Connected Little Boxes

Reference Manual

Rob Miles

Manual Version 1.24

Code version: 1.1.0.0

Contents

Connected Little Boxes Getting Started Web site

You can get started with a Connected Little Box by visiting the Connected Little Boxes web site. You can use your browser to download the software into your device and then set up the WiFi and MQTT connections.

You can find the website here:

```
https://www.connectedlittleboxes.com/gettingstarted.html
```

Box Overview

A Connected Little Box is an Internet of Things (IoT) device. A particular box can both send message and receive them at the same time. A box can also react to message from itself. For example, a box that contains both a PIR sensor and a light could turn red whenever it detects a person in front of it. A box understands commands that specify what sensors it is using and how they are connected. You can also tell a box what messages to send, when to send them and who to send them to.

The Connected Little Box software manages all the components in a box. All boxes run the same software, which is configured for that box. The settings for a box are stored as a number of name-value pairs. These can be entered directly by connecting a serial terminal directly to the USB port on the box. A box can also be configured over WiFi via a web interface hosted by the box on an internal access point. Connected Little Boxes use MQTT to communicate with each other and other devices. The MQTT connection supports usernames and passwords and can be configured to operate over secure sockets (SSL). It is possible to update settings remotely over an MQTT connection.

Connecting to a box

You control and configure a Connected Little Box directly via a serial port connection the box via a USB cable. You can use a local terminal program to connect to a box or you can use the Simpleterm web page on the Connected Little Boxes web site. You can find the Simpleterm web page here:

```
https://www.connectedlittleboxes.com/simpleterm.html
```

Box settings

Settings in a Connected Little Box are changed by assigning new values to them:

```
wifissid1=myHomeWiFi
```

Entering the command above would set the WiFi name for connection number 1. If you enter this command via the serial port the Connected Little Box will configure this setting:

```
Processing: wifissid1=myhomewifi
value set successfully
```

Settings are stored inside the box and will be retained if the box is disconnected from power. You can also view the contents of any settings by entering the name of the setting:

```
wifissid1
WiFiSSID1 [wifissid1]: myhomewifi
```

The box will not show you the value of any password settings:

```
Processing: wifipwd2
    WiFiPassword2 [wifipwd2]: *****
```

You can get a full list of all the commands and their values by using the dump command:

```
Processing: dump

pirsensorfitted=no
pirsensorinputpin=4
...
otaupdateprodkey=

Dump complete
```

You can add a filter to the dump command if you only want to view particular settings. Only settings that contain the filter string will be displayed:

```
Processing: dump wifi

switchinputwificonfig=0
wifiactive=yes
wifissid1= myhomewifi
wifipwd1=*****
wifissid2=
wifipwd2=*****
wifissid3=
wifipwd3=*****
wifissid4=
wifipwd4=*****
wifissid5=
wifipwd5=*****

Dump complete
```

You can find a complete list of all the settings in a box in Appendix 1 at the end of this document.

Box processes and sensors

A box contains several cooperating processes each of which is responsible for a particular behaviour or data stream. Commands to the processes are expressed as JSON messages.

A message will always contain a **process** property and a **command** property, followed by command specific properties. Commands can be performed directly on a device or sent to another one by using the **to** attribute.

```
{"process":"pixels","command":"setnamedcolour","colourname":"orange"}
```

This is a command to the pixels process to set the pixels on the box to the named colour orange.

Commands can also be triggered by sensors. This is achieved by adding **sensor** and **trigger** attributes to a command:

```
{"process":"pixels","command":"setnamedcolour","colourname":"red","sensor":"pir","trigger":"triggered"}
```

This command would set the colour of the pixels to red whenever a pir sensor connected to the box is triggered. A box automatically connects to a time service when it starts running and commands and be bound to alarms, timers or regular events.

Data values

Some commands accept data values as either a string of text for printing or display or a numeric value which can be used to configure a setting. The text or value can be entered directly into a command:

```
{"process":"pixels","command":"brightness","value":0.5}
```

This command would set the brightness of the pixels to the value 0.5. The value is between the values of 0 and 1 inclusive, so the above command would set the pixels to half brightness. Sensors can also produce values which can be used as data values:

```
{"process":"pixels","command":"brightness","sensor":"pot","trigger":"turned"}
```

This command uses the potentiometer (pot) to set the brightness of the pixels. Each time the potentiometer is turned it will generate a new value which is used to set the brightness. Note that for this command there is no need to supply a value when the command is entered. The value is supplied with the sensor is triggered.

Input sensors such as the potentiometer and the rotary encoder will deliver values which have been scaled into the range 0-1. Digital (on or off) sensors such as the PIR sensor and the button will deliver values of 1 or 0 depending on whether they have been triggered or not. You can configure a range of values for an environmental sensor that specify the maximum and minimum values for that sensor. The sensor will then deliver a value between 0 and 1 that is scaled between those rangers.

