### [Nucleo.A2](#) =>

---

#### Call type:

[\(top\)](#)

`Nucleo.A2`

#### Returns

A Pin

---

### [Nucleo.A3](#) =>

---

**Call type:**

`Nucleo.A3`

**Returns**

A Pin

---

## [Nucleo.A4](#) =>

---

**Call type:**

`Nucleo.A4`

**Returns**

A Pin

---

## [Nucleo.A5](#) =>

---

**Call type:**

`Nucleo.A5`

**Returns**

A Pin

---

## [Nucleo.D0](#) =>

---

**Call type:**

`Nucleo.D0`

**Returns**

A Pin

---

## [Nucleo.D1](#) =>

---

**Call type:**

`Nucleo.D1`

**Returns**

A Pin

---

## [Nucleo.D10](#) =>

---

**Call type:**

`Nucleo.D10`

**Returns**

A Pin

[\(top\)](#)

[\(top\)](#)

[\(top\)](#)

[\(top\)](#)

[\(top\)](#)

[\(top\)](#)

## [Nucleo.D11](#) ⇒

---

**Call type:**

`Nucleo.D11`

**Returns**

A Pin

## [Nucleo.D12](#) ⇒

---

**Call type:**

[\(top\)](#)

`Nucleo.D12`

**Returns**

A Pin

## [Nucleo.D13](#) ⇒

---

**Call type:**

[\(top\)](#)

`Nucleo.D13`

**Returns**

A Pin

## [Nucleo.D14](#) ⇒

---

**Call type:**

[\(top\)](#)

`Nucleo.D14`

**Returns**

A Pin

## [Nucleo.D15](#) ⇒

---

**Call type:**

[\(top\)](#)

`Nucleo.D15`

**Returns**

A Pin

## [Nucleo.D2](#) ⇒

---

**Call type:**

[\(top\)](#)

`Nucleo.D2`

**Returns**

A Pin

## [Nucleo.D3](#) =>

---

**Call type:**

`Nucleo.D3`

**Returns**

A Pin

[\(top\)](#)

## [Nucleo.D4](#) =>

---

**Call type:**

[\(top\)](#)

`Nucleo.D4`

**Returns**

A Pin

[\(top\)](#)

## [Nucleo.D5](#) =>

---

**Call type:**

[\(top\)](#)

`Nucleo.D5`

**Returns**

A Pin

[\(top\)](#)

## [Nucleo.D6](#) =>

---

**Call type:**

[\(top\)](#)

`Nucleo.D6`

**Returns**

A Pin

[\(top\)](#)

## [Nucleo.D7](#) =>

---

**Call type:**

[\(top\)](#)

`Nucleo.D7`

**Returns**

A Pin

[\(top\)](#)

## [Nucleo.D8](#) =>

---

**Call type:**

[\(top\)](#)

`Nucleo.D8`

**Returns**

A Pin

[\(top\)](#)

**Call type:**

([top](#))

**Nucleo.D9**

**Returns**

A Pin

---

## [Number Class](#)

---

This is the built-in JavaScript class for numbers.

[\(top\)](#)

### Methods and Fields

- [Number.MAX\\_VALUE](#)
- [Number.MIN\\_VALUE](#)
- [Number.NaN](#)
- [Number.NEGATIVE\\_INFINITY](#)
- [constructor Number\(value, ...\)](#)
- [Number.POSITIVE\\_INFINITY](#)
- [function Number.toFixed\(decimalPlaces\)](#)

---

### [Number.MAX\\_VALUE](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

`Number.MAX_VALUE`

#### Returns

Maximum representable value

---

### [Number.MIN\\_VALUE](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

`Number.MIN_VALUE`

#### Returns

Smallest representable value

---

### [Number.NaN](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

`Number.NaN`

#### Returns

Not a Number

---

### [Number.NEGATIVE\\_INFINITY](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

`Number.NEGATIVE_INFINITY`

#### Returns

Negative Infinity (-1/0)

---

## [constructor Number](#) ⇒

[View MDN documentation](#)

**Call type:**

`new Number(value, ...)`

**Parameters**

`value, ...` - A single value to be converted to a number

**Returns**

A Number object

**Description**

Creates a number

---

## [Number.POSITIVE\\_INFINITY](#) ⇒

[View MDN documentation](#)

**Call type:**

`Number.POSITIVE_INFINITY`

**Returns**

Positive Infinity (1/0)

---

## [function Number.toFixed](#) ⇒

[View MDN documentation](#)

**Call type:**

`function Number.toFixed(decimalPlaces)`

**Parameters**

`decimalPlaces` - A number between 0 and 20 specifying the number of decimal digits after the decimal point

**Returns**

A string

**Description**

Format the number as a fixed point number

[\(top\)](#)

---

## [Object Class](#)

---

This is the built-in class for Objects

[\(top\)](#)

### Methods and Fields

- [Object.assign\(args,...\)](#)
- [function Object.clone\(\)](#)
- [Object.create\(proto,propertiesObject\)](#)
- [Object.defineProperties\(obj,props\)](#)
- [Object.defineProperty\(obj, name, desc\)](#)
- [function Object.emit\(event, args,...\)](#)
- [Object.entries\(object\)](#)
- [Object.fromEntries\(entries\)](#)
- [Object.getOwnPropertyDescriptor\(obj, name\)](#)
- [Object.getOwnPropertyDescriptors\(obj\)](#)
- [Object.getOwnPropertyNames\(object\)](#)
- [Object.getPrototypeOf\(object\)](#)
- [function Object.hasOwnProperty\(name\)](#)
- [Object.keys\(object\)](#)
- [property Object.length](#)
- [constructor Object\(value\)](#)
- [function Object.on\(event, listener\)](#)
- [function Object.removeAllListeners\(event\)](#)
- [function Object.removeListener\(event, listener\)](#)
- [Object.setPrototypeOf\(object,prototype\)](#)
- [function Object.toString\(radix\)](#)
- [function Object.valueOf\(\)](#)
- [Object.values\(object\)](#)

---

### [Object.assign](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

`Object.assign(args, ...)`

#### Parameters

`args, ...` - The target object, then any items objects to use as sources of keys

#### Returns

The target object

#### Description

Appends all keys and values in any subsequent objects to the first object

**Note:** Unlike the standard ES6 [Object.assign](#), this will throw an exception if given raw strings, bools or numbers rather than objects.

---

### [function Object.clone](#) ⇒

---

#### Call type:

[\(top\)](#)

`function Object.clone()`

#### Returns

A copy of this Object

#### Description

Copy this object completely

## [Object.create](#) ⇒

---

[View MDN documentation](#)

**Call type:**

`Object.create(proto, propertiesObject)`

**Parameters**

`proto` - A prototype object

`propertiesObject` - An object containing properties. NOT IMPLEMENTED

**Returns**

A new object

**Description**

Creates a new object with the specified prototype object and properties. properties are currently unsupported.

## [Object.defineProperties](#) ⇒

---

[View MDN documentation](#)

**Call type:**

`Object.defineProperties(obj, props)`

**Parameters**

`obj` - An object

`props` - An object whose fields represent property names, and whose values are property descriptors.

**Returns**

The object, obj.

**Description**

Adds new properties to the Object. See [Object.defineProperty](#) for more information

## [Object.defineProperty](#) ⇒

---

[View MDN documentation](#)

**Call type:**

`Object.defineProperty(obj, name, desc)`

**Parameters**

`obj` - An object

`name` - The name of the property

`desc` - The property descriptor

**Returns**

The object, obj.

[\(top\)](#)

## Description

Add a new property to the Object. 'Desc' is an object with the following fields:

- `configurable` (bool = false) - can this property be changed/deleted (not implemented)
- `enumerable` (bool = false) - can this property be enumerated (not implemented)
- `value` (anything) - the value of this property
- `writable` (bool = false) - can the value be changed with the assignment operator?
- `get` (function) - the getter function, or undefined if no getter (only supported on some platforms)
- `set` (function) - the setter function, or undefined if no setter (only supported on some platforms)

**Note:** `configurable`, `enumerable` and `writable` are not implemented and will be ignored.

## [function Object.emit](#) ⇒

---

Call type:

[\(top\)](#)

```
function Object.emit(event, args, ...)
```

### Parameters

`event` - The name of the event, for instance 'data'

`args, ...` - Optional arguments

## Description

Call any event listeners that were added to this object with [Object.on](#), for instance `obj.emit('data', 'Foo')`.

For more information see [Object.on](#)

**Note:** This is not available in Embeddable Espruino C builds

## [Object.entries](#) ⇒

---

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
Object.entries(object)
```

### Parameters

`object` - The object to return values for

### Returns

An array of `[key, value]` pairs - one for each key on the given object

## Description

Return all enumerable keys and values of the given object

**Note:** This is not available in devices with low flash memory

## [Object.fromEntries](#) ⇒

---

[View MDN documentation](#)

Call type:

[\(top\)](#)

```
Object.fromEntries(entries)
```

### Parameters

`entries` - An array of `[key, value]` pairs to be used to create an object

## Returns

An object containing all the specified pairs

## Description

Transforms an array of key-value pairs into an object

**Note:** This is not available in devices with low flash memory

---

## [Object.getOwnPropertyDescriptor](#) ⇒

[View MDN documentation](#)

### Call type:

```
Object.getOwnPropertyDescriptor(obj, name)
```

### Parameters

`obj` - The object

`name` - The name of the property

## Returns

An object with a description of the property. The values of writable/enumerable/configurable may not be entirely correct due to Espruino's implementation.

## Description

Get information on the given property in the object, or undefined

---

## [Object.getOwnPropertyDescriptors](#) ⇒

[View MDN documentation](#)

### Call type:

```
Object.getOwnPropertyDescriptors(obj)
```

### Parameters

`obj` - The object

## Returns

An object containing all the property descriptors of an object

## Description

Get information on all properties in the object (from [Object.getOwnPropertyDescriptor](#)), or just {} if no properties

**Note:** This is not available in devices with low flash memory

---

## [Object.getOwnPropertyNames](#) ⇒

[View MDN documentation](#)

### Call type:

```
Object.getOwnPropertyNames(object)
```

### Parameters

`object` - The Object to return a list of property names for

[\(top\)](#)

## Returns

An array of the Object's own properties

## Description

Returns an array of all properties (enumerable or not) found directly on a given object.

## [Object.getPrototypeOf](#) ⇒

---

[View MDN documentation](#)

### Call type:

```
Object.getPrototypeOf(object)
```

### Parameters

`object` - An object

### Returns

The prototype

## Description

Get the prototype of the given object - this is like writing `object.__proto__` but is the 'proper' ES6 way of doing it

## [function Object.hasOwnProperty](#) ⇒

---

[View MDN documentation](#)

### Call type:

```
function Object.hasOwnProperty(name)
```

### Parameters

`name` - The name of the property to search for

### Returns

True if it exists, false if it doesn't

## Description

Return true if the object (not its prototype) has the given property.

NOTE: This currently returns false-positives for built-in functions in prototypes

## [Object.keys](#) ⇒

---

[View MDN documentation](#)

### Call type:

```
Object.keys(object)
```

### Parameters

`object` - The object to return keys for

### Returns

[\(top\)](#)

An array of strings - one for each key on the given object

## Description

Return all enumerable keys of the given object

### [property Object.length](#) ⇒

---

#### Call type:

[\(top\)](#)

`property Object.length`

#### Returns

The length of the object

## Description

Find the length of the object

### [constructor Object](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

`new Object(value)`

#### Parameters

`value` - A single value to be converted to an object

#### Returns

An Object

## Description

Creates an Object from the supplied argument

### [function Object.on](#) ⇒

---

#### Call type:

[\(top\)](#)

`function Object.on(event, listener)`

#### Parameters

`event` - The name of the event, for instance 'data'

`listener` - The listener to call when this event is received

## Description

Register an event listener for this object, for instance

```
Serial1.on('data',
 function(d) {...})
```

This is the same as Node.js's [EventEmitter](#) but on Espruino the functionality is built into every object:

- [Object.on](#)
- [Object.emit](#)
- [Object.removeListener](#)

- [Object.removeAllListeners](#)

```
var o = {};
// o can be any object...
// call an arrow function when the 'answer' event is received
o.on('answer', x => console.log(x));
// call a named function when the 'answer' event is received
function printAnswer(d) {
 console.log("The answer is", d);
}
o.on('answer', printAnswer);
// emit the 'answer' event - functions added with 'on' will be executed
o.emit('answer', 42);
// prints: 42
// prints: The answer is 42
// If you have a named function, it can be removed by name
o.removeListener('answer', printAnswer);
// Now 'printAnswer' is removed
o.emit('answer', 43);
// prints: 43
// Or you can remove all listeners for 'answer'
o.removeAllListeners('answer')
// Now nothing happens
o.emit('answer', 44);
// nothing printed
```

If you have more than one handler for an event, and you'd like that handler to stop the event being passed to other handlers then you can call [E.stopPropagation\(\)](#) in that handler.

**Note:** This is not available in Embeddable Espruino C builds

## [function Object.removeAllListeners](#) ⇒

---

Call type:

[\(top\)](#)

```
function Object.removeAllListeners(event)
```

Parameters

**event** - [optional] The name of the event, for instance '`data`'. If not specified *all* listeners are removed.

Description

Removes all listeners (if `event==undefined`), or those of the specified event.

```
Serial1.on("data", function(data) { ... });
Serial1.removeAllListeners("data");
// or
Serial1.removeAllListeners(); // removes all listeners for all event types
```

For more information see [Object.on](#)

## [function Object.removeListener](#) ⇒

---

Call type:

[\(top\)](#)

```
function Object.removeListener(event, listener)
```

Parameters

**event** - The name of the event, for instance 'data'

**listener** - The listener to remove

Description

Removes the specified event listener.

```
function foo(d) {
 console.log(d);
}
Serial1.on("data", foo);
Serial1.removeListener("data", foo);
```

For more information see [Object.on](#)

## [Object.setPrototypeOf](#) ⇒

---

[View MDN documentation](#)

## Call type:

`Object.setPrototypeOf(object, prototype)`

## Parameters

`object` - An object

`prototype` - The prototype to set on the object

## Returns

The object passed in

## Description

Set the prototype of the given object - this is like writing

`object.__proto__ = prototype`

but is the 'proper' ES6 way of doing it

---

## [function Object.toString](#) ⇒

[View MDN documentation](#)

## Call type:

`function Object.toString(radix)`

## Parameters

`radix` - [optional] If the object is an integer, the radix (between 2 and 36) to use. NOTE: Setting a radix does not work on floating point numbers.

## Returns

A String representing the object

## Description

Convert the Object to a string

---

## [function Object.valueOf](#) ⇒

[View MDN documentation](#)

## Call type:

`function Object.valueOf()`

## Returns

The primitive value of this object

## Description

Returns the primitive value of this object.

---

## [Object.values](#) ⇒

[View MDN documentation](#)

(top)

**Call type:**[\(top\)](#)

```
Object.values(object)
```

**Parameters**

**object** - The object to return values for

**Returns**

An array of values - one for each key on the given object

**Description**

Return all enumerable values of the given object

**Note:** This is not available in devices with low flash memory

---

## [OneWire Class](#)

---

This class provides a software-defined OneWire master. It is designed to be similar to Arduino's OneWire library.

[\(top\)](#)

**Note:** OneWire commands are very timing-sensitive, and on nRF52 devices (Bluetooth LE Espruino boards) the bluetooth stack can get in the way. Before version 2v18 of Espruino OneWire could be unreliable, but as of firmware 2v18 Espruino now schedules OneWire accesses with the bluetooth stack to ensure it doesn't interfere. OneWire is now reliable but some functions such as [OneWire.search](#) can now take a while to execute (around 1 second).

### Methods and Fields

- [constructor OneWire\(pin\)](#)
- [function OneWire.read\(count\)](#)
- [function OneWire.reset\(\)](#)
- [function OneWire.search\(command\)](#)
- [function OneWire.select\(rom\)](#)
- [function OneWire.skip\(\)](#)
- [function OneWire.write\(data\[, power\]\)](#)

---

### [constructor OneWire](#) ⇒

---

#### Call type:

[\(top\)](#)

```
new OneWire(pin)
```

#### Parameters

pin - The pin to implement OneWire on

#### Returns

A OneWire object

#### Description

Create a software OneWire implementation on the given pin

---

### [function OneWire.read](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function OneWire.read(count)
```

#### Parameters

count - [optional] The amount of bytes to read

#### Returns

The byte that was read, or a Uint8Array if count was specified and >=0

#### Description

Read a byte

---

### [function OneWire.reset](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function OneWire.reset()
```

## Returns

True is a device was present (it held the bus low)

## Description

Perform a reset cycle

---

## [function OneWire.search](#) ⇒

### Call type:

[\(top\)](#)

```
function OneWire.search(command)
```

### Parameters

command - (Optional) command byte. If not specified (or zero), this defaults to 0xF0. This can be set to 0xEC to perform a DS18B20 'Alarm Search' Command'

## Returns

An array of devices that were found

## Description

Search for devices

---

## [function OneWire.select](#) ⇒

### Call type:

[\(top\)](#)

```
function OneWire.select(rom)
```

### Parameters

rom - The device to select (get this using [OneWire.search\(\)](#))

## Description

Select a ROM - always performs a reset first

---

## [function OneWire.skip](#) ⇒

### Call type:

[\(top\)](#)

```
function OneWire.skip()
```

## Description

Skip a ROM

---

## [function OneWire.write](#) ⇒

### Call type:

[\(top\)](#)

```
function OneWire.write(data, power)
```

### Parameters

data - A byte (or array of bytes) to write

power - Whether to leave power on after write (default is false)

**Description**

Write one or more bytes

## [Pin Class](#)

---

This is the built-in class for Pins, such as D0,D1,LED1, or BTN

[\(top\)](#)

You can call the methods on Pin, or you can use Wiring-style functions such as digitalWrite

### Methods and Fields

- [function Pin.getInfo\(\)](#)
- [function Pin.getMode\(\)](#)
- [function Pin.mode\(mode\)](#)
- [Pin.Pin\(\)](#)
- [constructor Pin\(value\)](#)
- [function Pin.read\(\)](#)
- [function Pin.reset\(\)](#)
- [function Pin.set\(\)](#)
- [function Pin.toggle\(\)](#)
- [function Pin.write\(value\)](#)
- [function Pin.writeAtTime\(value,time\)](#)

### [function Pin.getInfo](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function Pin.getInfo()
```

#### Returns

An object containing information about this pins

#### Description

Get information about this pin and its capabilities. Of the form:

```
{
 "port" : "A", // the Pin's port on the chip
 "num" : 12, // the Pin's number
 "in_addr" : 0x..., // (if available) the address of the pin's input address in bit-banded memory (can be used with peek)
 "out_addr" : 0x..., // (if available) the address of the pin's output address in bit-banded memory (can be used with poke)
 "analog" : { ADCs : [1], channel : 12 }, // If analog input is available
 "functions" : {
 "TIM1":{type:"CH1, af:0},
 "I2C3":{type:"SCL", af:1}
 }
}
```

Will return undefined if pin is not valid.