Text messages

Some process commands can be supplied with a string of text for output in some form:

```
{"process":"console","command":"reporttext","text":"box running"}
```

This statement would send the message "box running" out of the serial port on the device.

Input sensors will provide a text string that describes their value. This will not be scaled in the range 0 to 1 as a value would be, so you can use this format for reporting sensor values:

```
{"process":"console","command":"reporttext","sensor":"bme280","trigger":"tempsec"}
```

The above command would cause a box to report the temperature value to the console every second. The value would be reported as the actual value:

```
27.1
27.1
27.1
```

The display and printing commands allow this value to be prefixed and postfixed with text, see the command descriptions for details.

Stored commands

A box contains named command stores that can be populated with commands. The commands in a store can be invoked directly, in response to sensor triggers or in response to system events. A command can be added to a store by adding a **store** property that specifies the store to be used and an **id** for that command in the store.

```
{"process":"pixels","command":"setnamedcolour","colourname":"red","sensor":"pir","
trigger":"triggered","store":"start","id":"alert"}
```

The statement above would add the command to a store called **start**. The command would have the id **alert**. The start command **store** is special, in that commands in this store are performed automatically when the box is started. This provides a way of giving boxes automatic behaviours. There are also **wifi**, **mqtt** and **clock** stores that can contain commands to be performed when a box connects to wifi, mqtt or the clock server.

---

pixels process

A connected Little Box can control a large number of pixels. The pixels settings for a box are as follows:

```
pixelcontrolpin=0
noofxpixels=12
noofypixels=1
```

```
pixelconfig=1
pixelbrightness=1.00
```

The **pixelcontrolpin** value sets the pin to be used to control the pixels that are connected to the processor.

The dimensions of the pixel display are set using two settings values **noofxpixels** and **noofypixels** If the number of y pixels is set at 1 the display wraps around so that lights that move off one end will reappear on the other. This allows the box to control pixel rings and loops with wrapping movement.

By default the pixels are configured as above, which is a 12 pixel chain or ring. If the **noofxpixels** value is set to 0 the pixel output is disabled.

There are two possible hardware configurations for the pixels which are set using the **pixelconfig** setting. Option 1 sends the colour data to the pixels as GRB (green, red, blue) and option 2 sends the colour data as RGB. If the colours that are displayed look wrong, try changing to the other setting. The default setting is option 1 – GRB.

The **pixelbrightness** value can be set between 0 and 1 to set the overall brightness of the pixels. This value is used to scale all the intensity values before they are displayed on the output.

The display is managed by the software as a number of "sprites" which have a particular colour and position on the pixels. A display can contain up to 25 sprites in the present version of the software. If there are more leds than sprites (for example of you have a large chain of pixels) the sprites are distributed evenly over the display. For small displays (for example a 4x4 text display) it is possible to display a sprite on every led. The number of sprites is calculated automatically by the software depending on the overall size of the display, up to the maximum of 25.There are two possible sprite animations: walking and mask.

Colour Test

If you want to view each colour by name and view it on the pixels you can use the command **colours** on the serial connection to display each colour and the name of that colour in turn:

```
Processing: colours Press space to step through each colour
Press the ESC key to exit
Colour:black
Colour:red
Colour:green
Colour:blue
Colour:yellow
Colour:orange
Colour:magenta
Colour display ended
```

Each named colour is displayed in turn. Note that during the colour display the connected little box will not respond to incoming messages or generate sensor events.

These are all the sprite commands:

setcolour - Sets the colour of the pixels

This command sets the colour of all the sprites being displayed on the pixels. The colour intensity values are set in the range 0-1. The rate at which the colour changes from the existing colour to the new one is set with the **steps** property.

Property	Value	Range	Default value	Type
red	amount of red	(0-1)		float
blue	amount of blue	(0-1)		float
green	amount of green	(0-1)		float
steps	no of 50Hz steps to complete the change	positive int	20	int

The following command sets the colour of the pixels to half brightness red over a period of 1 second.

```
{"process":"pixels","command":"setcolour","red":0.5,"green":0,"blue":0,"steps":50}
```

setnamedcolour - Sets the pixels to a named colour

This command sets the colour of all the sprites being displayed on the pixels. The colour is set to one of a number of named colours.

Property	Value	Range	Default value	Type
colourname	name of the colour to set			text
steps	no of 50Hz steps to complete the change	positive int	20	int

These are the named colours that are available.

black	red	green	blue	yellow	orange	orchid
cyan	white	azure	blueviolet	brown	chartreuse	darkgoldenrod
darkgreen	darkmagenta	darkorange	darkred	darkturquoise	darkviolet	deeppink
deepskyblue	firebrick	forestgreen	gold	indianred	lawngreen	lightseagreen
limegreen	maroon	mediumblue	mediumspringgreen	mediumviolet	midnightblue	navy
orchid	purple	saddlebrown	salmon	seagreen	springgreen	teal
tomato	violet					

Sets the colour of the pixels to orange over a period of .4 of a second (that is 20 steps)

```
{"process":"pixels","command":"setnamedcolour","colourname":"orange"}
```

setrandomcolour - Sets the pixels to a random colour

This command sets the colour of the pixels to a random colour which is determined by the current time.

Property	Value	Range	Default value	Type
steps	no of 50Hz steps to complete the change	positive int	20	int
options	**timed** – random colour is seeded from the time in minutes			

The random colour sequence is seeded by the current minute of the time value. This makes it possible to have a number of different light systems all displaying the same random sequence of synchronised colours.

```
{"process":"pixels","command":"setrandomcolour","options":"timed"}
```

This command sets a random colour on the pixels. Any devices that perform this command in the same minute will also display the same colour.

twinkle - Sets the pixels to twinkle random colours

This command sets the colours of the lights displayed by the pixels to random colours. Each light will have a random colour. The timed option means that the random sequence of colours will be seeded from the current minute.

Property	Value	Range	Default value	Type
steps	no of 50Hz steps to complete the change	positive int	20	int
options	**timed** – the random colour is seeded from the time in minutes			

```
{"process":"pixels","command":"twinkle","options":"timed"}
```

This command sets a random colour on each of the sprites. Any devices that perform this command in the same minute will display the same colour sequence on their sprites. Note that some devices will display more colours if they have more sprites active on their display.

brightness - Sets the pixel brightness

Sets the overall brightness of the pixels. Can be used to fade the pixels on and off if you set a large steps value.

Property	Value	Range	Default value	Type
value	brightness level	(0-1)		float
steps	no of 50Hz steps to complete the change	positive int	20	int

```
{"process":"pixels","command":"brightness","value":0,"steps":500}
```

Fades down the over 10 seconds. Note light fades and colour changes can be used together. A light can change colour and brightness at the same time.

pattern - Sets the pixel pattern

Set a pattern on the leds. The pattern can be either "walking" or "mask".

Property	Value	Range	Default value	Type
steps	no of 50Hz steps to complete the change	positive int	20	int
pattern	**walking** – a number of coloured sprites walk over the display **pattern –** a static pattern of sprites is displayed. Each sprite is a different colour			
colourmask	the colour mask to be used. See the description below			

Walking Sprite Display

In a walking display the sprites move over the display area. The sprites can all be the same colour or each can be different. The colours of the sprites in a walking display be set for each sprite by using a colour mask.

Colour mask

The colour mask sets the colour of individual sprites to a fixed colour. This is useful if you have a letter display and you want to highlight particular words.

	0	1	2	3
0	R	E	D	
1	N	O	S	E
2	D	A	Y	
3	2	0	2	1

The above table shows a led arrangement for a sign that contains text. The display has been defined by setting **noofxpixels** to 4 and **noofypixels** to 4. We would like to set the colour of the pixels at the different positions so that the word "RED" is red, the word "NOSE" is green, the word "DAY" is blue and the number digits are displayed in orange, yellow, cyan and magenta respectively. We can create a mask that describes these requirements. The mask that defines the colours is a string with a single character expressing the colour of that element in the display.

```
"RRRKGGGBBBKOYCM"
```

The mask is applied starting at the top right-hand corner of the raster (location 0,0) and then moving along each row. At the end of the row the mask moves back to the start of the next one. If the mask string is longer than the number of sprites on the display the excess colour characters are ignored. If the mask is shorter than the number of sprites it will be repeated.

```
"RB"
```

The mask string above would set the display to alternate red and blue sprites. The colour characters that are available are as follows:

K	black
R	red
G	green
B	blue
Y	yellow
O	orange
M	magenta
C	cyan
W	white
V	violet
P	purple
T	teal

You can use a mask to set the colours in a mask command or in a walking sprite command. If the number of pixels is less than 25 (for a letter grid) each sprite is mapped directly onto a pixel.

walking pixel pattern

The walking pattern "walks" several lights over the leds. The mask pattern distributes several coloured sprites over the leds.

```
{"process":"pixels","command":"pattern","pattern":"walking","colourmask":"RGBY}
```

The above command sets a walking pattern of red, green, blue and yellow over the sprites. The colour sequence will be repeated over all the sprites.

mask pixel pattern

The mask pattern maps the coloured mask directly onto the sprites to create a static light display.

```
{"process":"pixels","command":"pattern","pattern":"mask","colourmask":"RB"}
```

The above command sets alternate red and blue sprites along the length of the leds.

map - Sets the pixel colour to a mapped value

This command maps a numeric value onto a particular pixel colour in a colour mask. The command picks the colour character from the array at the position that matches the numeric value delivered to command.