**Note:** This is not available in devices with low flash memory

### [function Pin.getMode](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function Pin.getMode()
```

#### Returns

The pin mode, as a string

#### Description

Return the current mode of the given pin. See [pinMode](#) for more information.

### [function Pin.mode](#) ⇒

---

**Call type:**[\(top\)](#)

```
function Pin.mode(mode)
```

**Parameters**

mode - The mode - a string that is either 'analog', 'input', 'input\_pullup', 'input\_pulldown', 'output', 'opendrain', 'af\_output' or 'af\_opendrain'. Do not include this argument if you want to revert to automatic pin mode setting.

**Description**

Set the mode of the given pin. See [pinMode](#) for more information on pin modes.

[Pin.Pin](#) ⇒**Call type:**[\(top\)](#)

```
Pin.Pin()
```

**Description**

This is the built-in class for Pins, such as D0,D1,LED1, or BTN

You can call the methods on Pin, or you can use Wiring-style functions such as digitalWrite

[constructor Pin](#) ⇒**Call type:**[\(top\)](#)

```
new Pin(value)
```

**Parameters**

value - A value to be converted to a pin. Can be a number, pin, or String.

**Returns**

A Pin object

**Description**

Creates a pin from the given argument (or returns undefined if no argument)

[function Pin.read](#) ⇒**Call type:**[\(top\)](#)

```
function Pin.read()
```

**Returns**

Whether pin is a logical 1 or 0

**Description**

Returns the input state of the pin as a boolean.

**Note:** if you didn't call [pinMode](#) beforehand then this function will also reset the pin's state to "input"

[function Pin.reset](#) ⇒**Call type:**[\(top\)](#)

```
function Pin.reset()
```

## Description

Sets the output state of the pin to a 0

**Note:** if you didn't call `pinMode` beforehand then this function will also reset the pin's state to "output"

---

## [function Pin.set](#) ⇒

### Call type:

[\(top\)](#)

```
function Pin.set()
```

## Description

Sets the output state of the pin to a 1

**Note:** if you didn't call `pinMode` beforehand then this function will also reset the pin's state to "output"

---

## [function Pin.toggle](#) ⇒

### Call type:

[\(top\)](#)

```
function Pin.toggle()
```

## Returns

True if the pin is high after calling the function

## Description

Toggles the state of the pin from off to on, or from on to off.

**Note:** This method doesn't currently work on the ESP8266 port of Espruino.

**Note:** if you didn't call `pinMode` beforehand then this function will also reset the pin's state to "output"

---

## [function Pin.write](#) ⇒

### Call type:

[\(top\)](#)

```
function Pin.write(value)
```

## Parameters

`value` - Whether to set output high (true/1) or low (false/0)

## Description

Sets the output state of the pin to the parameter given

**Note:** if you didn't call `pinMode` beforehand then this function will also reset the pin's state to "output"

---

## [function Pin.writeAtTime](#) ⇒

### Call type:

[\(top\)](#)

```
function Pin.writeAtTime(value, time)
```

## Parameters

`value` - Whether to set output high (true/1) or low (false/0)

`time` - Time at which to write

## Description

Sets the output state of the pin to the parameter given at the specified time.

**Note:** this **doesn't** change the mode of the pin to an output. To do that, you need to use `pin.write(0)` or `pinMode(pin, 'output')` first.

**Note:** This is not available in devices with low flash memory

---

## [Pixl Class](#)

---

Class containing utility functions for [Pixl.js](#)

[\(top\)](#)

### Methods and Fields

- [Pixl.getBatteryPercentage\(\)](#)
- [Pixl.lcdw\(c\)](#)
- [Pixl.menu\(menu\)](#)
- [Pixl.setContrast\(c\)](#)
- [Pixl.setLCDPower\(isOn\)](#)

---

### [Pixl.getBatteryPercentage](#) ⇒

---

#### Call type:

[\(top\)](#)

```
Pixl.getBatteryPercentage()
```

#### Returns

A percentage between 0 and 100

#### Description

DEPRECATED - Please use [E.getBattery\(\)](#) instead.

Return an approximate battery percentage remaining based on a normal CR2032 battery (2.8 - 2.2v)

---

### [Pixl.lcdw](#) ⇒

---

#### Call type:

[\(top\)](#)

```
Pixl.lcdw(c)
```

#### Parameters

c -

#### Description

Writes a command directly to the ST7567 LCD controller

---

### [Pixl.menu](#) ⇒

---

#### Call type:

[\(top\)](#)

```
Pixl.menu(menu)
```

#### Parameters

menu - An object containing name->function mappings to be used in a menu

#### Returns

A menu object with draw, move and **select** functions

#### Description

Display a menu on Pixl.js's screen, and set up the buttons to navigate through it.

DEPRECATED: Use [E.showMenu](#)

## [Pixl.setContrast](#) →

---

**Call type:**

[\(top\)](#)

```
Pixl.setContrast(c)
```

**Parameters**

c - Contrast between 0 and 1

**Description**

Set the LCD's contrast

## [Pixl.setLCDPower](#) →

---

**Call type:**

[\(top\)](#)

```
Pixl.setLCDPower(isOn)
```

**Parameters**

isOn - True if the LCD should be on, false if not

**Description**

This function can be used to turn Pixl.js's LCD off or on.

- With the LCD off, Pixl.js draws around 0.1mA
- With the LCD on, Pixl.js draws around 0.25mA

---

## [process Class](#)

---

This class contains information about Espruino itself

[\(top\)](#)

### Methods and Fields

- [process.env](#)
- [process.memory\(gc\)](#)
- [event\\_process.uncaughtException\(exception\)](#)
- [process.version](#)

---

### [process.env](#) ⇒

---

#### Call type:

[\(top\)](#)

```
process.env
```

#### Returns

An object

#### Description

Returns an Object containing various pre-defined variables.

- VERSION - is the Espruino version
- GIT\_COMMIT - is Git commit hash this firmware was built from
- BOARD - the board's ID (e.g. PUCKJS)
- RAM - total amount of on-chip RAM in bytes
- FLASH - total amount of on-chip flash memory in bytes
- SPIFLASH - (on Bangle.js) total amount of off-chip flash memory in bytes
- HWVERSION - For Puck.js this is the board revision (1, 2, 2.1), or for Bangle.js it's 1 or 2
- STORAGE - memory in bytes dedicated to the [Storage](#) module
- SERIAL - the serial number of this chip
- CONSOLE - the name of the current console device being used ([serial1](#), [USB](#), [Bluetooth](#), etc)
- MODULES - a list of built-in modules separated by commas
- EXPTR - The address of the `exportPtrs` structure in flash (this includes links to built-in functions that compiled JS code needs)
- APP\_RAM\_BASE - On nRF5x boards, this is the RAM required by the Softdevice *if it doesn't exactly match what was allocated*. You can use this to update LD\_APP\_RAM\_BASE in the BOARD.py file

For example, to get a list of built-in modules, you can use `process.env.MODULES.split(',')`

**Note:** `process.env` is not writeable - so as not to waste RAM, the contents are generated on demand. If you need to be able to change them, use `process.env=process.env;` first to ensure the values stay allocated.

---

### [process.memory](#) ⇒

---

#### Call type:

[\(top\)](#)

```
process.memory(gc)
```

#### Parameters

gc - [optional] A boolean. If `undefined` or `true` Garbage collection is performed, if `false` it is not

#### Returns

Information about memory usage

#### Description

Run a Garbage Collection pass, and return an object containing information on memory usage.

- free : Memory that is available to be used (in blocks)
- usage : Memory that has been used (in blocks)

- `total` : Total memory (in blocks)
- `history` : Memory used for command history - that is freed if memory is low. Note that this is INCLUDED in the figure for 'free'
- `gc` : Memory freed during the GC pass
- `gctime` : Time taken for GC pass (in milliseconds)
- `blocksize` : Size of a block (variable) in bytes
- `stackEndAddress` : (on ARM) the address (that can be used with peek/poke/etc) of the END of the stack. The stack grows down, so unless you do a lot of recursion the bytes above this can be used.
- `stackFree` : (on ARM) how many bytes of free execution stack are there at the point of execution.
- `flash_start` : (on ARM) the address of the start of flash memory (usually `0x8000000`)
- `flash_binary_end` : (on ARM) the address in flash memory of the end of Espruino's firmware.
- `flash_code_start` : (on ARM) the address in flash memory of pages that store any code that you save with `save()`.
- `flash_length` : (on ARM) the amount of flash memory this firmware was built for (in bytes). **Note:** Some STM32 chips actually have more memory than is advertised.

Memory units are specified in 'blocks', which are around 16 bytes each (depending on your device). The actual size is available in `blocksize`. See <http://www.espruino.com/Performance> for more information.

**Note:** To find free areas of flash memory, see `require('Flash').getFree()`

## [event process.uncaughtException](#) ⇒

---

Call type:

[\(top\)](#)

```
process.on('uncaughtException', function(exception) { ... });
```

Parameters

`exception` - The uncaught exception

Description

This event is called when an exception gets thrown and isn't caught (e.g. it gets all the way back to the event loop).

You can use this for logging potential problems that might occur during execution when you might not be able to see what is written to the console, for example:

```
var lastError;
process.on('uncaughtException', function(e) {
 lastError=e;
 print(e,e.stack+"\n"+e.stack);
});
function checkError() {
 if (!lastError) return print("No Error");
 print(lastError,lastError.stack+"\n"+lastError.stack);
}
```

**Note:** When this is used, exceptions will cease to be reported on the console - which may make debugging difficult!

## [process.version](#) ⇒

---

Call type:

[\(top\)](#)

```
process.version
```

Returns

The version of Espruino

Description

Returns the version of Espruino as a String

---

## [Promise Class](#)

---

This is the built-in class for ES6 Promises

[\(top\)](#)

### Methods and Fields

- [Promise.all\(promises\)](#)
- [function Promise.catch\(onRejected\)](#)
- [constructor Promise\(executor\)](#)
- [Promise.reject\(promises\)](#)
- [Promise.resolve\(promises\)](#)
- [function Promise.then\(onFulfilled, onRejected\)](#)

### [Promise.all](#) ⇒

---

[View MDN documentation](#)

#### Call type:

`Promise.all(promises)`

#### Parameters

`promises` - An array of promises

#### Returns

A new Promise

#### Description

Return a new promise that is resolved when all promises in the supplied array are resolved.

**Note:** This is not available in devices with low flash memory

### [function Promise.catch](#) ⇒

---

[View MDN documentation](#)

#### Call type:

`function Promise.catch(onRejected)`

#### Parameters

`onRejected` - A callback that is called when this promise is rejected

#### Returns

The original Promise

### [constructor Promise](#) ⇒

---

[View MDN documentation](#)

#### Call type:

`new Promise(executor)`

#### Parameters

[\(top\)](#)

`executor` - A function of the form `function (resolve, reject)`

## Returns

A Promise

## Description

Create a new Promise. The executor function is executed immediately (before the constructor even returns) and

**Note:** This is not available in devices with low flash memory

## [Promise.reject](#) ⇒

---

[View MDN documentation](#)

### Call type:

`Promise.reject(promises)`

### Parameters

`promises` - Data to pass to the `.catch` handler

## Returns

A new Promise

## Description

Return a new promise that is already rejected (at idle it'll call `.catch`)

**Note:** This is not available in devices with low flash memory

## [Promise.resolve](#) ⇒

---

[View MDN documentation](#)

### Call type:

[\(top\)](#)

`Promise.resolve(promises)`

### Parameters

`promises` - Data to pass to the `.then` handler

## Returns

A new Promise

## Description

Return a new promise that is already resolved (at idle it'll call `.then`)

**Note:** This is not available in devices with low flash memory

## [function Promise.then](#) ⇒

---

[View MDN documentation](#)

### Call type:

[\(top\)](#)

`function Promise.then(onFulfilled, onRejected)`

### Parameters

`onFulfilled` - A callback that is called when this promise is resolved

`onRejected` - [optional] A callback that is called when this promise is rejected (or nothing)

## Returns

The original Promise

## [Puck Class](#)

---

Class containing [Puck.js's](#) utility functions.

[\(top\)](#)

### Methods and Fields

- [event Puck.accel\(\)](#)
- [Puck.accel\(\)](#)
- [Puck.accelOff\(\)](#)
- [Puck.accelOn\(samplerate\)](#)
- [Puck.accelRd\(reg\)](#)
- [Puck.accelWr\(reg, data\)](#)
- [Puck.capSense\(tx, rx\)](#)
- [Puck.getBatteryPercentage\(\)](#)
- [Puck.getTemperature\(\)](#)
- [Puck.IR\(data, cathode, anode\)](#)
- [Puck.light\(\)](#)
- [Puck.mag\(\)](#)
- [event Puck.mag\(\)](#)
- [Puck.magOff\(\)](#)
- [Puck.magOn\(samplerate\)](#)
- [Puck.magRd\(reg\)](#)
- [Puck.magTemp\(\)](#)
- [Puck.magWr\(reg, data\)](#)
- [Puck.selfTest\(\)](#)

### [event Puck.accel](#) →

---

#### Call type:

[\(top\)](#)

```
Puck.on('accel', function() { ... });
```

#### Description

Only on Puck.js v2.0

Called after [Puck.accelOn\(\)](#) every time accelerometer data is sampled. There is one argument which is an object of the form `{acc:{x,y,z}, gyro:{x,y,z}}` containing the data.

The data is as it comes off the accelerometer and is not scaled to 1g. For more information see [Puck.accel\(\)](#) or [the Puck.js page on the magnetometer](#).

**Note:** This is only available in Puck.js devices

### [Puck.accel](#) →

---

#### Call type:

[\(top\)](#)

```
Puck.accel()
```

#### Returns

An Object `{acc:{x,y,z}, gyro:{x,y,z}}` of accelerometer/gyro readings

#### Description

Turn on the accelerometer, take a single reading, and then turn it off again.

The values reported are the raw values from the chip. In normal configuration:

- accelerometer: full-scale (32768) is 4g, so you need to divide by 8192 to get correctly scaled values
- gyro: full-scale (32768) is 245 dps, so you need to divide by 134 to get correctly scaled values

If taking more than one reading, we'd suggest you use [Puck.accelOn\(\)](#) and the [Puck.accel](#) event.

**Note:** This is only available in Puck.js devices

## [Puck.accelOff](#) ⇒

---

### Call type:

```
Puck.accelOff()
```

### Description

Turn the accelerometer off after it has been turned on by [Puck.accelOn\(\)](#).

Check out [the Puck.js page on the accelerometer](#) for more information.

**Note:** This is only available in Puck.js devices

[\(top\)](#)

## [Puck.accelOn](#) ⇒

---

### Call type:

```
Puck.accelOn(samplerate)
```

### Parameters

samplerate - The sample rate in Hz, or **undefined** (default is 12.5 Hz)

### Description

Accepted values are:

- 1.6 Hz (no Gyro) - 40uA (2v05 and later firmware)
- 12.5 Hz (with Gyro)- 350uA
- 26 Hz (with Gyro) - 450 uA
- 52 Hz (with Gyro) - 600 uA
- 104 Hz (with Gyro) - 900 uA
- 208 Hz (with Gyro) - 1500 uA
- 416 Hz (with Gyro) (not recommended)
- 833 Hz (with Gyro) (not recommended)
- 1660 Hz (with Gyro) (not recommended)

Once [Puck.accelOn\(\)](#) is called, the [Puck.accel](#) event will be called each time data is received. [Puck.accelOff\(\)](#) can be called to turn the accelerometer off.

For instance to light the red LED whenever Puck.js is face up:

```
Puck.on('accel', function(a) {
 digitalWrite(LED1, a.acc.z > 0);
});
Puck.accelOn();
```

Check out [the Puck.js page on the accelerometer](#) for more information.

**Note:** Puck.js cannot currently read every sample from the accelerometer at sample rates above 208Hz.

**Note:** This is only available in Puck.js devices

[\(top\)](#)

## [Puck.accelRd](#) ⇒

---

### Call type:

```
Puck.accelRd(reg)
```

### Parameters

reg -

### Returns

See description above

### Description

Reads a register from the LSM6DS3TR-C Accelerometer. Can be used for configuring advanced functions.

[\(top\)](#)

Check out [the Puck.js page on the accelerometer](#) for more information and links to modules that use this function.

**Note:** This is only available in Puck.js devices

## [Puck.accelWr](#) →

---

**Call type:**

[\(top\)](#)

```
Puck.accelWr(reg, data)
```

**Parameters**

reg -

data -

**Description**

Writes a register on the LSM6DS3TR-C Accelerometer. Can be used for configuring advanced functions.

Check out [the Puck.js page on the accelerometer](#) for more information and links to modules that use this function.

**Note:** This is only available in Puck.js devices

## [Puck.capSense](#) →

---

**Call type:**

[\(top\)](#)

```
Puck.capSense(tx, rx)
```

**Parameters**

tx -

rx -

**Returns**

Capacitive sense counter

**Description**

Capacitive sense - the higher the capacitance, the higher the number returned.

If called without arguments, a value depending on the capacitance of what is attached to pin D11 will be returned. If you attach a length of wire to D11, you'll be able to see a higher value returned when your hand is near the wire than when it is away.

You can also supply pins to use yourself, however if you do this then the TX pin must be connected to RX pin and sense plate via a roughly 1MOhm resistor.

When not supplying pins, Puck.js uses an internal resistor between D12(tx) and D11(rx).

**Note:** This is only available in Puck.js devices

## [Puck.getBatteryPercentage](#) →

---

**Call type:**

[\(top\)](#)

```
Puck.getBatteryPercentage()
```

**Returns**

A percentage between 0 and 100

**Description**

DEPRECATED - Please use [E.getBattery\(\)](#) instead.

Return an approximate battery percentage remaining based on a normal CR2032 battery (2.8 - 2.2v).

**Note:** This is only available in Puck.js devices

## [Puck.getTemperature](#) ⇒

---

**Call type:**

[\(top\)](#)

```
Puck.getTemperature()
```

**Returns**

Temperature in degrees C

**Description**

On Puck.js v2.0 this will use the on-board PCT2075TP temperature sensor, but on Puck.js the less accurate on-chip Temperature sensor is used.

**Note:** This is only available in Puck.js devices

## [Puck.IR](#) ⇒

---

**Call type:**

[\(top\)](#)

```
Puck.IR(data, cathode, anode)
```

**Parameters**

`data` - An array of pulse lengths, in milliseconds

`cathode` - [optional] pin to use for IR LED cathode - if not defined, the built-in IR LED is used

`anode` - [optional] pin to use for IR LED anode - if not defined, the built-in IR LED is used

**Description**

Transmit the given set of IR pulses - data should be an array of pulse times in milliseconds (as `[on, off, on, off, on, etc]`).

For example `Puck.IR(pulseTimes)` - see <http://www.espruino.com/Puck.js+Infrared> for a full example.

You can also attach an external LED to Puck.js, in which case you can just execute `Puck.IR(pulseTimes, led_cathode, led_anode)`

It is also possible to just supply a single pin for IR transmission with `Puck.IR(pulseTimes, led_anode)` (on 2v05 and above).

**Note:** This is only available in Puck.js devices

## [Puck.light](#) ⇒

---

**Call type:**

[\(top\)](#)

```
Puck.light()
```

**Returns**

A light value from 0 to 1

**Description**

Return a light value based on the light the red LED is seeing.

**Note:** If called more than 5 times per second, the received light value may not be accurate.

**Note:** This is only available in Puck.js devices

## [Puck.mag](#) ⇒

---

## Call type:

[\(top\)](#)

```
Puck.mag()
```

## Returns

An Object `{x,y,z}` of magnetometer readings as integers

## Description

Turn on the magnetometer, take a single reading, and then turn it off again.

An object of the form `{x,y,z}` is returned containing magnetometer readings. Due to residual magnetism in the Puck and magnetometer itself, with no magnetic field the Puck will not return `{x:0,y:0,z:0}`.

Instead, it's up to you to figure out what the 'zero value' is for your Puck in your location and to then subtract that from the value returned. If you're not trying to measure the Earth's magnetic field then it's a good idea to just take a reading at startup and use that.

With the aerial at the top of the board, the y reading is vertical, x is horizontal, and z is through the board.

Readings are in increments of 0.1 micro Tesla (uT). The Earth's magnetic field varies from around 25-60 uT, so the reading will vary by 250 to 600 depending on location.

**Note:** This is only available in Puck.js devices

---

## [event Puck.mag](#) ⇒

## Call type:

[\(top\)](#)

```
Puck.on('mag', function() { ... });
```

## Description

Called after `Puck.magOn()` every time magnetometer data is sampled. There is one argument which is an object of the form `{x,y,z}` containing magnetometer readings as integers (for more information see [Puck.mag\(\)](#)).

Check out [the Puck.js page on the magnetometer](#) for more information.

**Note:** This is only available in Puck.js devices

---

## [Puck.magOff](#) ⇒

## Call type:

[\(top\)](#)

```
Puck.magOff()
```

## Description

Turn the magnetometer off

**Note:** This is only available in Puck.js devices

---

## [Puck.magOn](#) ⇒

## Call type:

[\(top\)](#)

```
Puck.magOn(samplerate)
```

## Parameters

`samplerate` - The sample rate in Hz, or undefined

## Description

Turn the magnetometer on and start periodic sampling. Samples will then cause a 'mag' event on 'Puck':

```
Puck.magOn();
Puck.on('mag', function(xyz) {
```

```
console.log(xyz);
// {x:..., y:..., z:...}
});
// Turn events off with Puck.magOff();
```

This call will be ignored if the sampling is already on.

If given an argument, the sample rate is set (if not, it's at 0.63 Hz). The sample rate must be one of the following (resulting in the given power consumption):

- 80 Hz - 900uA
- 40 Hz - 550uA
- 20 Hz - 275uA
- 10 Hz - 137uA
- 5 Hz - 69uA
- 2.5 Hz - 34uA
- 1.25 Hz - 17uA
- 0.63 Hz - 8uA
- 0.31 Hz - 8uA
- 0.16 Hz - 8uA
- 0.08 Hz - 8uA

When the battery level drops too low while sampling is turned on, the magnetometer may stop sampling without warning, even while other Puck functions continue uninterrupted.

Check out [the Puck.js page on the magnetometer](#) for more information.

**Note:** This is only available in Puck.js devices

## [Puck.magRd](#) ⇒

---

**Call type:**

[\(top\)](#)

**Puck.magRd(*reg*)**

**Parameters**

*reg* -

**Returns**

See description above

**Description**

Reads a register from the LIS3MDL / MAX3110 Magnetometer. Can be used for configuring advanced functions.

Check out [the Puck.js page on the magnetometer](#) for more information and links to modules that use this function.

**Note:** This is only available in Puck.js devices

## [Puck.magTemp](#) ⇒

---

**Call type:**

[\(top\)](#)

**Puck.magTemp()**

**Returns**

Temperature in degrees C

**Description**

Turn on the magnetometer, take a single temperature reading from the MAG3110 chip, and then turn it off again.

(If the magnetometer is already on, this just returns the last reading obtained)

[E.getTemperature\(\)](#) uses the microcontroller's temperature sensor, but this uses the magnetometer's.

The reading obtained is an integer (so no decimal places), but the sensitivity is factory trimmed. to 1°C, however the temperature offset isn't - so absolute readings may still need calibrating.

**Note:** This is only available in Puck.js devices

## [Puck.magWr](#) ⇒

---

### Call type:

[\(top\)](#)

```
Puck.magWr(reg, data)
```

### Parameters

reg -

data -

### Description

Writes a register on the LIS3MDL / MAX3110 Magnetometer. Can be used for configuring advanced functions.

Check out [the Puck.js page on the magnetometer](#) for more information and links to modules that use this function.

**Note:** This is only available in Puck.js devices

## [Puck.selfTest](#) ⇒

---

### Call type:

[\(top\)](#)

```
Puck.selfTest()
```

### Returns

True if the self-test passed

### Description

Run a self-test, and return true for a pass. This checks for shorts between pins, so your Puck shouldn't have anything connected to it.

**Note:** This self-test auto starts if you hold the button on your Puck down while inserting the battery, leave it pressed for 3 seconds (while the green LED is lit) and release it soon after all LEDs turn on. 5 red blinks is a fail, 5 green is a pass.

If the self test fails, it'll set the Puck.js Bluetooth advertising name to `Puck.js !ERR` where ERR is a 3 letter error code.

**Note:** This is only available in Puck.js devices

---

## [ReferenceError Class](#)

---

The base class for reference errors - where a variable which doesn't exist has been accessed.

([top](#))

### Methods and Fields

- [constructor ReferenceError\(message\)](#)
- [function ReferenceError.toString\(\)](#)

#### [constructor ReferenceError](#) ⇒

---

[View MDN documentation](#)

##### Call type:

`new ReferenceError(message)`

##### Parameters

`message` - [optional] An message string

##### Returns

A ReferenceError object

##### Description

Creates a ReferenceError object

#### [function ReferenceError.toString](#) ⇒

---

##### Call type:

`function ReferenceError.toString()`

##### Returns

A String

## [RegExp Class](#)

---

The built-in class for handling Regular Expressions

[\(top\)](#)

**Note:** Espruino's regular expression parser does not contain all the features present in a full ES6 JS engine. However it does contain support for the all the basics.

### Methods and Fields

- [function RegExp.exec\(str\)](#)
- [constructor RegExp\(regex, flags\)](#)
- [function RegExp.test\(str\)](#)

#### [function RegExp.exec](#) ⇒

---

[View MDN documentation](#)

##### Call type:

```
function RegExp.exec(str)
```

##### Parameters

str - A string to match on

##### Returns

A result array, or null

##### Description

Test this regex on a string - returns a result array on success, or `null` otherwise.

`/Wo/.exec("Hello World")` will return:

```
[
 "Wo",
 "index": 6,
 "input": "Hello World"
]
```

Or with groups `/W(o)rld/.exec("Hello World")` returns:

```
[
 "World",
 "o", "index": 6,
 "input": "Hello World"
]
```

**Note:** This is not available in devices with low flash memory

#### [constructor RegExp](#) ⇒

---

[View MDN documentation](#)

##### Call type:

[\(top\)](#)

```
new RegExp(regex, flags)
```

##### Parameters

regex - A regular expression as a string

flags - Flags for the regular expression as a string

##### Returns

A RegExp object

## Description

Creates a RegExp object, for handling Regular Expressions

**Note:** This is not available in devices with low flash memory

## [function RegExp.test](#) ⇒

---

[View MDN documentation](#)

### Call type:

[\(top\)](#)

```
function RegExp.test(str)
```

### Parameters

str - A string to match on

### Returns

true for a match, or false

## Description

Test this regex on a string - returns **true** on a successful match, or **false** otherwise

**Note:** This is not available in devices with low flash memory

## [Serial Class](#)

---

This class allows use of the built-in USARTs

(top)

Methods may be called on the [USB](#), [Serial1](#), [Serial2](#), [Serial3](#), [Serial4](#), [Serial5](#) and [Serial6](#) objects. While different processors provide different numbers of USARTs, on official Espruino boards you can always rely on at least [Serial1](#) being available

### Instances

- [Bluetooth](#) The Bluetooth Serial port - used when data is sent or received over Bluetooth
- [LoopbackA](#) A loopback serial device. Data sent to [LoopbackA](#) comes out of [LoopbackB](#) and
- [LoopbackB](#) A loopback serial device. Data sent to [LoopbackB](#) comes out of [LoopbackA](#) and
- [Serial1](#) The first Serial (USART) port
- [Serial2](#) The second Serial (USART) port
- [Serial3](#) The third Serial (USART) port
- [Serial4](#) The fourth Serial (USART) port
- [Serial5](#) The fifth Serial (USART) port
- [Serial6](#) The sixth Serial (USART) port
- [Telnet](#) A telnet serial device that maps to the built-in telnet console server (devices)
- [Terminal](#) A simple VT100 terminal emulator.
- [USB](#) The USB Serial port

### Methods and Fields

- [function Serial.available\(\)](#)
- [event Serial.data\(data\)](#)
- [Serial.find\(pin\)](#)
- [function Serial.flush\(\)](#)
- [event Serial.framing\(\)](#)
- [function Serial.inject\(data, ...\)](#)
- [event Serial.parity\(\)](#)
- [function Serial.pipe\(destination, options\)](#)
- [function Serial.print\(string\)](#)
- [function Serial.println\(string\)](#)
- [function Serial.read\(chars\)](#)
- [constructor Serial\(\)](#)
- [function Serial.setConsole\(force\)](#)
- [function Serial.setup\(baudrate, options\)](#)
- [function Serial.unsetup\(\)](#)
- [function Serial.write\(data, ...\)](#)

### [function Serial.available](#) ⇒

---

#### Call type:

(top)

```
function Serial.available()
```

#### Returns

How many bytes are available

#### Description

Return how many bytes are available to read. If there is already a listener for data, this will always return 0.

### [event Serial.data](#) ⇒

---

#### Call type:

(top)

```
Serial.on('data', function(data) { ... });
```

#### Parameters

`data` - A string containing one or more characters of received data

## Description

The `data` event is called when data is received. If a handler is defined with `x.on('data', function(data) { ... })` then it will be called, otherwise data will be stored in an internal buffer, where it can be retrieved with `x.read()`

## [Serial.find](#) ⇒

---

### Call type:

[\(top\)](#)

```
Serial.find(pin)
```

### Parameters

`pin` - A pin to search with

### Returns

An object of type [Serial](#), or `undefined` if one couldn't be found.

## Description

Try and find a USART (Serial) hardware device that will work on this pin (e.g. [Serial1](#))

May return undefined if no device can be found.

## [function Serial.flush](#) ⇒

---

### Call type:

[\(top\)](#)

```
function Serial.flush()
```

### Description

Flush this serial stream (pause execution until all data has been sent)

**Note:** This is not available in devices with low flash memory

## [event Serial.framing](#) ⇒

---

### Call type:

[\(top\)](#)

```
Serial.on('framing', function() { ... });
```

### Description

The `framing` event is called when there was activity on the input to the UART but the `STOP` bit wasn't in the correct place. This is either because there was noise on the line, or the line has been pulled to 0 for a long period of time.

To enable this, you must initialise Serial with

```
SerialX.setup(..., { ...,
errors:true });
```

**Note:** Even though there was an error, the byte will still be received and passed to the `data` handler.

**Note:** This only works on STM32 and NRF52 based devices (e.g. all official Espruino boards)

## [function Serial.inject](#) ⇒

---

### Call type:

[\(top\)](#)

```
function Serial.inject(data, ...)
```

### Parameters

`data, ...` - One or more items to write. May be ints, strings, arrays, or special objects (see [E.toInt8Array](#) for more info).

## Description

Add data to this device as if it came directly from the input - it will be returned via `serial.on('data', ...)`:

```
Serial1.on('data', function(d) { print("Got",d); });
Serial1.inject('Hello World');
// prints "Got Hel","Got lo World" (characters can be split over multiple callbacks)
```

This is most useful if you wish to send characters to Espruino's REPL (console) while it is on another device.

**Note:** This is not available in devices with low flash memory

## [event Serial.parity](#) ⇒

---

### Call type:

([top](#)).

```
Serial.on('parity', function() { ... });
```

## Description

The `parity` event is called when the UART was configured with a parity bit, and this doesn't match the bits that have actually been received.

To enable this, you must initialise Serial with

```
SerialX.setup(..., { ...,
errors:true });
```

**Note:** Even though there was an error, the byte will still be received and passed to the data handler.

**Note:** This only works on STM32 and NRF52 based devices (e.g. all official Espruino boards)

## [function Serial.pipe](#) ⇒

---

### Call type:

([top](#)).