Property	Value	Range	Default value	Type
value	Value to map	(0-1)		float
steps	no of 50Hz steps to complete the change	positive int	20	int
options	**mix** – the colour of the light is interpolated between the adjacent pixels in the mask. If the mix option is not set the colour of the light is set to the "nearest" colour in the colourmask.			
colourmask	The string of colour characters into which the range will be mapped.			

```
{"process":"pixels", "command":"map","colourmask":"RGB","value":0.5,
"sensor":"pot","trigger":"turned"}
```

The above command would map the value 0.5 into the colour mask "RGB". This would cause the pixels to display the green colour. The command is bound to a potentiometer and will receive numeric values if the potentiometer is moved. If the potentiometer is turned to the minimum position the light will display red. If the potentiometer is turned to the maximum position the light will display blue.

The **mix** option has not been selected which means that that colours will change directly from one to the other as the data values moves through the range. If the mix option had been selected the light would display yellow as the value increased from 0 and cyan as the value increased from 0.5.

---

console process

A Connected Little Box will respond to commands sent via a serial connection. It is also possible to access the console using the remote command. This command will trigger the command output on the box which obeys the command. If a box sends a console command to another box the command output will not be displayed.

remote - Perform a remote console command

Property	Value
commandtext	Text of the console command to be obeyed by the device.

The command is performed exactly as if it had been typed into the console. This can be useful for remote management of the device.

```
{"process":"console","command":"remote","commandtext":"noofxpixels=12"}
```

This command would set the number of x pixels to 12.

reporttext – Reports a message

This command reports a message to the console.

Property	Value	Default value	Type
text	message text to be reported		text
pre	text to be displayed before the message text	empty string	text
post	text to be displayed after the message text	empty string	Text

This command can be used to report a message on the console. A connected device (for example a PC on a USB connection) could use the message to trigger an action.

```
{"process":"console","command":"reporttext","text":"box running"}
```

This would send the message "box running" to the serial console. This command can also be used to report the contents of a value:

```
{"process":"console","command":"reporttext","text":"99.9","sensor":"bme280","trigg
er":"tempsec","pre":"Temp:","post":" degrees"}
```

This command would report the temperature string from the BME280 sensor on the console at one second intervals:

```
Temp:23.3 degrees
Temp:23.3 degrees
Temp:23.3 degrees
```

Note that the initial value of 9

reportjson – Reports a message on the console as a JSON object

This command reports a message as a JSON object. You must specify the attribute name for the message that is sent.

Property	Value	Default value	Type
value	value to be reported		Text
attr	JSON attribute for the value		Text

This command can be used to generate messages but it is intended to be used so that a device can report values via the console to a connected device.

```
{"process":"console","command":"reportjson","text":"starting","sensor":"bme280","trigger":"tempsec","attr":"temp"}
```

This would send a temperature value every second to the console. The value is encoded in a JSON object and the temperature value has the attribute **temp**:

```
{"temp":"24.0"}
{"temp":"24.0"}
{"temp":"24.0"}
```

---

outpin process

The outpin process controls a digital output pin from box. This can be used to control digital outputs, for example lights or relays. The pin can be configured by the following settings.:

```
outpin=2
outpinactivehigh=yes
outpinactive=yes
outpininitialhigh=no
```

The output is initially not active. It can be enabled by changing the **servoactive** setting to yes. The **outpin** setting sets the pin to be used. Note that this is the pin number on the device itself, not the connector. For example on the Wemos D1 Mini device the default pin above (pin number 2) is connected to the pin labelled D4 which is actually pin 17 on the physical device. The **outpinactivehigh** setting determines whether the pin is set to low or high when active. Some devices contain circuits that invert the polarity of output pins, this setting is used to allow for this. The **outpininitialhigh** setting determines the initial state of the pin when the box is switched on.

setoutpinstate - Sets the state of the outpin

The value is used to set the state of the output pin. If the input value is greater than or equal to 0.5 the output is set to high.

Property	Value	Range	Default value	Type
value	output state	(0-1)		Float

```
{"process":"outpin","command":"setoutpinstate","value":1}
```

The command above sets the output pin to the high state.

pulseoutpinstate - Pulse the outpin

The value is used to pulse the output pin. The output will be set according to the input value and then return to its previous position after the specified pulse time.

Property	Value	Range	Default value	Type
value	output state	(0-1)		Float
pulselen	pulse length in seconds	(0-10)		Float

```
{"process":"outpin","command":"pulseoutpinstate","value":1,"pulselen":2.5}
```

The command above moves the servo to position 1 which will be one end of the range. The value of **pulselen** specifies a length of time in seconds that the output should be held before returning to the previous level.

setinitoutpinstate - Sets the initial state of the outpin

The value is used to set the initial state of the output pin (i.e. the state when the box is first powered on). The range of 0-1 provides movement over the full extent of the servo range.

Property	Value	Range	Default value	Type
Value	servo position	(0-1)		float

```
{"process":"outpin","command":"initialoutpinstate","value":0}
```

Set the output to the low state when the box starts.

---

servo process

The servo process controls a servo connected to a box. The servo can be configured as follows:

```
servooutputpin=12
servoinitialangle=0.00
servoactive=no
```

The servo is initially not active. It can be enabled by changing the **servoactive** setting to yes.

setservopos - Sets the position of the servo

The value is used to set the position of the servo. The range of 0-1 provides movement over the full extent of the servo range.

Property	Value	Range	Default value	Type
value	servo position	(0-1)		Float

```
{"process":"servo","command":"setservopos","value":1}
```

The command above moves the servo to position 1 which will be one end of the range.

pulseservopos - Pulse the position of the servo

The value is used to pulse the position of the servo. The range of 0-1 provides movement over the full extent of the servo range. The servo will move to the position specified by the value and then return to its previous position after the specified pulse time.

Property	Value	Range	Default value	Type
value	servo position	(0-1)		Float
pulselen	pulse length in seconds	(0-10)		Float

```
{"process":"servo","command":"pulseservopos","value":1,"pulselen":2.5}
```

The command above moves the servo to position 1 which will be one end of the range. The value of **pulselen** specifies a length of time in seconds that the servo should hold that position before returning to the previous position.

setinitservopos - Sets the initial position of the servo

The value is used to set the initial position of the servo (i.e. the position the servo moves to when the box is first powered on). The range of 0-1 provides movement over the full extent of the servo range.

Property	Value	Range	Default value	Type
Value	servo position	(0-1)		float

```
{"process":"servo","command":"setinitservopos","value":0.33}
```

Move the servo to position 0.33 (a third of the way round) when the box starts.

registration process

This process performs registration commands on a device. These commands will result in messages being sent over MQTT to the registration topic on the MQTT server. The commands do not have any properties.

register - Perform a remote Registration command

This command will cause the device to register on the remote server.

```
{"process":"registration","command":"register"}
```

This will cause the device to register with the server host.

getsetup - Get the device setup

```
{"process":"registration","command":"getsetup"}
```

getsettings - Get the settings for a process or sensor

name - name of item : text

```
{"process":"registration","command":"getsettings","name":"pixels"}
```

Get the settings in json for the pixels process.

max7219 process

The max7219 process controls a MAX7219 display. The MAX7219 can be configured as follows:

```
max7219Messagesactive=no
max7219defaultMessage=Hello!
max7219datapin=4
max7219cspin=5
max7219clockpin=15
max7219xdev=4
max7219ydev=1
max7219stickymessageMS=30000
max7219frametimeMS=1000
max7219framedelay=0.10
max7219brightness=0.50
```

The **max7219Messagesactive** setting must be set to **yes** for the printer to be enabled. The pin settings can be configured to match the connections to your display. You can also set the configuration of the 8x8 display modules, which can be arranged in a grid. However the present version of the software only uses a row of devices (i.e. the value of **max7219ydev** should be left at 1. Later versions of the software may allow for positioning of text on the display matrix.

The **max7219stickymessageMS** determines how long "sticky" messages persist on the display before they can be replaced by another one.

The **max7219frametimeMS** value sets the maximum time in milliseconds between scroll events. The default value is 1000 milliseconds. This value is used in conjunction with the **max7219framedelay** value which sets the fraction of the **max7219frametimeMS** to be used as the delay value between scrolls. This rather complex configuration is used because you can bind the scroll speed to a value from a sensor or control and you may want to control the range of speed values available from that control or sensor.

The **max7219brightness** setting controls the brightness of the display. The maximum brightness is 1, the display will be off of the value is set to 0. This setting can be bound to a sensor or control which can then control the brightness of the display.

display - Displays a message

This command displays a message on the Max7219.

Property	Value	Default value	Type
text	message text to be displayed		text
pre	text to be displayed before the message text	empty string	text
post	text to be displayed after the message text	empty string	text
options	**scroll** – message will scroll across the display **small** – the messages will be displayed in small text **sticky** – the message will persist on the screen		

The **scroll** option causes the display to scroll. It can be used to display larger amounts of text that would fit on the display. The **small** option selects a smaller font for the display. The **sticky** option causes the message to stay on the display for an amount of time set by the **max7219stickymessageMS** setting. Any other display operations will be ignored while a sticky message is being displayed. A sticky message can be replaced by another sticky message.

```
{"process":"max7219","command":"display","text":"rob"}
```

This would display the message "rob"

```
{"process":"max7219","command":"display","text":"great big
message","options":"scroll,small","pre":"pre*","post":"*post"}
```

This would display the message "pre*great big message*post" in small font and scrolling. The **pre** and the **post** options are useful when displaying sensor values.

default - Sets the message displayed at power on

This command sets the default text on the display, i.e. the text displayed when the box is first switched on. By default this is the message "Hello!". If you want the display to be blank on start-up, display an empty string.