```
function Serial.pipe(destination, options)
```

## Parameters

`destination` - The destination file/stream that will receive content from the source.

`options` - [optional] An object { `chunkSize : int=32`, `end : bool=true`, `complete : function` }

`chunkSize` : The amount of data to pipe from source to destination at a time

`complete` : a function to call when the pipe activity is complete

`end` : call the 'end' function on the destination when the source is finished

## Description

Pipe this USART to a stream (an object with a 'write' method)

**Note:** This is not available in devices with low flash memory

## [function Serial.print](#) ⇒

---

### Call type:

([top](#)).

```
function Serial.print(string)
```

## Parameters

`string` - A String to print

## Description

Print a string to the serial port - without a line feed

**Note:** This function replaces any occurrences of \n in the string with \r\n. To avoid this, use [Serial.write](#).

## [function Serial.println](#) ⇒

---

Call type:

[\(top\)](#)

```
function Serial.println(string)
```

Parameters

**string** - A String to print

Description

Print a line to the serial port with a newline (\r\n) at the end of it.

**Note:** This function converts data to a string first, e.g. `Serial.print([1,2,3])` is equivalent to

```
Serial.print("1,2,3"). If you'd like
to write raw bytes, use
```

```
Serial.write`.
```

## [function Serial.read](#) ⇒

---

Call type:

[\(top\)](#)

```
function Serial.read(chars)
```

Parameters

**chars** - The number of characters to read, or undefined/0 for all available

Returns

A string containing the required bytes.

Description

Return a string containing characters that have been received

## [constructor Serial](#) ⇒

---

Call type:

[\(top\)](#)

```
new Serial()
```

Returns

A Serial object

Description

Create a software Serial port. This has limited functionality (only low baud rates), but it can work on any pins.

Use `Serial.setup` to configure this port.

## [function Serial.setConsole](#) ⇒

---

Call type:

[\(top\)](#)

```
function Serial.setConsole(force)
```

Parameters

**force** - Whether to force the console to this port

## Description

Set this Serial port as the port for the JavaScript console (REPL).

Unless `force` is set to true, changes in the connection state of the board (for instance plugging in USB) will cause the console to change.

See [E.setConsole](#) for a more flexible version of this function.

## [function Serial.setup](#) ⇒

---

### Call type:

(top)

```
function Serial.setup(baudrate, options)
```

### Parameters

`baudrate` - The baud rate - the default is 9600

`options` - [optional] A structure containing extra information on initialising the serial port - see below.

### Description

Setup this Serial port with the given baud rate and options.

e.g.

```
Serial1.setup(9600,{rx:a_pin, tx:a_pin});
```

The second argument can contain:

```
{
 rx:pin, // Receive pin (data in to Espruino)
 tx:pin, // Transmit pin (data out of Espruino)
 ck:pin, // (default none) Clock Pin
 cts:pin, // (default none) Clear to Send Pin
 bytesize:8, // (default 8) How many data bits - 7 or 8
 parity:null/'none'/'o'/'odd'/'e'/'even',
 // (default none) Parity bit
 stopbits:1, // (default 1) Number of stop bits to use
 flow:null/undefined/'none'/'xon', // (default none) software flow control
 path:null/undefined/string // Linux Only - the path to the Serial device to use
 errors:false // (default false) whether to forward framing/parity errors
}
```

You can find out which pins to use by looking at [your board's reference page](#) and searching for pins with the `UART/USART` markers.

If not specified in options, the default pins are used for rx and tx (usually the lowest numbered pins on the lowest port that supports this peripheral). `ck` and `cts` are not used unless specified.

Note that even after changing the RX and TX pins, if you have called setup before then the previous RX and TX pins will still be connected to the Serial port as well - until you set them to something else using [digitalWrite](#) or [pinMode](#).

Flow control can be xOn/xOff (`flow:'xon'`) or hardware flow control (receive only) if `cts` is specified. If `cts` is set to a pin, the pin's value will be 0 when Espruino is ready for data and 1 when it isn't.

By default, framing or parity errors don't create `framing` or `parity` events on the `Serial` object because storing these errors uses up additional storage in the queue. If you're intending to receive a lot of malformed data then the queue might overflow `E.getErrorFlags()` would return `FIFO_FULL`. However if you need to respond to `framing` or `parity` errors then you'll need to use `errors:true` when initialising serial.

On Linux builds there is no default Serial device, so you must specify a path to a device - for instance: `Serial1.setup(9600,{path:"/dev/ttyACM0"})`

You can also set up 'software serial' using code like:

```
var s = new Serial();
s.setup(9600,{rx:a_pin, tx:a_pin});
```

However software serial doesn't use `ck`, `cts`, `parity`, `flow` or `errors` parts of the initialisation object.

## [function Serial.unsetup](#) ⇒

---

### Call type:

(top)

```
function Serial.unsetup()
```

### Description

If the serial (or software serial) device was set up, uninitialise it.

**Note:** This is not available in devices with low flash memory

## [function Serial.write](#) ⇒

---

### Call type:

[\(top\)](#)

```
function Serial.write(data, ...)
```

### Parameters

`data, ...` - One or more items to write. May be ints, strings, arrays, or special objects (see [E.toInt8Array](#) for more info).

### Description

Write a character or array of data to the serial port

This method writes unmodified data, e.g. `Serial.write([1,2,3])` is equivalent to `Serial.write("\1\2\3")`. If you'd like data converted to a string first, use [Serial.print](#).

---

## [Server Class](#)

---

The socket server created by `require('net').createServer`

[\(top\)](#)

### Methods and Fields

- [function Server.close\(\)](#)
- [function Server.listen\(port\)](#)

---

### [function Server.close](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function Server.close()
```

#### Description

Stop listening for new connections

---

### [function Server.listen](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function Server.listen(port)
```

#### Parameters

`port` - The port to listen on

#### Returns

The HTTP server instance that 'listen' was called on

#### Description

Start listening for new connections on the given port

---

## [Socket Class](#)

---

An actual socket connection - allowing transmit/receive of TCP data

[\(top\)](#)

### Methods and Fields

- [function Socket.available\(\)](#)
- [event Socket.close\(had\\_error\)](#)
- [event Socket.data\(data\)](#)
- [event Socket.drain\(\)](#)
- [function Socket.end\(data\)](#)
- [event Socket.error\(details\)](#)
- [function Socket.pipe\(destination\\_options\)](#)
- [function Socket.read\(chars\)](#)
- [function Socket.write\(data\)](#)

---

### [function Socket.available](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function Socket.available()
```

#### Returns

How many bytes are available

#### Description

Return how many bytes are available to read. If there is already a listener for data, this will always return 0.

---

### [event Socket.close](#) ⇒

---

#### Call type:

[\(top\)](#)

```
Socket.on('close', function(had_error) { ... });
```

#### Parameters

had\_error - A boolean indicating whether the connection had an error (use an error event handler to get error details).

#### Description

Called when the connection closes.

---

### [event Socket.data](#) ⇒

---

#### Call type:

[\(top\)](#)

```
Socket.on('data', function(data) { ... });
```

#### Parameters

data - A string containing one or more characters of received data

#### Description

The 'data' event is called when data is received. If a handler is defined with `x.on('data', function(data) { ... })` then it will be called, otherwise data will be stored in an internal buffer, where it can be retrieved with `x.read()`

## [event Socket.drain](#) ⇒

---

**Call type:**

```
Socket.on('drain', function() { ... });
```

### Description

An event that is fired when the buffer is empty and it can accept more data to send.

[\(top\)](#)

## [function Socket.end](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Socket.end(data)
```

### Parameters

`data` - A string containing data to send

### Description

Close this socket - optional data to append as an argument.

See [socket.write](#) for more information about the data argument

## [event Socket.error](#) ⇒

---

**Call type:**

[\(top\)](#)

```
Socket.on('error', function(details) { ... });
```

### Parameters

`details` - An error object with an error code (a negative integer) and a message.

### Description

There was an error on this socket and it is closing (or wasn't opened in the first place). If a "connected" event was issued on this socket then the error event is always followed by a close event. The error codes are:

- -1: socket closed (this is not really an error and will not cause an error callback)
- -2: out of memory (typically while allocating a buffer to hold data)
- -3: timeout
- -4: no route
- -5: busy
- -6: not found (DNS resolution)
- -7: max sockets (... exceeded)
- -8: unsent data (some data could not be sent)
- -9: connection reset (or refused)
- -10: unknown error
- -11: no connection
- -12: bad argument
- -13: SSL handshake failed
- -14: invalid SSL data

## [function Socket.pipe](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function Socket.pipe(destination, options)
```

### Parameters

`destination` - The destination file/stream that will receive content from the source.

options - [optional] An object { `chunkSize : int=32`, `end : bool=true`, `complete : function` }  
chunkSize : The amount of data to pipe from source to destination at a time  
complete : a function to call when the pipe activity is complete  
end : call the 'end' function on the destination when the source is finished

## Description

Pipe this to a stream (an object with a 'write' method)

**Note:** This is not available in devices with low flash memory

---

## [function Socket.read](#) ⇒

### Call type:

[\(top\)](#)

```
function Socket.read(chars)
```

### Parameters

chars - The number of characters to read, or undefined/0 for all available

### Returns

A string containing the required bytes.

## Description

Return a string containing characters that have been received

---

## [function Socket.write](#) ⇒

### Call type:

[\(top\)](#)

```
function Socket.write(data)
```

### Parameters

data - A string containing data to send

### Returns

For node.js compatibility, returns the boolean false. When the send buffer is empty, a drain event will be sent

## Description

This function writes the data argument as a string. Data that is passed in (including arrays) will be converted to a string with the normal JavaScript `toString` method.

If you wish to send binary data then you need to convert that data directly to a String. This can be done with [`String.fromCharCode`](#), however it's often easier and faster to use the Espruino-specific [`E.toString`](#), which will read its arguments as an array of bytes and convert that to a String:

```
socket.write(E.toString([0,1,2,3,4,5]));
```

If you need to send something other than bytes, you can use 'Typed Arrays', or even [`DataView`](#):

```
var d = new DataView(new ArrayBuffer(8)); // 8 byte array buffer
d.setFloat32(0, 765.3532564); // write float at bytes 0-3
d.setInt8(4, 42); // write int8 at byte 4
socket.write(E.toString(d.buffer))
```

---

## [SPI Class](#)

---

This class allows use of the built-in SPI ports. Currently it is SPI master only.

[\(top\)](#)

### Instances

- [SPI1](#) The first SPI port
- [SPI2](#) The second SPI port
- [SPI3](#) The third SPI port

### Methods and Fields

- [SPI.find\(pin\)](#)
- [function SPI.send\(data, nss\\_pin\)](#)
- [function SPI.send4bit\(data, bit0, bit1, nss\\_pin\)](#)
- [function SPI.send8bit\(data, bit0, bit1, nss\\_pin\)](#)
- [function SPI.setup\(options\)](#)
- [constructor SPI\(\)](#)
- [function SPI.write\(data, ...\)](#)

---

### [SPI.find](#) ⇒

---

#### Call type:

[\(top\)](#)

```
SPI.find(pin)
```

#### Parameters

pin - A pin to search with

#### Returns

An object of type [SPI](#), or [undefined](#) if one couldn't be found.

#### Description

Try and find an SPI hardware device that will work on this pin (e.g. [SPI1](#))

May return undefined if no device can be found.

---

### [function SPI.send](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function SPI.send(data, nss_pin)
```

#### Parameters

data - The data to send - either an Integer, Array, String, or Object of the form {data: ..., count:#}

nss\_pin - An nSS pin - this will be lowered before SPI output and raised afterwards (optional). There will be a small delay between when this is lowered and when sending starts, and also between sending finishing and it being raised.

#### Returns

The data that was returned

#### Description

Send data down SPI, and return the result. Sending an integer will return an integer, a String will return a String, and anything else will return a Uint8Array.

Sending multiple bytes in one call to send is preferable as they can then be transmitted end to end. Using multiple calls to send() will result in significantly slower transmission speeds.

For maximum speeds, please pass either Strings or Typed Arrays as arguments. Note that you can even pass arrays of arrays, like [1, [2, 3, 4], 5]

## [function SPI.send4bit](#) ⇒

---

### Call type:

[\(top\)](#)

```
function SPI.send4bit(data, bit0, bit1, nss_pin)
```

### Parameters

`data` - The data to send - either an integer, array, or string

`bit0` - The 4 bits to send for a 0 (MSB first)

`bit1` - The 4 bits to send for a 1 (MSB first)

`nss_pin` - An nSS pin - this will be lowered before SPI output and raised afterwards (optional). There will be a small delay between when this is lowered and when sending starts, and also between sending finishing and it being raised.

### Description

Send data down SPI, using 4 bits for each 'real' bit (MSB first). This can be useful for faking one-wire style protocols

Sending multiple bytes in one call to send is preferable as they can then be transmitted end to end. Using multiple calls to send() will result in significantly slower transmission speeds.

## [function SPI.send8bit](#) ⇒

---

### Call type:

[\(top\)](#)

```
function SPI.send8bit(data, bit0, bit1, nss_pin)
```

### Parameters

`data` - The data to send - either an integer, array, or string

`bit0` - The 8 bits to send for a 0 (MSB first)

`bit1` - The 8 bits to send for a 1 (MSB first)

`nss_pin` - An nSS pin - this will be lowered before SPI output and raised afterwards (optional). There will be a small delay between when this is lowered and when sending starts, and also between sending finishing and it being raised

### Description

Send data down SPI, using 8 bits for each 'real' bit (MSB first). This can be useful for faking one-wire style protocols

Sending multiple bytes in one call to send is preferable as they can then be transmitted end to end. Using multiple calls to send() will result in significantly slower transmission speeds.

**Note:** This is not available in devices with low flash memory

## [function SPI.setup](#) ⇒

---

### Call type:

[\(top\)](#)

```
function SPI.setup(options)
```

### Parameters

`options` - An Object containing extra information on initialising the SPI port

### Description

Set up this SPI port as an SPI Master.

Options can contain the following (defaults are shown where relevant):

```
{
 sck:pin,
 miso:pin,
 mosi:pin,
 baud:integer=100000, // ignored on software SPI
 mode:integer=0, // between 0 and 3
 order:string='msb' // can be 'msb' or 'lsb'
 bits:8 // only available for software SPI
}
```

If `sck`, `miso` and `mosi` are left out, they will automatically be chosen. However if one or more is specified then the unspecified pins will not be set up.

You can find out which pins to use by looking at [your board's reference page](#) and searching for pins with the `SPI` marker. Some boards such as those based on nRF52 chips can have SPI on any pins, so don't have specific markings.

The SPI mode is between 0 and 3 - see [http://en.wikipedia.org/wiki/Serial\\_Peripheral\\_Interface\\_Bus#Clock\\_polarity\\_and\\_phase](http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#Clock_polarity_and_phase)

On STM32F1-based parts, you cannot mix AF and non-AF pins (SPI pins are usually grouped on the chip - and you can't mix pins from two groups). Espruino will not warn you about this.

## [constructor SPI](#) ⇒

---

**Call type:**

[\(top\)](#)

```
new SPI()
```

**Returns**

A SPI object

**Description**

Create a software SPI port. This has limited functionality (no baud rate), but it can work on any pins.

Use `SPI.setup` to configure this port.

## [function SPI.write](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function SPI.write(data, ...)
```

**Parameters**

`data, ...` - One or more items to write. May be ints, strings, arrays, or special objects (see [E.toInt8Array](#) for more info). If the last argument is a pin, it is taken to be the NSS pin

**Description**

Write a character or array of characters to SPI - without reading the result back.

For maximum speeds, please pass either Strings or Typed Arrays as arguments.

## [Storage Library](#)

---

This module allows you to read and write part of the nonvolatile flash memory of your device using a filesystem-like API.

(top)

Also see the [Flash](#) library, which provides a low level, more dangerous way to access all parts of your flash memory.

The [Storage](#) library provides two distinct types of file:

- `require("Storage").write(...)/require("Storage").read(...)`/etc create simple contiguous files of fixed length. This is the recommended file type.
- `require("Storage").open(...)` creates a [Storagefile](#), which stores the file in numbered chunks (`"filename\1"/"filename\2"/etc`). It allows data to be appended and for the file to be read line by line.

You must read a file using the same method you used to write it - e.g. you can't create a file with `require("Storage").open(...)` and then read it with `require("Storage").read(...)`.

**Note:** In firmware 2v05 and later, the maximum length for filenames is 28 characters. However in 2v04 and earlier the max length is 8.

### Methods and Fields

- `require("Storage").compact()`
- `require("Storage").debug()`
- `require("Storage").erase(name)`
- `require("Storage").eraseAll()`
- `require("Storage").getFree()`
- `require("Storage").getStats()`
- `require("Storage").hash(regex)`
- `require("Storage").list(regex, filter)`
- `require("Storage").open(name, mode)`
- `require("Storage").optimise()`
- `require("Storage").read(name, offset, length)`
- `require("Storage").readArrayBuffer(name)`
- `require("Storage").readJSON(name, noExceptions)`
- `require("Storage").write(name, data, offset, size)`
- `require("Storage").writeJSON(name, data)`

### [Storage.compact](#) ⇒

---

#### Call type:

(top)

```
require("Storage").compact()
```

#### Description

The Flash Storage system is journaling. To make the most of the limited write cycles of Flash memory, Espruino marks deleted/replaced files as garbage/trash files and moves on to a fresh part of flash memory. Espruino only fully erases those files when it is running low on flash, or when `compact` is called.