Property	Value	Default value	Type
text	message text to be displayed on power up		text

```
{"process":"max7219","command":"display","default":"On"}
```

This command would display the message "On" when the box is powered up.

scrollspeed - Scroll speed (0-1)

The scroll speed is set according to the supplied value. The value gives the fraction of the frame time that the display will delay for each scroll frame.

Property	Value	Range	Default value	Type
value	scroll speed	(0-1)		float

The frame time is set via the setting **max7219frametimeMS** which is initially set to 1000. This means that the scroll speed value gives fractions of a second. If you want to have very slow scrolling text you can make the **max7219frametimeMS** value very large.

```
{"process":"max7219","command":"scrollspeed","value":0.5}
```

This would cause the scroll to advance every half second with the default frame time.

brightness - Brightness (0-1)

Sets the brightness of the display. A brightness value of 1 is full brightness. A brightness value of 0 is off.

Property	Value	Range	Default value	Type

value	Display brightness level	(0-1)		float

```
{"process":"max7219","command":"brightness","value":0.1}
```

This command would set the brightness of the display to quite dim.

showvalue - show a value

This command displays a floating-point value on the display.

Property	Value	Default value	Type
value	numeric value to be displayed		float
pre	text to be displayed before the message text	empty string	text
post	text to be displayed after the message text	empty string	text
options	**scroll** – message will scroll across the display **small** – the messages will be displayed in small text **sticky** – the message will persist on the screen		

```
{"process":"max7219","command":"showvalue","value":18.1,"pre":"temp ","post":"
degrees","options":"small,scroll"}
```

This will display the following scrolling message in a small font:

```
temp 18.1 degrees
```

This command can be bound to a sensor or input trigger event. When the event fires the command will be performed with the content of the value variable set to the current sensor reading.

---

printer process

You can connect a printer so that a box can produce printed output. The printer is configured by the following settings.

```
printeron=no
printerbaud=19200
printerdatapin=16
```

The **printeron** setting determines whether the printer is available. It must be set to "yes" before the printer can be used.

The **printerbaud** setting determines the rate at which data is sent to the printer. The value above works with my thermal printer but you may need to change it for your device.

The **printerdatapin** setting is used to set the GPIO pin that is to be used for the serial connection. Note that on the ESP8266 the only pin that can be used is pin 16.

print – print a message

Use this command to print a message on the printer. The text of the message is set when the command is performed. It will be replaced by text from a sensor trigger.

Property	Value	Default value	Type
text	message text to be printed		text
pre	text to be printed before the message text	empty string	text
post	text to be printed after the message text	empty string	text
options	**sameline** – the printer will not take a new line after the print **datestamp** – a datestamp will be printed before the message		

The **datestamp** option precedes the printed output with a line giving the current date and time. If the box does not have a working network connection the date and time will not be printed. The **datestamp** and **sameline** options should not be used together as the timestamp will disrupt any layout.

```
{"process":"printer","command":"print","text":"rob","options":"datestamp"}
```

This will print the text "rob". Before the text is printed a datestamp line is printed. The output has the following appearance:

```
Mon Mar 22 2021 16:06:11
rob
```

---

outpin process

The **outpin** process lets you control the state of a single digital output pin. It is configured using the following settings:

```
outpin=16
outpinactivehigh=yes
outpinactive=no
```

The **outpin** setting determines the pin to be used for output. The **outpinactivehigh** setting lets you set the state of the pin when it is high. The default setting is that the pin is set high when the output is active. The **outpinactive** setting determines whether or not the output pin is active. Note that the default output pin number is set to 16 which is the same pin as that used for the printer.

setoutpinstate

Use this command to set the state of the output pin.

Property	Value	Default value	Type
value	a value between 0 and 1 that determines whether the pin will be high or not. A value less than 0.5 will set the pin low. A value greater than or equal to 0.5 will set the pin high.		float

The value property can be set directly. If the state is being controlled by a sensor the value property can be omitted.

```
{"process":"outpin","command":"setoutpinstate","value":1}
```

The above statement would set the output pin to high.

setinitoutpinstate

Use this command to set the initial state of the output pin.

Property	Value	Default value	Type
value	a value between 0 and 1 that determines whether the pin will be high or not. A value less than 0.5 will set the pin low. A value greater than or equal to 0.5 will set the pin high.		Float
hold	time in seconds that the state will be held in the range 0-10	0	Float

The value property can be set directly. If the state is being controlled by a sensor the value property can be omitted.

```
{"process":"outpin","command":"setinitoutpinstate","value":1}
```

The above statement would set the output pin to high when the device is switched on. The value of **hold** specifies a length of time in seconds that the pin should hold that state before returning to the previous position. If the value is 0 or the **hold** property is omitted the new state is held indefinitely.