`compact` may fail if there isn't enough RAM free on the stack to use as swap space, however in this case it will not lose data.

**Note:** `compact` rearranges the contents of memory. If code is referencing that memory (e.g. functions that have their code stored in flash) then they may become garbled when compaction happens. To avoid this, call `eraseFiles` before uploading data that you intend to reference to ensure that uploaded files are right at the start of flash and cannot be compacted further.

**Note:** This is not available in devices with low flash memory

### [Storage.debug](#) ⇒

---

#### Call type:

(top)

```
require("Storage").debug()
```

#### Description

This writes information about all blocks in flash memory to the console - and is only useful for debugging flash storage.

**Note:** This is only available in debug builds

## [Storage.erase](#) ⇒

---

Call type:

[\(top\)](#)

```
require("Storage").erase(name)
```

### Parameters

name - The filename - max 28 characters (case sensitive)

### Description

Erase a single file from the flash storage area.

**Note:** This function should be used with normal files, and not [StorageFile](#)s created with `require("Storage").open(filename, ...)`

## [Storage.eraseAll](#) ⇒

---

Call type:

[\(top\)](#)

```
require("Storage").eraseAll()
```

### Description

Erase the flash storage area. This will remove all files created with `require("Storage").write(...)` as well as any code saved with [save\(\)](#) or [E.setBootCode\(\)](#).

## [Storage.getFree](#) ⇒

---

Call type:

[\(top\)](#)

```
require("Storage").getFree()
```

### Returns

The amount of free bytes

### Description

Return the amount of free bytes available in Storage. Due to fragmentation there may be more bytes available, but this represents the maximum size of file that can be written.

**Note:** This is not available in devices with low flash memory

## [Storage.getStats](#) ⇒

---

Call type:

[\(top\)](#)

```
require("Storage").getStats()
```

### Returns

An object containing info about the current Storage system

### Description

Returns:

```
{
 totalBytes // Amount of bytes in filesystem
 freeBytes // How many bytes are left at the end of storage?
 fileBytes // How many bytes of allocated files do we have?
 fileCount // How many allocated files do we have?
 trashBytes // How many bytes of trash files do we have?
 trashCount // How many trash files do we have? (can be cleared with .compact)
}
```

**Note:** This is not available in devices with low flash memory

## [Storage.hash](#) ⇒

---

Call type:

[\(top\)](#)

```
require("Storage").hash(regex)
```

### Parameters

regex - [optional] If supplied, filenames are checked against this regular expression (with `String.match(regexp)`) to see if they match before being hashed

### Returns

A hash of the files matching

### Description

List all files in the flash storage area matching the specified regex (ignores StorageFiles), and then hash their filenames *and* file locations.

Identical files may have different hashes (e.g. if Storage is compacted and the file moves) but the changes of different files having the same hash are extremely small.

```
// Hash files
require("Storage").hash()
// Files ending in '.boot.js'
require("Storage").hash(/\.\boot\.\js$/)
```

**Note:** This function is used by Bangle.js as a way to cache files. For instance the bootloader will add all `.boot.js` files together into a single `.boot0` file, but it needs to know quickly whether anything has changed.

**Note:** This is not available in devices with low flash memory

## [Storage.list](#) ⇒

---

Call type:

[\(top\)](#)

```
require("Storage").list(regex, filter)
```

### Parameters

regex - [optional] If supplied, filenames are checked against this regular expression (with `String.match(regexp)`) to see if they match before being returned

filter - [optional] If supplied, File Types are filtered based on this: `{sf:true}` or `{sf:false}` for whether to show StorageFile

### Returns

An array of filenames

### Description

List all files in the flash storage area. An array of Strings is returned.

By default this lists files created by [StorageFile](#) (`require("Storage").open`) which have a file number ("`\1`"/"`\2`"/etc) appended to them.

```
// All files
require("Storage").list()
// Files ending in '.js'
require("Storage").list(/\.\js$/)
// All Storage Files
require("Storage").list(undefined, {sf:true})
// All normal files (e.g. created with Storage.write)
require("Storage").list(undefined, {sf:false})
```

**Note:** This will output system files (e.g. saved code) as well as files that you may have written.

## [Storage.open](#) ⇒

---

Call type:

[\(top\)](#)

```
require("Storage").open(name, mode)
```

## Parameters

name - The filename - max **27** characters (case sensitive)

mode - The open mode - must be either '**r**' for read, '**w**' for write , or '**a**' for append

## Returns

An object containing {read,write,erase}

## Description

Open a file in the Storage area. This can be used for appending data (normal read/write operations only write the entire file).

Please see [StorageFile](#) for more information (and examples).

**Note:** These files write through immediately - they do not need closing.

**Note:** This is not available in devices with low flash memory

---

## [Storage.optimise](#) ⇒

### Call type:

[\(top\)](#)

```
require("Storage").optimise()
```

## Description

Writes a lookup table for files into Bangle.js's storage. This allows any file stored up to that point to be accessed quickly.

**Note:** This is not available in devices with low flash memory

---

## [Storage.read](#) ⇒

### Call type:

[\(top\)](#)

```
require("Storage").read(name, offset, length)
```

## Parameters

name - The filename - max 28 characters (case sensitive)

offset - [optional] The offset in bytes to start from

length - [optional] The length to read in bytes (if <=0, the entire file is read)

## Returns

A string of data, or **undefined** if the file is not found

## Description

Read a file from the flash storage area that has been written with `require("Storage").write(...)`.

This function returns a memory-mapped String that points to the actual memory area in read-only memory, so it won't use up RAM.

As such you can check if a file exists efficiently using `require("Storage").read(filename) !== undefined`.

If you evaluate this string with `eval`, any functions contained in the String will keep their code stored in flash memory.

**Note:** This function should be used with normal files, and not [StorageFile](#)s created with `require("Storage").open(filename, ...)`

---

## [Storage.readArrayBuffer](#) ⇒

### Call type:

[\(top\)](#)

```
require("Storage").readArrayBuffer(name)
```

## Parameters

name - The filename - max 28 characters (case sensitive)

## Returns

An ArrayBuffer containing data from the file, or undefined

## Description

Read a file from the flash storage area that has been written with `require("Storage").write(...)`, and return the raw binary data as an ArrayBuffer.

This can be used:

- In a `DataView` with `new DataView(require("Storage").readArrayBuffer("x"))`
- In a `Uint8Array/Float32Array/etc` with

```
new
Uint8Array(require("Storage").readArrayBuffer("x"))
```

**Note:** This function should be used with normal files, and not `StorageFile`s created with `require("Storage").open(filename, ...)`

**Note:** This is not available in devices with low flash memory

---

## [Storage.readJSON](#) ⇒

### Call type:

([top](#))

```
require("Storage").readJSON(name, noExceptions)
```

## Parameters

name - The filename - max 28 characters (case sensitive)

noExceptions - If true and the JSON is not valid, just return `undefined` - otherwise an `Exception` is thrown

## Returns

An object containing parsed JSON from the file, or undefined

## Description

Read a file from the flash storage area that has been written with `require("Storage").write(...)`, and parse JSON in it into a JavaScript object.

This is identical to `JSON.parse(require("Storage").read(...))`. It will throw an exception if the data in the file is not valid JSON.

**Note:** This function should be used with normal files, and not `StorageFile`s created with `require("Storage").open(filename, ...)`

**Note:** This is not available in devices with low flash memory

---

## [Storage.write](#) ⇒

### Call type:

([top](#))

```
require("Storage").write(name, data, offset, size)
```

## Parameters

name - The filename - max 28 characters (case sensitive)

data - The data to write

offset - [optional] The offset within the file to write

size - [optional] The size of the file (if a file is to be created that is bigger than the data)

## Returns

True on success, false on failure

## Description

Write/create a file in the flash storage area. This is nonvolatile and will not disappear when the device resets or power is lost.

Simply write `require("Storage").write("MyFile", "Some data")` to write a new file, and `require("Storage").read("MyFile")` to read it.

If you supply:

- A String, it will be written as-is
- An array, will be written as a byte array (but read back as a String)
- An object, it will automatically be converted to a JSON string before being written.

**Note:** If an array is supplied it will not be converted to JSON. To be explicit about the conversion you can use [Storage.writeJSON](#)

You may also create a file and then populate data later **as long as you don't try and overwrite data that already exists**. For instance:

```
var f = require("Storage");
f.write("a","Hello",0,14);
f.write("a"," ",5);
f.write("a","World!!!",6);
print(f.read("a")); // "Hello World!!!"
f.write("a","",0); // Writing to location 0 again will cause the file to be re-written
print(f.read("a")); // "
```

This can be useful if you've got more data to write than you have RAM available - for instance the Web IDE uses this method to write large files into onboard storage.

**Note:** This function should be used with normal files, and not [StorageFile](#)s created with `require("Storage").open(filename, ...)`

---

## [Storage.writeJSON](#) =>

### Call type:

[\(top\)](#)

```
require("Storage").writeJSON(name, data)
```

### Parameters

name - The filename - max 28 characters (case sensitive)

data - The JSON data to write

### Returns

True on success, false on failure

## Description

Write/create a file in the flash storage area. This is nonvolatile and will not disappear when the device resets or power is lost.

Simply write `require("Storage").writeJSON("MyFile", [1,2,3])` to write a new file, and `require("Storage").readJSON("MyFile")` to read it.

This is equivalent to: `require("Storage").write(name, JSON.stringify(data))`

**Note:** This function should be used with normal files, and not [StorageFile](#)s created with `require("Storage").open(filename, ...)`

**Note:** This is not available in devices with low flash memory

## StorageFile Class

---

These objects are created from `require("Storage").open` and allow Storage items to be read/written.

[\(top\)](#)

The `Storage` library writes into Flash memory (which can only be erased in chunks), and unlike a normal filesystem it allocates files in one long contiguous area to allow them to be accessed easily from Espruino.

This presents a challenge for `StorageFile` which allows you to append to a file, so instead `StorageFile` stores files in chunks. It uses the last character of the filename to denote the chunk number (e.g. `"foobar\1"`, `"foobar\2"`, etc).

This means that while `StorageFile` files exist in the same area as those from `Storage`, they should be read using `Storage.open` (and not `Storage.read`).

```
f = s.open("foobar", "w");
f.write("Hello");
f.write(" World\n");
f.write("Hello\n");
f.write("World 2\n");
// there's no need to call 'close'
// then
f = s.open("foobar", "r");
f.read(13) // "Hello World\nH"
f.read(13) // "ello\nWorld 2\n"
f.read(13) // "Hello World 3"
f.read(13) // "\n"
f.read(13) // undefined
// or
f = s.open("foobar", "r");
f.readLine() // "Hello World\n"
f.readLine() // "Hello\n"
f.readLine() // "World 2\n"
f.readLine() // "Hello World 3\n"
f.readLine() // undefined
// now get rid of file
f.erase();
```

**Note:** `StorageFile` uses the fact that all bits of erased flash memory are 1 to detect the end of a file. As such you should not write character code 255 (`\xFF`) to these files.

### Methods and Fields

- [`function StorageFile.erase\(\)`](#)
- [`function StorageFile.getLength\(\)`](#)
- [`function StorageFile.pipe\(destination, options\)`](#)
- [`function StorageFile.read\(len\)`](#)
- [`function StorageFile.readLine\(\)`](#)
- [`function StorageFile.write\(data\)`](#)

### [function StorageFile.erase](#) ⇒

---

Call type:

[\(top\)](#)

```
function StorageFile.erase()
```

#### Description

Erase this file

**Note:** This is not available in devices with low flash memory

### [function StorageFile.getLength](#) ⇒

---

Call type:

[\(top\)](#)

```
function StorageFile.getLength()
```

#### Returns

The current length in bytes of the file

#### Description

Return the length of the current file.

This requires Espruino to read the file from scratch, which is not a fast operation.

**Note:** This is not available in devices with low flash memory

## [function StorageFile.pipe](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function StorageFile.pipe(destination, options)
```

**Parameters**

destination - The destination file/stream that will receive content from the source.

options - [optional] An object { chunkSize : **int**=32, end : **bool**=**true**, complete : **function** }

chunkSize : The amount of data to pipe from source to destination at a time

complete : a function to call when the pipe activity is complete

end : call the 'end' function on the destination when the source is finished

**Description**

Pipe this file to a stream (an object with a 'write' method)

**Note:** This is not available in devices with low flash memory

## [function StorageFile.read](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function StorageFile.read(len)
```

**Parameters**

len - How many bytes to read

**Returns**

A String, or undefined

**Description**

Read 'len' bytes of data from the file, and return a String containing those bytes.

If the end of the file is reached, the String may be smaller than the amount of bytes requested, or if the file is already at the end, **undefined** is returned.

**Note:** This is not available in devices with low flash memory

## [function StorageFile.readLine](#) ⇒

---

**Call type:**

[\(top\)](#)

```
function StorageFile.readLine()
```

**Returns**

A line of data

**Description**

Read a line of data from the file (up to and including "\n")

**Note:** This is not available in devices with low flash memory

## [function StorageFile.write](#) ⇒

---

Call type:

[\(top\)](#)

```
function StorageFile.write(data)
```

### Parameters

data - The data to write. This should not include '\xFF' (character code 255)

### Description

Append the given data to a file. You should not attempt to append "\xFF" (character code 255).

**Note:** This is not available in devices with low flash memory

## [String Class](#)

---

This is the built-in class for Text Strings.

(top)

Text Strings in Espruino are not zero-terminated, so you can store zeros in them.

### Methods and Fields

- [function String.charAt\(pos\)](#)
- [function String.charCodeAt\(pos\)](#)
- [function String.concat\(args, ...\)](#)
- [function String.endsWith\(searchString, length\)](#)
- [String.fromCharCode\(code, ...\)](#)
- [function String.includes\(substring, fromIndex\)](#)
- [function String.indexOf\(substring, fromIndex\)](#)
- [function String.lastIndexOf\(substring, fromIndex\)](#)
- [property String.length](#)
- [function String.match\(substr\)](#)
- [function String.padEnd\(targetLength, padString\)](#)
- [function String.padStart\(targetLength, padString\)](#)
- [function String.repeat\(count\)](#)
- [function String.replace\(subStr, newSubStr\)](#)
- [function String.slice\(start, end\)](#)
- [function String.split\(separator\)](#)
- [function String.startsWith\(searchString, position\)](#)
- [constructor String\(str, ...\)](#)
- [function String.substr\(start, len\)](#)
- [function String.substring\(start, end\)](#)
- [function String.toLowerCase\(\)](#)
- [function String.toUpperCase\(\)](#)
- [function String.trim\(\)](#)

### [function String.charAt](#) ⇒

---

[View MDN documentation](#)

#### Call type:

```
function String.charAt(pos)
```

#### Parameters

pos - The character number in the string. Negative values return characters from end of string (-1 = last char)

#### Returns

The character in the string

#### Description

Return a single character at the given position in the String.

### [function String.charCodeAt](#) ⇒

---

[View MDN documentation](#)

#### Call type:

(top)

```
function String.charCodeAt(pos)
```

#### Parameters

pos - The character number in the string. Negative values return characters from end of string (-1 = last char)

## Returns

The integer value of a character in the string

## Description

Return the integer value of a single character at the given position in the String.

Note that this returns 0 not 'NaN' for out of bounds characters

---

## [function String.concat](#) ⇒

[View MDN documentation](#)

### Call type:

```
function String.concat(args, ...)
```

### Parameters

args, ... - Strings to append

## Returns

The result of appending all arguments to this string

## Description

Append all arguments to this [String](#) and return the result. Does not modify the original [String](#).

**Note:** This is not available in devices with low flash memory

---

## [function String.endsWith](#) ⇒

[View MDN documentation](#)

### Call type:

```
function String.endsWith(searchString, length)
```

### Parameters

searchString - The string to search for

length - The 'end' of the string - if left off the actual length of the string is used

## Returns

**true** if the given characters are found at the end of the string, otherwise, **false**.

---

## [String.fromCharCode](#) ⇒

[View MDN documentation](#)

### Call type:

```
String.fromCharCode(code, ...)
```

### Parameters

code, ... - One or more character codes to create a string from (range 0-255).

## Returns

The character

[\(top\)](#)

## Description

Return the character(s) represented by the given character code(s).

### [function String.includes](#) ⇒

---

[View MDN documentation](#)

#### Call type:

```
function String.includes(substring, fromIndex)
```

#### Parameters

`substring` - The string to search for

`fromIndex` - The start character index (or 0 if not defined)

#### Returns

`true` if the given characters are in the string, otherwise, `false`.

### [function String.indexOf](#) ⇒

---

[View MDN documentation](#)

#### Call type:

```
function String.indexOf(substring, fromIndex)
```

#### Parameters

`substring` - The string to search for

`fromIndex` - Index to search from

#### Returns

The index of the string, or -1 if not found

## Description

Return the index of substring in this string, or -1 if not found

### [function String.lastIndexOf](#) ⇒

---

[View MDN documentation](#)

#### Call type:

```
function String.lastIndexOf(substring, fromIndex)
```

#### Parameters

`substring` - The string to search for

`fromIndex` - Index to search from

#### Returns

The index of the string, or -1 if not found

## Description

Return the last index of substring in this string, or -1 if not found

[\(top\)](#)

## [property String.length](#) ⇒

---

[View MDN documentation](#)

**Call type:**

```
property String.length
```

**Returns**

The value of the string

**Description**

Find the length of the string

(top)

## [function String.match](#) ⇒

---

[View MDN documentation](#)

**Call type:**

```
function String.match(substr)
```

**Parameters**

`substr` - Substring or RegExp to match

**Returns**

A match array or `null` (see below):

**Description**

Matches an occurrence `substr` in the string.

Returns `null` if no match, or:

```
"abcdef".match("b") == [
 "b", // array index 0 - the matched string
 index: 1, // the start index of the match
 input: "b" // the input string
]
"abcdefabcdef".match(/bcd/) == [
 "bcd", index: 1,
 input: "abcdefabcdef"
]
```

'Global' RegExp matches just return an array of matches (with no indices):

```
"abcdefabcdef".match(/bcd/g) = [
 "bcd",
 "bcd"
]
```

## [function String.padEnd](#) ⇒

---

[View MDN documentation](#)

**Call type:**

```
function String.padEnd(targetLength, padString)
```

**Parameters**

`targetLength` - The length to pad this string to

`padString` - [optional] The string to pad with, default is `' '`

**Returns**

(top)

A string containing this string padded to the correct length

## Description

Pad this string at the end to the required number of characters

```
"Hello".padEnd(10) == "Hello "
"123".padEnd(10, "-") == "123----"
```

**Note:** This is not available in devices with low flash memory

---

## [function String.padStart](#) ⇒

[View MDN documentation](#)

### Call type:

```
function String.padStart(targetLength, padString)
```

### Parameters

`targetLength` - The length to pad this string to

`padString` - [optional] The string to pad with, default is `' '`

### Returns

A string containing this string padded to the correct length

## Description

Pad this string at the beginning to the required number of characters

```
"Hello".padStart(10) == " Hello"
"123".padStart(10, "-") == "----123"
```

**Note:** This is not available in devices with low flash memory

---

## [function String.repeat](#) ⇒

[View MDN documentation](#)

### Call type:

```
function String.repeat(count)
```

### Parameters

`count` - An integer with the amount of times to repeat this String

### Returns

A string containing repetitions of this string

## Description

Repeat this string the given number of times.

**Note:** This is not available in devices with low flash memory

---

## [function String.replace](#) ⇒

[View MDN documentation](#)

### Call type:

```
function String.replace(subStr, newSubStr)
```

[\(top\)](#)

## Parameters

`subStr` - The string to search for

`newSubStr` - The string to replace it with

## Returns

This string with `subStr` replaced

## Description

Search and replace ONE occurrence of `subStr` with `newSubStr` and return the result. This doesn't alter the original string. Regular expressions not supported.

---

## [function String.slice](#) ⇒

[View MDN documentation](#)

### Call type:

```
function String.slice(start, end)
```

## Parameters

`start` - The start character index, if negative it is from the end of the string

`end` - [optional] The end character index, if negative it is from the end of the string, and if omitted it is the end of the string

## Returns

Part of this string from start for len characters

---

## [function String.split](#) ⇒

[View MDN documentation](#)

### Call type:

```
function String.split(separator)
```

## Parameters

`separator` - The separator [String](#) or [RegExp](#) to use

## Returns

Part of this string from start for len characters

## Description

Return an array made by splitting this string up by the separator. e.g. `'1,2,3'.split(',')==['1', '2', '3']`

Regular Expressions can also be used to split strings, e.g.

```
'1a2b3
4'.split(/[^0-9]/)==['1', '2', '3', '4']
.
```

---

## [function String.startsWith](#) ⇒

[View MDN documentation](#)

### Call type:

```
function String.startsWith(searchString, position)
```

[\(top\)](#)

## Parameters

`searchString` - The string to search for  
`position` - The start character index (or 0 if not defined)

## Returns

`true` if the given characters are found at the beginning of the string, otherwise, `false`.

## [constructor String](#) ⇒

---

[View MDN documentation](#)

### Call type:

```
new String(str, ...)
```

## Parameters

`str, ...` - A value to turn into a string. If undefined or not supplied, an empty String is created.

## Returns

A String

## Description

Create a new String

## [function String.substr](#) ⇒

---

[View MDN documentation](#)

### Call type:

```
function String.substr(start, len)
```

## Parameters

`start` - The start character index  
`len` - The number of characters

## Returns

Part of this string from start for len characters

## [function String.substring](#) ⇒

---

[View MDN documentation](#)

### Call type:

```
function String.substring(start, end)
```

## Parameters

`start` - The start character index (inclusive)  
`end` - The end character index (exclusive)

## Returns

The part of this string between start and end

[\(top\)](#)

## [function String.toLowerCase](#) ⇒

---

[View MDN documentation](#)

**Call type:**

([top](#))

```
function String.toLowerCase()
```

**Parameters**

**Returns**

The lowercase version of this string

## [function String.toUpperCase](#) ⇒

---

[View MDN documentation](#)

**Call type:**

([top](#))

```
function String.toUpperCase()
```

**Parameters**

**Returns**

The uppercase version of this string

## [function String.trim](#) ⇒

---

[View MDN documentation](#)

**Call type:**

([top](#))

```
function String.trim()
```

**Returns**

A String with Whitespace removed from the beginning and end

**Description**

Return a new string with any whitespace (tabs, space, form feed, newline, carriage return, etc) removed from the beginning and end.

---

## [SyntaxError Class](#)

---

The base class for syntax errors

(top)

### Methods and Fields

- [constructor SyntaxError\(message\)](#)
- [function SyntaxError.toString\(\)](#)

---

### [constructor SyntaxError](#) ⇒

---

[View MDN documentation](#)

#### Call type:

(top)

```
new SyntaxError(message)
```

#### Parameters

message - [optional] An message string

#### Returns

A SyntaxError object

#### Description

Creates a SyntaxError object

---

### [function SyntaxError.toString](#) ⇒

---

#### Call type:

(top)

```
function SyntaxError.toString()
```

#### Returns

A String

---

## [TelnetServer Library](#)

---

This library implements a telnet console for the Espruino interpreter. It requires a network connection, e.g. Wifi, and **currently only functions on the ESP8266 and on Linux**. It uses port 23 on the ESP8266 and port 2323 on Linux.

[\(top\)](#)

**Note:** To enable on Linux, run `./espruino --telnet`

### Methods and Fields

- [`require\("TelnetServer"\).setOptions\(options\)`](#)

---

### [TelnetServer.setOptions](#) ⇒

---

Call type:

[\(top\)](#)

```
require("TelnetServer").setOptions(options)
```

#### Parameters

`options` - Options controlling the telnet console server { `mode : 'on|off'` }

---

## [tensorflow Library](#)

---

### Methods and Fields

[\(top\)](#)

- [require\("tensorflow"\).create\(arenaSize, model\)](#)

### [tensorflow.create](#) ⇒

---

#### Call type:

[\(top\)](#)

```
require("tensorflow").create(arenaSize, model)
```

#### Parameters

arenaSize - The TensorFlow Arena size

model - The model to use - this should be a flat array/string

#### Returns

A tensorflow instance

---

## [TFMicroInterpreter Class](#)

---

Class containing an instance of TFMicroInterpreter

[\(top\)](#)

### Methods and Fields

- [function TFMicroInterpreter.getInput\(\)](#)
- [function TFMicroInterpreter.getOutput\(\)](#)
- [function TFMicroInterpreter.invoke\(\)](#)

---

### [function TFMicroInterpreter.getInput](#) ⇒

---

Call type:

[\(top\)](#)

```
function TFMicroInterpreter.getInput()
```

### Returns

An arraybuffer referencing the input data

---

### [function TFMicroInterpreter.getOutput](#) ⇒

---

Call type:

[\(top\)](#)

```
function TFMicroInterpreter.getOutput()
```

### Returns

An arraybuffer referencing the output data

---

### [function TFMicroInterpreter.invoke](#) ⇒

---

Call type:

[\(top\)](#)

```
function TFMicroInterpreter.invoke()
```

## [tls Library](#)

---

This library allows you to create TCPIP servers and clients using TLS encryption

[\(top\)](#)

In order to use this, you will need an extra module to get network connectivity.

This is designed to be a cut-down version of the [node.js library](#). Please see the [Internet](#) page for more information on how to use it.

### Methods and Fields

- [require\("tls"\).connect\(options, callback\)](#)

#### [tls.connect](#) ⇒

---

##### Call type:

[\(top\)](#)

```
require("tls").connect(options, callback)
```

##### Parameters

options - An object containing host,port fields

callback - A function(res) that will be called when a connection is made. You can then call `res.on('data', function(data) { ... })` and `res.on('close', function() { ... })` to deal with the response.

##### Returns

Returns a new net.Socket object

##### Description

Create a socket connection using TLS

Options can have `ca`, `key` and `cert` fields, which should be the decoded content of the certificate.

```
var options = url.parse("localhost:1234");
options.key = atob("MIJKQ ... OZs08C");
options.cert = atob("MIIIfi ... Uf93rN+");
options.ca = atob("MIIFgDCC ... GosQML4sc=");
require("tls").connect(options, ...);
```

If you have the certificates as .pem files, you need to load these files, take the information between the lines beginning with ----, remove the newlines from it so you have raw base64, and then feed it into `atob` as above.

You can also:  
\* Just specify the filename (<=100 characters) and it will be loaded and parsed if you have an SD card connected. For instance `options.key = "key.pem"`;  
\* Specify a function, which will be called to retrieve the data. For instance `options.key = function() { eeprom.load\_my\_info(); };

For more information about generating and using certificates, see:

<https://engineering.circle.com/https-authorized-certs-with-node-js/>

(You'll need to use 2048 bit certificates as opposed to 4096 bit shown above)

**Note:** This is only available in devices with TLS and SSL support (Espruino Pico and Espruino WiFi only)

---

## [tv Library](#)

---

This library provides TV out capability on the Espruino and Espruino Pico.

[\(top\)](#)

See the [Television](#) page for more information.

### Methods and Fields

- `require("tv").setup(options, width)`

---

### [tv.setup](#) ⇒

---

#### Call type:

[\(top\)](#)

```
require("tv").setup(options, width)
```

#### Parameters

`options` - Various options for the TV output

`width` -

#### Returns

A graphics object

#### Description

This initialises the TV output. Options for PAL are as follows:

```
var g = require('tv').setup({ type : "pal",
 video : A7, // Pin - SPI MOSI Pin for Video output (MUST BE SPI1)
 sync : A6, // Pin - Timer pin to use for video sync
 width : 384,
 height : 270, // max 270
});
```

and for VGA:

```
var g = require('tv').setup({ type : "vga",
 video : A7, // Pin - SPI MOSI Pin for Video output (MUST BE SPI1)
 hsync : A6, // Pin - Timer pin to use for video sync
 vsync : A5, // Pin - pin to use for video sync
 width : 220,
 height : 240,
 repeat : 2, // amount of times to repeat each line
});
```

or

```
var g = require('tv').setup({ type : "vga",
 video : A7, // Pin - SPI MOSI Pin for Video output (MUST BE SPI1)
 hsync : A6, // Pin - Timer pin to use for video sync
 vsync : A5, // Pin - pin to use for video sync
 width : 220,
 height : 480,
 repeat : 1, // amount of times to repeat each line
});
```

See the [Television](#) page for more information.

---

## [TypeError Class](#)

---

The base class for type errors

[\(top\)](#)

### Methods and Fields

- [function TypeError.toString\(\)](#)
- [constructor TypeError\(message\)](#)

---

### [function TypeError.toString](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function TypeError.toString()
```

#### Returns

A String

---

### [constructor TypeError](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

```
new TypeError(message)
```

#### Parameters

message - [optional] An message string

#### Returns

A TypeError object

#### Description

Creates a TypeError object

---

## [Uint16Array Class](#)

---

This is the built-in JavaScript class for a typed array of 16 bit unsigned integers.

[\(top\)](#)

Instantiate this in order to efficiently store arrays of data (Espruino's normal arrays store data in a map, which is inefficient for non-sparse arrays).

Arrays of this type include all the methods from [ArrayBufferView](#)

### Methods and Fields

- [constructor Uint16Array\(arr, byteOffset, length\)](#)

---

#### [constructor Uint16Array](#) ⇒

---

[View MDN documentation](#)

##### Call type:

[\(top\)](#)

```
new Uint16Array(arr, byteOffset, length)
```

##### Parameters

`arr` - The array or typed array to base this off, or an integer which is the array length

`byteOffset` - The byte offset in the ArrayBuffer (ONLY IF the first argument was an ArrayBuffer)

`length` - The length (ONLY IF the first argument was an ArrayBuffer)

##### Returns

A typed array

##### Description

Create a typed array based on the given input. Either an existing ArrayBuffer, an Integer as a Length, or a simple array. If an [ArrayBufferView](#) (e.g. [Uint8Array](#), rather than [ArrayBuffer](#)) is given, it will be completely copied rather than referenced.

---

## [Uint24Array Class](#)

---

This is the built-in JavaScript class for a typed array of 24 bit unsigned integers.

[\(top\)](#)

Instantiate this in order to efficiently store arrays of data (Espruino's normal arrays store data in a map, which is inefficient for non-sparse arrays).

Arrays of this type include all the methods from [ArrayBufferView](#)

### Methods and Fields

- [constructor Uint24Array\(arr, byteOffset, length\)](#)

---

### [constructor Uint24Array](#) ⇒

---

#### Call type:

[\(top\)](#)

```
new Uint24Array(arr, byteOffset, length)
```

#### Parameters

`arr` - The array or typed array to base this off, or an integer which is the array length

`byteOffset` - The byte offset in the ArrayBuffer (ONLY IF the first argument was an ArrayBuffer)

`length` - The length (ONLY IF the first argument was an ArrayBuffer)

#### Returns

A typed array

#### Description

Create a typed array based on the given input. Either an existing ArrayBuffer, an Integer as a Length, or a simple array. If an [ArrayBufferView](#) (e.g. [Uint8Array](#) rather than [ArrayBuffer](#)) is given, it will be completely copied rather than referenced.

**Note:** This is not available in devices with low flash memory

## [Uint32Array Class](#)

---

This is the built-in JavaScript class for a typed array of 32 bit unsigned integers.

(top)

Instantiate this in order to efficiently store arrays of data (Espruino's normal arrays store data in a map, which is inefficient for non-sparse arrays).

Arrays of this type include all the methods from [ArrayBufferView](#)

### Methods and Fields

- [constructor Uint32Array\(arr, byteOffset, length\)](#)

#### [constructor Uint32Array](#) ⇒

---

[View MDN documentation](#)

##### Call type:

(top)

```
new Uint32Array(arr, byteOffset, length)
```

##### Parameters

`arr` - The array or typed array to base this off, or an integer which is the array length

`byteOffset` - The byte offset in the ArrayBuffer (ONLY IF the first argument was an ArrayBuffer)

`length` - The length (ONLY IF the first argument was an ArrayBuffer)

##### Returns

A typed array

##### Description

Create a typed array based on the given input. Either an existing ArrayBuffer, an Integer as a Length, or a simple array. If an [ArrayBufferView](#) (e.g. [Uint8Array](#), rather than [ArrayBuffer](#)) is given, it will be completely copied rather than referenced.

---

## [Uint8Array Class](#)

---

This is the built-in JavaScript class for a typed array of 8 bit unsigned integers.

[\(top\)](#)

Instantiate this in order to efficiently store arrays of data (Espruino's normal arrays store data in a map, which is inefficient for non-sparse arrays).

Arrays of this type include all the methods from [ArrayBufferView](#)

### Methods and Fields

- [constructor Uint8Array\(arr, byteOffset, length\)](#)

---

### [constructor Uint8Array](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

```
new Uint8Array(arr, byteOffset, length)
```

#### Parameters

`arr` - The array or typed array to base this off, or an integer which is the array length

`byteOffset` - The byte offset in the ArrayBuffer (ONLY IF the first argument was an ArrayBuffer)

`length` - The length (ONLY IF the first argument was an ArrayBuffer)

#### Returns

A typed array

#### Description

Create a typed array based on the given input. Either an existing ArrayBuffer, an Integer as a Length, or a simple array. If an [ArrayBufferView](#) (e.g. [Uint8Array](#), rather than [ArrayBuffer](#)) is given, it will be completely copied rather than referenced.

## [Uint8ClampedArray Class](#)

---

This is the built-in JavaScript class for a typed array of 8 bit unsigned integers that are automatically clamped to the range 0 to 255.

[\(top\)](#)

Instantiate this in order to efficiently store arrays of data (Espruino's normal arrays store data in a map, which is inefficient for non-sparse arrays).

Arrays of this type include all the methods from [ArrayBufferView](#)

### Methods and Fields

- [constructor Uint8ClampedArray\(arr, byteOffset, length\)](#)

### [constructor Uint8ClampedArray](#) ⇒

---

[View MDN documentation](#)

#### Call type:

[\(top\)](#)