Directing process messages to another box using to

A box can send a command to another box by adding a **to** property to the command. The **to** property is followed by the destination topic that the receiver is listening on:

```
{"process":"pixels","command":"setnamedcolour","colourname":"orange",
"to":"CLB-b00808"}
```

The above command would set the pixels orange on a box with the device name **CLB-b00808**. Message transmission will only be attempted if the box is connected to the WiFi and MQTT host. If the destination device does not have a process that can respond to the command (for example the device **CLB-b00808** does not have any pixels) the command will be ignored when it is received.

When a box receives a message that has been sent in this way it will contain a **from** attribute that identifies the box that sent the message.

```
{"process":"pixels","command":"setnamedcolour","colourname":"orange",
"from":"CLB-ea7343"}
```

This would be the message received by box **CLB-b00808** if the originator was **CLB-ea7343**.

Sensors

Sensors can be either environmental sensors (for example a temperature sensor) or control inputs (for example a button or a rotary encoder). A sensor can be used to trigger an action when it has a new reading, or at regular intervals. Each sensor has a different set of trigger behaviours. There is also a clock sensor which can be used to generate sensor triggers at regular intervals or on alarm conditions.

Any command can be assigned a sensor and given a trigger setting. The sensor will then be set to "listen" for the given trigger and then perform the command when the trigger event occurs. A sensor can have multiple listeners assigned to it which can listen for the same or different triggers. The maximum number of listeners that can be assigned in a single connected little box is 10.

When a sensor triggers an event, it may set the value of the **value** and **text** elements of the command it is triggering. The **value** element is set to a normalised value between 0 and 1 of the sensor reading. This can be used to control the behaviour of a process. For example the output of a rotary encoder could be used to control the brightness of the pixel display. The **text** element may be set to a text description of the sensor data. This could be displayed or printed.

Pir sensor

The sensor uses a Passive Infrared Sensor to detect human presence. The driver was written for an HC-SR501 PIR Sensor device in its default configuration. These are the settings in a box for this sensor.

```
Pirsensorfitted=no
pirsensorinputpin=4
piractivehigh=yes
```

By default the sensor is disabled, you enable it by setting **pirsensorfitted** to **yes**. The default input pin is GPIO 4. The setting **piractivehigh** allows a box to be adapted for a sensor that provides an active low signal.

Pir sensor test

The pirtest command can be used to test the PIR sensor. When the command is issued the connected little box will repeatedly check the input from the PIR sensor and print a message when the input changes state. Press any key to end the test.

```
Processing: pirtest PIR Sensor test
Press the ESC key to end the test
    triggered: 1
PIR test ended
```

Note that during the PIR test the connected little box will not respond to incoming messages or generate sensor events.

Trigger:changed

This trigger is raised when the state of the PIR sensor changes. The value element is set to 0 if the sensor is clear or 1 if the sensor has been triggered. The text element is set to "triggered" or "cleared" to reflect the new status of the sensor.

```
{"process":"max7219","command":"display","text":"starting","sensor":"pir","trigger
":"changed","to":"command/CLB-3030da","options":"sticky,small"}
```

This command would send a message to a connected little box with the address "command/CLB-3030da". The box should be fitted with a max7219 display and will be updated each time the PIR status changes. The message would be sticky (i.e. it would remain on the display for a while) and it uses the small font. The command would display the message "starting" until the first message was received from the PIR sensor.

```
{"process":"pixels","command":"brightness","value":0,"steps":20,
"sensor":"pir","trigger":"changed","to":"command/CLB-eab714"}
```

This command would send a message to a connected little box with the address "command/CLB-eab714". When the pir was triggered the brightness of the leds would be set to 1, when the pir was clear the leds would be set to 0. If you want more control over the precise brightness when the sensor is triggered and when not, you can assign pixel controls to the triggered and cleared events.

Trigger:triggered

This trigger is raised when the PIR sensor detects a movement. The value element and the text element of the triggered command are not changed by this trigger.

```
{"process":"printer","command":"print","text":"Door 1","options":"datestamp",
"to":"command/CLB-b00808","sensor":"pir","trigger":"triggered"}
```

This command would print the message "Door 1" on a printer connected to the box with the address "command/CLB-b00808" each time the PIR sensor detected someone. The message would have a datestamp.

```
{"process":"pixels","command":"setnamedcolour","colourname":"red","steps":20,
"sensor":"pir","trigger":"triggered","to":"command/CLB-eab714"}
```

This command would send a message to a connected little box with the address "command/CLB-eab714". This command would set the colour of the pixel display to red when the PIR sensor was triggered.