```
new Uint8ClampedArray(arr, byteOffset, length)
```

#### Parameters

`arr` - The array or typed array to base this off, or an integer which is the array length

`byteOffset` - The byte offset in the ArrayBuffer (ONLY IF the first argument was an ArrayBuffer)

`length` - The length (ONLY IF the first argument was an ArrayBuffer)

#### Returns

A typed array

#### Description

Create a typed array based on the given input. Either an existing ArrayBuffer, an Integer as a Length, or a simple array. If an [ArrayBufferView](#) (e.g. [Uint8Array](#), rather than [ArrayBuffer](#)) is given, it will be completely copied rather than referenced.

Clamped arrays clamp their values to the allowed range, rather than 'wrapping'. e.g. after `a[0]=12345;`, `a[0]==255`.

## Unistroke Class

---

This class provides functionality to recognise gestures drawn on a touchscreen. It is only built into Bangle.js 2.

[\(top\)](#)

Usage:

```
var strokes = {
 stroke1 : Unistroke.new(new Uint8Array([x1, y1, x2, y2, x3, y3, ...])),
 stroke2 : Unistroke.new(new Uint8Array([x1, y1, x2, y2, x3, y3, ...])),
 stroke3 : Unistroke.new(new Uint8Array([x1, y1, x2, y2, x3, y3, ...]))
};
var r = Unistroke.recognise(strokes,new Uint8Array([x1, y1, x2, y2, x3, y3, ...]))
print(r); // stroke1/stroke2/stroke3
```

### Methods and Fields

- [Unistroke.new\(xy\)](#)
- [Unistroke.recognise\(strokes,xy\)](#)

#### Unistroke.new ⇒

---

Call type:

[\(top\)](#)

```
Unistroke.new(xy)
```

#### Parameters

xy - An array of interleaved XY coordinates

#### Returns

A string of data representing this unistroke

#### Description

Create a new Unistroke based on XY coordinates

**Note:** This is only available in Bangle.js 2 smartwatches

#### Unistroke.recognise ⇒

---

Call type:

[\(top\)](#)

```
Unistroke.recognise(strokes, xy)
```

#### Parameters

strokes - An object of named strokes : {arrow:..., circle:...}

xy - An array of interleaved XY coordinates

#### Returns

The key name of the matched stroke

#### Description

Recognise based on an object of named strokes, and a list of XY coordinates

**Note:** This is only available in Bangle.js 2 smartwatches

---

## [url Class](#)

---

This class helps to convert URLs into Objects of information ready for http.request/get

[\(top\)](#)

### Methods and Fields

- [url.parse\(urlStr, parseQuery\)](#)

---

## [url.parse](#) ⇒

---

### Call type:

[\(top\)](#)

`url.parse(urlStr, parseQuery)`

### Parameters

`urlStr` - A URL to be parsed

`parseQuery` - Whether to parse the query string into an object not (default = false)

### Returns

An object containing options for [http.request](#) or [http.get](#). Contains `method`, `host`, `path`, `pathname`, `search`, `port` and `query`

### Description

A utility function to split a URL into parts

This is useful in web servers for instance when handling a request.

For instance `url.parse("/a?b=c&d=e", true)` returns `{"method": "GET", "host": "", "path": "/a?b=c&d=e", "pathname": "/a", "search": "?b=c&d=e", "port": 80, "query": {"b": "c", "d": "e"}}`

---

## [Waveform Class](#)

---

This class handles waveforms. In Espruino, a Waveform is a set of data that you want to input or output.

[\(top\)](#)

### Methods and Fields

- [function Waveform.startInput\(output, freq, options\)](#)
- [function Waveform.startOutput\(output, freq, options\)](#)
- [function Waveform.stop\(\)](#)
- [constructor Waveform\(samples, options\)](#)

---

### [function Waveform.startInput](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function Waveform.startInput(output, freq, options)
```

#### Parameters

`output` - The pin to output on

`freq` - The frequency to output each sample at

`options` - Optional options struct `{time:float,repeat:bool}` where: `time` is the time that the waveform will start output at, e.g. `getTime() + 1` (otherwise it is immediate), `repeat` is a boolean specifying whether to repeat the given sample

#### Description

Will start inputting the waveform on the given pin that supports analog. If not repeating, it'll emit a `finish` event when it is done.

**Note:** This is not available in devices with low flash memory

---

### [function Waveform.startOutput](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function Waveform.startOutput(output, freq, options)
```

#### Parameters

`output` - The pin to output on

`freq` - The frequency to output each sample at

`options` - Optional options struct `{time:float,repeat:bool}` where: `time` is the time that the waveform will start output at, e.g. `getTime() + 1` (otherwise it is immediate), `repeat` is a boolean specifying whether to repeat the given sample

#### Description

Will start outputting the waveform on the given pin - the pin must have previously been initialised with `analogWrite`. If not repeating, it'll emit a `finish` event when it is done.

**Note:** This is not available in devices with low flash memory

---

### [function Waveform.stop](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function Waveform.stop()
```

#### Description

Stop a waveform that is currently outputting

**Note:** This is not available in devices with low flash memory

## [constructor Waveform](#) ⇒

---

### Call type:

[\(top\)](#)

```
new Waveform(samples, options)
```

### Parameters

`samples` - The number of samples

`options` - Optional options struct {`doubleBuffer:bool`, `bits : 8/16`} where: `doubleBuffer` is whether to allocate two buffers or not (default false), and `bits` is the amount of bits to use (default 8).

### Returns

An Waveform object

### Description

Create a waveform class. This allows high speed input and output of waveforms. It has an internal variable called `buffer` (as well as `buffer2` when double-buffered - see `options` below) which contains the data to input/output.

When double-buffered, a 'buffer' event will be emitted each time a buffer is finished with (the argument is that buffer). When the recording stops, a 'finish' event will be emitted (with the first argument as the buffer).

**Note:** This is not available in devices with low flash memory

## [Wifi Library](#)

---

The Wifi library is designed to control the Wifi interface. It supports functionality such as connecting to wifi networks, getting network information, starting an access point, etc.

[\(top\)](#)

It is available on these devices:

- [Espruino WiFi](#)
- [ESP8266](#)
- [ESP32](#)

Certain features may or may not be implemented on your device however we have documented what is available and what isn't.

If you're not using one of the devices above, a separate WiFi library is provided. For instance:

- An [ESP8266 connected to an Espruino board](#)
- An [CC3000 WiFi Module](#)

[Other ways of connecting to the net](#) such as GSM, Ethernet and LTE have their own libraries.

You can use the WiFi library as follows:

```
var wifi = require("Wifi");
wifi.connect("my-ssid", {password:"my-pwd"}, function(ap){ console.log("connected:", ap); });
```

On ESP32/ESP8266 if you want the connection to happen automatically at boot, add `wifi.save()`. On other platforms, place `wifi.connect` in a function called `onInit`.

### Methods and Fields

- [event require\("Wifi"\).associated\(details\)](#)
- [event require\("Wifi"\).auth\\_change\(details\)](#)
- [require\("Wifi"\).connect\(ssid, options, callback\)](#)
- [event require\("Wifi"\).connected\(details\)](#)
- [event require\("Wifi"\).dhcp\\_timeout\(\)](#)
- [require\("Wifi"\).disconnect\(callback\)](#)
- [event require\("Wifi"\).disconnected\(details\)](#)
- [require\("Wifi"\).getAPDetails\(callback\)](#)
- [require\("Wifi"\).getAPIP\(callback\)](#)
- [require\("Wifi"\).getDetails\(callback\)](#)
- [require\("Wifi"\).getHostName\(hostname, callback\)](#)
- [require\("Wifi"\).getHostname\(callback\)](#)
- [require\("Wifi"\).getP\(callback\)](#)
- [require\("Wifi"\).getStatus\(callback\)](#)
- [require\("Wifi"\).ping\(hostname, callback\)](#)
- [event require\("Wifi"\).probe\\_recv\(details\)](#)
- [require\("Wifi"\).restore\(\)](#)
- [require\("Wifi"\).save\(what\)](#)
- [require\("Wifi"\).scan\(callback\)](#)
- [require\("Wifi"\).setAPIP\(settings, callback\)](#)
- [require\("Wifi"\).setConfig\(settings\)](#)
- [require\("Wifi"\).setHostName\(hostname, callback\)](#)
- [require\("Wifi"\).setIP\(settings, callback\)](#)
- [require\("Wifi"\).setSNTP\(server, tz\\_offset\)](#)
- [event require\("Wifi"\).sta\\_joined\(details\)](#)
- [event require\("Wifi"\).sta\\_left\(details\)](#)
- [require\("Wifi"\).startAP\(ssid, options, callback\)](#)
- [require\("Wifi"\).stopAP\(callback\)](#)
- [require\("Wifi"\).turbo\(enable, callback\)](#)

---

### [event Wifi.associated](#) ⇒

Call type:

[\(top\)](#)

```
wifi.on('associated', function(details) { ... });
```

#### Parameters

`details` - An object with event details

#### Description

The 'associated' event is called when an association with an access point has succeeded, i.e., a connection to the AP's network has been established.

On ESP32/ESP8266 there is a `details` parameter which includes:

- `ssid` - The SSID of the access point to which the association was established
- `mac` - The BSSID/mac address of the access point
- `channel` - The wifi channel used (an integer, typ 1..14)

## [event Wifi.auth\\_change](#) ⇒

---

**Call type:**

[\(top\)](#)

```
Wifi.on('auth_change', function(details) { ... });
```

**Parameters**

`details` - An object with event details

**Description**

The 'auth\_change' event is called when the authentication mode with the associated access point changes. The details include:

- `oldMode` - The old auth mode (string: open, wep, wpa, wpa2, wpa\_wpa2)
- `newMode` - The new auth mode (string: open, wep, wpa, wpa2, wpa\_wpa2)

**Note:** This is only available in ESP32 boards and ESP8266 boards running Espruino

## [Wifi.connect](#) ⇒

---

**Call type:**

[\(top\)](#)

```
require("Wifi").connect(ssid, options, callback)
```

**Parameters**

`ssid` - The access point network id.

`options` - [optional] Connection options.

`callback` - A `callback(err)` function to be called back on completion. `err` is null on success, or contains an error string on failure.

**Description**

Connect to an access point as a station. If there is an existing connection to an AP it is first disconnected if the SSID or password are different from those passed as parameters. Put differently, if the passed SSID and password are identical to the currently connected AP then nothing is changed. When the connection attempt completes the callback function is invoked with one `err` parameter, which is NULL if there is no error and a string message if there is an error. If DHCP is enabled the callback occurs once an IP address has been obtained, if a static IP is set the callback occurs once the AP's network has been joined. The callback is also invoked if a connection already exists and does not need to be changed.

The options properties may contain:

- `password` - Password string to be used to access the network.
- `dnsServers` (array of String) - An array of up to two DNS servers in dotted decimal format string.
- `channel` - Wifi channel of the access point (integer, typ 0..14, 0 means any channel), only on ESP8266.
- `bssid` - Mac address of the access point (string, type "00:00:00:00:00:00"), only on ESP8266.

Notes:

- the options should include the ability to set a static IP and associated netmask and gateway, this is a future enhancement.
- the only error reported in the callback is "Bad password", all other errors (such as access point not found or DHCP timeout) just cause connection retries. If the reporting of such temporary errors is desired, the caller must use its own timeout and the `getDetails().status` field.
- the `connect` call automatically enables station mode, it can be disabled again by calling `disconnect`.

## [event Wifi.connected](#) ⇒

---

**Call type:**

[\(top\)](#)

```
Wifi.on('connected', function(details) { ... });
```

## Parameters

`details` - An object with event details

## Description

The 'connected' event is called when the connection with an access point is ready for traffic. In the case of a dynamic IP address configuration this is when an IP address is obtained, in the case of static IP address allocation this happens when an association is formed (in that case the 'associated' and 'connected' events are fired in rapid succession).

On ESP32/ESP8266 there is a `details` parameter which includes:

- `ip` - The IP address obtained as string
- `netmask` - The network's IP range mask as string
- `gw` - The network's default gateway as string

---

## [event Wifi.dhcp\\_timeout](#) ⇒

### Call type:

[\(top\)](#)

```
Wifi.on('dhcp_timeout', function() { ... });
```

## Description

The 'dhcp\_timeout' event is called when a DHCP request to the connected access point fails and thus no IP address could be acquired (or renewed).

**Note:** This is only available in ESP32 boards and ESP8266 boards running Espruino

---

## [Wifi.disconnect](#) ⇒

### Call type:

[\(top\)](#)

```
require("Wifi").disconnect(callback)
```

## Parameters

`callback` - [optional] An `callback()` function to be called back on disconnection. The callback function receives no argument.

## Description

Disconnect the wifi station from an access point and disable the station mode. It is OK to call `disconnect` to turn off station mode even if no connection exists (for example, connection attempts may be failing). Station mode can be re-enabled by calling `connect` or `scan`.

---

## [event Wifi.disconnected](#) ⇒

### Call type:

[\(top\)](#)

```
Wifi.on('disconnected', function(details) { ... });
```

## Parameters

`details` - An object with event details

## Description

The 'disconnected' event is called when an association with an access point has been lost.

On ESP32/ESP8266 there is a `details` parameter which includes:

- `ssid` - The SSID of the access point from which the association was lost
- `mac` - The BSSID/mac address of the access point
- `reason` - The reason for the disconnection (string)

---

## [Wifi.getAPDetails](#) ⇒

**Call type:**[\(top\)](#)

```
require("Wifi").getAPDetails(callback)
```

**Parameters**

callback - [optional] A `callback(details)` function to be called back with the current access point details, i.e. the same object as returned directly.

**Returns**

An object representing the current access point details, if available immediately.

**Description**

Retrieve the current access point configuration and status. The details object has the following properties:

- `status` - Current access point status: `enabled` or `disabled`
- `stations` - an array of the stations connected to the access point. This array may be empty. Each entry in the array is an object describing the station which, at a minimum contains `ip` being the IP address of the station.
- `ssid` - SSID to broadcast.
- `password` - Password for authentication.
- `authMode` - the authentication required of stations: `open`, `wpa`, `wpa2`, `wpa_wpa2`.
- `hidden` - True if the SSID is hidden, false otherwise.
- `maxConn` - Max number of station connections supported.
- `savedSsid` - the SSID to broadcast automatically at boot time, null if the access point is to be disabled at boot.

**Note:** This is only available in ESP32 boards and ESP8266 boards running Espruino

---

**Wifi.getAPIP** ⇒**Call type:**[\(top\)](#)

```
require("Wifi").getAPIP(callback)
```

**Parameters**

callback - [optional] A `callback(err, ipinfo)` function to be called back with the the IP information.

**Returns**

An object representing the esp8266's Access Point IP information, if available immediately (**ONLY** on ESP8266/ESP32).

**Description**

Return the access point IP information in an object which contains:

- `ip` - IP address as string (typ "192.168.4.1")
- `netmask` - The interface netmask as string
- `gw` - The network gateway as string
- `mac` - The MAC address as string of the form 00:00:00:00:00:00

---

**Wifi.getDetails** ⇒**Call type:**[\(top\)](#)

```
require("Wifi").getDetails(callback)
```

**Parameters**

callback - [optional] An `callback(details)` function to be called back with the wifi details, i.e. the same object as returned directly.

**Returns**

An object representing the wifi station details, if available immediately.

**Description**

Retrieve the wifi station configuration and status details. The details object has the following properties:

- **status** - Details about the wifi station connection, one of `off`, `connecting`, `wrong_password`, `no_ap_found`, `connect_fail`, or `connected`. The `off`, `bad_password` and `connected` states are stable, the other states are transient. The connecting state will either result in `connected` or one of the error states (`bad_password`, `no_ap_found`, `connect_fail`) and the `no_ap_found` and `connect_fail` states will result in a reconnection attempt after some interval.
- **rssi** - signal strength of the connected access point in dB, typically in the range -110 to 0, with anything greater than -30 being an excessively strong signal.
- **ssid** - SSID of the access point.
- **password** - the password used to connect to the access point.
- **authMode** - the authentication used: `open`, `wpa`, `wpa2`, `wpa_wpa2` (not currently supported).
- **savedSsid** - the SSID to connect to automatically at boot time, null if none.

**Note:** This is only available in ESP32 boards and ESP8266 boards running Espruino

## [Wifi.getHostName](#) ⇒

---

Call type:

[\(top\)](#)

```
require("Wifi").getHostName(hostname, callback)
```

Parameters

`hostname` - The hostname to lookup.

`callback` - The `callback(ip)` to invoke when the IP is returned. `ip==null` on failure.

Description

Lookup the hostname and invoke a callback with the IP address as integer argument. If the lookup fails, the callback is invoked with a null argument. **Note:** only a single hostname lookup can be made at a time, concurrent lookups are not supported.

**Note:** This is only available in ESP8266 boards running Espruino and ESP32 boards

## [Wifi.getHostname](#) ⇒

---

Call type:

[\(top\)](#)

```
require("Wifi").getHostname(callback)
```

Parameters

`callback` - [optional] A `callback(hostname)` function to be called back with the hostname.

Returns

The currently configured hostname, if available immediately.

Description

Returns the hostname announced to the DHCP server and broadcast via mDNS when connecting to an access point.

**Note:** This is only available in ESP8266 boards running Espruino and ESP32 boards

## [Wifi.getIP](#) ⇒

---

Call type:

[\(top\)](#)

```
require("Wifi").getIP(callback)
```

Parameters

`callback` - [optional] A `callback(err, ipinfo)` function to be called back with the IP information.

Returns

An object representing the station IP information, if available immediately (**ONLY** on ESP8266/ESP32).

## Description

Return the station IP information in an object as follows:

- ip - IP address as string (e.g. "192.168.1.5")
- netmask - The interface netmask as string (ESP8266/ESP32 only)
- gw - The network gateway as string (ESP8266/ESP32 only)
- mac - The MAC address as string of the form 00:00:00:00:00:00

Note that the ip, netmask, and gw fields are omitted if no connection is established:

## [Wifi.getStatus](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("Wifi").getStatus(callback)
```

### Parameters

callback - Optional `callback(status)` function to be called back with the current Wifi status, i.e. the same object as returned directly.

### Returns

An object representing the current WiFi status, if available immediately.

## Description

Retrieve the current overall WiFi configuration. This call provides general information that pertains to both station and access point modes. The `getDetails` and `getAPDetails` calls provide more in-depth information about the station and access point configurations, respectively. The status object has the following properties:

- station - Status of the wifi station: off, connecting, ...
- ap - Status of the wifi access point: disabled, enabled.
- mode - The current operation mode: off, sta, ap, sta+ap.
- phy - Modulation standard configured: 11b, 11g, 11n (the esp8266 docs are not very clear, but it is assumed that 11n means b/g/n). This setting limits the modulations that the radio will use, it does not indicate the current modulation used with a specific access point.
- powersave - Power saving mode: none (radio is on all the time), ps-pol1 (radio is off between beacons as determined by the access point's DTIM setting). Note that in 'ap' and 'sta+ap' modes the radio is always on, i.e., no power saving is possible.
- savedMode - The saved operation mode which will be applied at boot time: off, sta, ap, sta+ap.

**Note:** This is only available in ESP32 boards and ESP8266 boards running Espruino

## [Wifi.ping](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("Wifi").ping(hostname, callback)
```

### Parameters

hostname - The host to ping

callback - A `callback(time)` function to invoke when a ping is received

## Description

Issues a ping to the given host, and calls a callback with the time when the ping is received.

**Note:** This is only available in Espruino WiFi boards and ESP8266 boards running Espruino and ESP32 boards

## [event Wifi.probe\\_recv](#) ⇒

---

### Call type:

[\(top\)](#)

```
Wifi.on('probe_recv', function(details) { ... });
```

### Parameters

`details` - An object with event details

## Description

The 'probe\_recv' event is called when a probe request is received from some station by the esp8266's access point. The details include:

- `mac` - The MAC address of the station in string format (00:00:00:00:00:00)
- `rssi` - The signal strength in dB of the probe request

**Note:** This is only available in ESP32 boards and ESP8266 boards running Espruino

---

## [Wifi.restore](#) ⇒

### Call type:

[\(top\)](#)

```
require("Wifi").restore()
```

## Description

Restores the saved Wifi configuration from flash. See [Wifi.save\(\)](#).

**Note:** This is only available in ESP32 boards and ESP8266 boards running Espruino

---

## [Wifi.save](#) ⇒

### Call type:

[\(top\)](#)

```
require("Wifi").save(what)
```

## Parameters

`what` - An optional parameter to specify what to save, on the esp8266 the two supported values are `clear` and `sta+ap`. The default is `sta+ap`

## Description

On boards where this is not available, just issue the `connect` commands you need to run at startup from an `onInit` function.

Save the current wifi configuration (station and access point) to flash and automatically apply this configuration at boot time, unless `what=="clear"`, in which case the saved configuration is cleared such that wifi remains disabled at boot. The saved configuration includes:

- mode (off/sta/ap/sta+ap)
- SSIDs & passwords
- phy (11b/g/n)
- powersave setting
- DHCP hostname

**Note:** This is only available in ESP32 boards and ESP8266 boards running Espruino

---

## [Wifi.scan](#) ⇒

### Call type:

[\(top\)](#)

```
require("Wifi").scan(callback)
```

## Parameters

`callback` - A `callback(err, ap_list)` function to be called back on completion. `err==null` and `ap_list` is an array on success, or `err` is an error string and `ap_list` is undefined on failure.

## Description

Perform a scan for access points. This will enable the station mode if it is not currently enabled. Once the scan is complete the callback function is called with an array of APs found, each AP is an object with:

- `ssid`: SSID string.
- `mac`: access point MAC address in 00:00:00:00:00:00 format.
- `authMode`: open, wep, wpa, wpa2, or wpa\_wpa2.
- `channel`: wifi channel 1..13.

- **hidden**: true if the SSID is hidden (ESP32/ESP8266 only)
- **rssi**: signal strength in dB in the range -110..0.

Notes: \* in order to perform the scan the station mode is turned on and remains on, use `Wifi.disconnect()` to turn it off again, if desired. \* only one scan can be in progress at a time.

## [Wifi.setAPIP](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("Wifi").setAPIP(settings, callback)
```

### Parameters

`settings` - Configuration settings

`callback` - A `callback(err)` function to invoke when ip is set. `err==null` on success, or a string on failure.

### Description

The `settings` object must contain the following properties.

- `ip` IP address as string (e.g. "192.168.5.100")
- `gw` The network gateway as string (e.g. "192.168.5.1")
- `netmask` The interface netmask as string (e.g. "255.255.255.0")

**Note:** This is only available in Espruino WiFi boards and ESP8266 boards running Espruino

## [Wifi.setConfig](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("Wifi").setConfig(settings)
```

### Parameters

`settings` - An object with the configuration settings to change.

### Description

Sets a number of global wifi configuration settings. All parameters are optional and which are passed determines which settings are updated. The settings available are:

- `phy` - Modulation standard to allow: `11b`, `11g`, `11n` (the esp8266 docs are not very clear, but it is assumed that `11n` means b/g/n).
- `powersave` - Power saving mode: `none` (radio is on all the time), `ps-pol1` (radio is off between beacons as determined by the access point's DTIM setting). Note that in 'ap' and 'sta+ap' modes the radio is always on, i.e., no power saving is possible.

Note: esp8266 SDK programmers may be missing an "opmode" option to set the sta/ap/sta+ap operation mode. Please use `connect`/`scan`/`disconnect`/`startAP`/`stopAP`, which all set the esp8266 opmode indirectly.

**Note:** This is only available in ESP32 boards and ESP8266 boards running Espruino

## [Wifi.setHostname](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("Wifi").setHostname(hostname, callback)
```

### Parameters

`hostname` - The new hostname.

`callback` - [optional] A `callback()` function to be called back when the hostname is set

### Description

Set the hostname. Depending on implementation, the hostname is sent with every DHCP request and is broadcast via mDNS. The DHCP hostname may be visible in the access point and may be forwarded into DNS as `hostname.local`. If a DHCP lease currently exists changing the hostname will cause a disconnect and

reconnect in order to transmit the change to the DHCP server. The mDNS announcement also includes an announcement for the "espruino" service.

**Note:** This is only available in ESP8266 boards running Espruino and Espruino WiFi boards and ESP32 boards

## [Wifi.setIP](#) ⇒

---

### Call type:

```
require("Wifi").setIP(settings, callback)
```

### Parameters

`settings` - Configuration settings

`callback` - A `callback(err)` function to invoke when ip is set. `err==null` on success, or a string on failure.

### Description

The `settings` object must contain the following properties.

- `ip` IP address as string (e.g. "192.168.5.100")
- `gw` The network gateway as string (e.g. "192.168.5.1")
- `netmask` The interface netmask as string (e.g. "255.255.255.0")

**Note:** This is only available in ESP8266 boards running Espruino and Espruino WiFi boards

[\(top\)](#)

## [Wifi.setSNTP](#) ⇒

---

### Call type:

[\(top\)](#)

```
require("Wifi").setSNTP(server, tz_offset)
```

### Parameters

`server` - The NTP server to query, for example, us.pool.ntp.org

`tz_offset` - Local time zone offset in the range -11..13.

### Description

Starts the SNTP (Simple Network Time Protocol) service to keep the clock synchronized with the specified server. Note that the time zone is really just an offset to UTC and doesn't handle daylight savings time. The interval determines how often the time server is queried and Espruino's time is synchronized. The initial synchronization occurs asynchronously after `setSNTP` returns.

**Note:** This is only available in ESP8266 boards running Espruino and ESP32 boards

## [event Wifi.sta\\_joined](#) ⇒

---

### Call type:

[\(top\)](#)

```
wifi.on('sta_joined', function(details) { ... });
```

### Parameters

`details` - An object with event details

### Description

The 'sta\_joined' event is called when a station establishes an association (i.e. connects) with the esp8266's access point. The details include:

- `mac` - The MAC address of the station in string format (00:00:00:00:00:00)

**Note:** This is only available in ESP32 boards and ESP8266 boards running Espruino

## [event Wifi.sta\\_left](#) ⇒

---

**Call type:**[\(top\)](#)

```
wifi.on('sta_left', function(details) { ... });
```

**Parameters**

details - An object with event details

**Description**

The 'sta\_left' event is called when a station disconnects from the esp8266's access point (or its association times out?). The details include:

- mac - The MAC address of the station in string format (00:00:00:00:00:00)

**Note:** This is only available in ESP32 boards and ESP8266 boards running Espruino

---

[Wifi.startAP](#) ⇒**Call type:**[\(top\)](#)

```
require("Wifi").startAP(ssid, options, callback)
```

**Parameters**

ssid - The network id.

options - [optional] Configuration options.

callback - Optional `callback(err)` function to be called when the AP is successfully started. `err==null` on success, or an error string on failure.

**Description**

Create a WiFi access point allowing stations to connect. If the password is NULL or an empty string the access point is open, otherwise it is encrypted. The callback function is invoked once the access point is set-up and receives one `err` argument, which is NULL on success and contains an error message string otherwise.

The `options` object can contain the following properties.

- authMode - The authentication mode to use. Can be one of "open", "wpa2", "wpa", "wpa\_wpa2". The default is open (but open access points are not recommended).
- password - The password for connecting stations if authMode is not open.
- channel - The channel to be used for the access point in the range 1..13. If the device is also connected to an access point as a station then that access point determines the channel.
- hidden - The flag if visible or not (0:visible, 1:hidden), default is visible.

Notes:

- the options should include the ability to set the AP IP and associated netmask, this is a future enhancement.
- the `startAP` call automatically enables AP mode. It can be disabled again by calling `stopAP`.

---

[Wifi.stopAP](#) ⇒**Call type:**[\(top\)](#)

```
require("Wifi").stopAP(callback)
```

**Parameters**

callback - [optional] An `callback()` function to be called back on successful stop. The callback function receives no argument.

**Description**

Stop being an access point and disable the AP operation mode. AP mode can be re-enabled by calling `startAP`.

---

[Wifi.turbo](#) ⇒**Call type:**[\(top\)](#)

```
require("Wifi").turbo(enable, callback)
```

## Parameters

`enable` - true (or a baud rate as a number) to enable, false to disable

`callback` - A `callback()` function to invoke when turbo mode has been set

## Description

Switch to using a higher communication speed with the WiFi module.

- `true` = 921600 baud
- `false` = 115200
- `1843200` (or any number) = use a specific baud rate. \* e.g. `wifi.turbo(true,callback)` or `wifi.turbo(1843200,callback)`

**Note:** This is only available in Espruino WiFi boards

---

## [WioLTE Class](#)

---

Class containing utility functions for the Seeed WIO LTE board

[\(top\)](#)

### Methods and Fields

- [WioLTE.A4](#)
- [WioLTE.A6](#)
- [WioLTE.D20](#)
- [WioLTE.D38](#)
- [WioLTE.I2C](#)
- [WioLTE.LED\(red,green,blue\)](#)
- [WioLTE.setGrovePower\(onoff\)](#)
- [WioLTE.setLEDPower\(onoff\)](#)
- [WioLTE.UART](#)

---

### [WioLTE.A4](#) ⇒

---

#### Call type:

[\(top\)](#)

`WioLTE.A4`

#### Returns

See description above

---

### [WioLTE.A6](#) ⇒

---

#### Call type:

[\(top\)](#)

`WioLTE.A6`

#### Returns

See description above

---

### [WioLTE.D20](#) ⇒

---

#### Call type:

[\(top\)](#)

`WioLTE.D20`

#### Returns

See description above

---

### [WioLTE.D38](#) ⇒

---

#### Call type:

[\(top\)](#)

`WioLTE.D38`

#### Returns

See description above

---

### [WioLTE.I2C](#) ⇒

---

**Call type:**[\(top\)](#)`WiOLTE.I2C`**Returns**

See description above

---

[WiOLTE.LED](#) ⇒**Call type:**[\(top\)](#)`WiOLTE.LED(red, green, blue)`**Parameters**

red - 0-255, red LED intensity

green - 0-255, green LED intensity

blue - 0-255, blue LED intensity

**Description**

Set the WIO's LED

---

[WiOLTE.setGrovePower](#) ⇒**Call type:**[\(top\)](#)`WiOLTE.setGrovePower(onoff)`**Parameters**

onoff - Whether to turn the Grove connectors power on or off (D38/D39 are always powered)

**Description**

Set the power of Grove connectors, except for D38 and D39 which are always on.

---

[WiOLTE.setLEDPower](#) ⇒**Call type:**[\(top\)](#)`WiOLTE.setLEDPower(onoff)`**Parameters**

onoff - true = on, false = off

**Description**

Turn power to the WIO's LED on or off.

Turning the LED on won't immediately display a color - that must be done with `WiOLTE.LED(r,g,b)`

---

[WiOLTE.UART](#) ⇒**Call type:**[\(top\)](#)`WiOLTE.UART`

## Returns

See description above

---

## WIZnet Library

---

Library for communication with the WIZnet Ethernet module

[\(top\)](#)

### Methods and Fields

- [`require\("WIZnet"\).connect\(spi, cs\)`](#)

---

### WIZnet.connect ⇒

---

#### Call type:

[\(top\)](#)

```
require("WIZnet").connect(spi, cs)
```

#### Parameters

`spi` - Device to use for SPI (or undefined to use the default)

`cs` - The pin to use for Chip Select

#### Returns

An Ethernet Object

#### Description

Initialise the WIZnet module and return an Ethernet object

**Note:** This is only available in builds with support for WIZnet Ethernet modules built in

---

## [WLAN Class](#)

---

An instantiation of a WiFi network adaptor

[\(top\)](#)

### Methods and Fields

- [function WLAN.connect\(ap, key, callback\)](#)
- [function WLAN.disconnect\(\)](#)
- [function WLAN.getIP\(\)](#)
- [function WLAN.reconnect\(\)](#)
- [function WLAN.setIP\(options\)](#)

---

### [function WLAN.connect](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function WLAN.connect(ap, key, callback)
```

#### Parameters

ap - Access point name

key - WPA2 key (or undefined for unsecured connection)

callback - Function to call back with connection status. It has one argument which is one of 'connect'/'disconnect'/'dhcp'

#### Returns

True if connection succeeded, false if it didn't.

#### Description

Connect to a wireless network

---

### [function WLAN.disconnect](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function WLAN.disconnect()
```

#### Description

Completely uninitialized and power down the CC3000. After this you'll have to use `require("CC3000").connect()` again.

---

### [function WLAN.getIP](#) ⇒

---

#### Call type:

[\(top\)](#)

```
function WLAN.getIP()
```

#### Returns

See description above

#### Description

Get the current IP address

## [function WLAN.reconnect](#) ⇒

---

**Call type:**

```
function WLAN.reconnect()
```

### Description

Completely uninitialized and power down the CC3000, then reconnect to the old access point.

## [function WLAN.setIP](#) ⇒

---

**Call type:**

```
function WLAN.setIP(options)
```

### Parameters

`options` - Object containing IP address options { `ip` : '1,2,3,4', `subnet`, `gateway`, `dns` }, or do not supply an object in order to force DHCP.

### Returns

True on success

### Description

Set the current IP address for get an IP from DHCP (if no options object is specified).

**Note:** Changes are written to non-volatile memory, but will only take effect after calling `wlan.reconnect()`

[\(top\)](#)