Trigger:cleared

This trigger is raised at the end of a movement detection. The value element and the text element of the triggered command are not changed by this trigger.

```
{"process":"pixels","command":"setnamedcolour","colourname":"green","steps":20,
"sensor":"pir","trigger":"cleared","to":"command/CLB-eab714"}
```

This command would send a message to a connected little box with the address "command/CLB-eab714". This command would set the colour of the pixel display to green when the PIR sensor was cleared.

```
{"process":"servo","command":"setservopos","value":0.5,"sensor":"pir","trigger":"c
leared","to":"command/CLB-285627"}
```

This command would send a message to a connected little box with the address "command/CLB-285627". This command would set the servo position to 0.5 when the PIR sensor was cleared.

Button sensor

The button sensor reads the status of a push button. The button is implemented as a push to make connection that connects a GPIO pin to ground. The button wiring can be simplified by using another GPIO pin as the ground pin if a connection is not available.

```
Buttoninputpin=14
buttoninputgroundpin=16
pushbuttonfitted=no
```

If the ground pin is not required it can be disabled by setting **buttoninputgroundpin** to -1.

Button sensor test

The button sensor can be tested with the **buttontest** command. The command repeatedly tests the button state and displays a message each time that it detects a change.

```
Processing: buttontest Button Sensor test
Press the ESC key to end the test
    pressed: 1
    pressed: 2
    pressed: 3
Button test ended
```

Note that during the button test the connected little box will not respond to incoming messages or generate sensor events.

Trigger:changed

This trigger is raised when the state of the button changes. The value element is set to 0 if the button is up or 1 if the button is pressed. The text element is set to "down" or "up" to reflect the new status of the button.

```
{"process":"max7219","command":"display","text":"starting","sensor":"button","trig
ger":"changed","to":"command/CLB-3030da","options":"sticky,small"}
```

This command would send a message to a connected little box with the address "command/CLB-3030da". The box should be fitted with a max7219 display and will be updated each time the button status changes. The message would be sticky (i.e. it would remain on the display for a while) and it uses the small font. The command would display the message "starting" until the first message was received from the button.

Trigger:pressed

This trigger is raised when the button state changes from up to pressed. It does not set the value or text elements.

```
{"process":"printer","command":"print","text":"button pressed",
"options":"datestamp", "to":"command/CLB-b00808", "sensor":"button",
"trigger":"pressed"}
```

This command would print the message "button pressed" on a printer connected to the box with the address "command/CLB-b00808" each time the button was pressed. The message would have a datestamp.

Trigger:released

This trigger is raised when the button state changes from up to pressed. It does not set the value or text elements.

```
{"process":"servo","command":"setservopos","value":0.0,
"sensor":"button","trigger":"released","to":"command/CLB-285627"}
```

This command would set the servo to position 0 on a box with the address command/CLB-285627 when the button was released.

Clock sensor

The clock sensor is used to provide timed trigger events. There are three alarms. There are also two timers which can operate as single shot or repeating timers. The clock can also trigger events every second, minute, hour or day.

Trigger:alarm1 + alarm2 + alarm3

You can set these alarms to fire triggers at a particular time of the day. They are controlled by the following setting values:

```
alarm1hour=7
alarm1min=0
alarm1enabled=no
alarm1timematch=yes
```

The **alarm1hour** and **alarm1min** settings give the hour and minute when the alarm will fire, in 24 hour format.

The **alarm1enabled** setting is used to enable and disable the alarm.

The **alarm1timematch** determines how the alarm will fire. Normally the alarm will fire when the time matches the alarm time. For example the alarm above will fire the first time that the hour is 7 and the minute is 0. Sometimes you want the alarm to fire after the alarm time. For example, if you turn your central heating controller off and on in the middle of the day, you will still want the 7:00 am heating on event to be triggered so that your heating is on at that time. If you set the **alarm1timematch** setting to **no** this will make a 7:00 am alarm trigger if the box is restarted any time after 7:00. The settings for alarms 2 and 3 work in the same way.

```
Alarm1hour=11
alarm1min=45
alarm1enabled=yes
alarm1timematch=yes
```

```
{"process":"pixels","command":"setnamedcolour","colourname":"orange","sensor":"clo
ck","trigger":"alarm1"}
```

The settings and command above would cause the pixels to become orange at 11:45.

Trigger:timer1 + timer2

You can use the timers to cause triggers after a specified interval, or at regular intervals. They are controlled by the following setting values:

```
timer1=30
timer1enabled=no
timer1singleshot=yes
```

The **timer1** value sets the number of minutes that the timer is to run for. The **timer1enabled** value enables and disables the timer. If **timer1singleshot** is set to **no** the timer will repeatedly trigger at the specified interval. The settings for timer2 work in the same way.

```
Timer1=3
timer1enabled=yes
timer1singleshot=no
```

```
{"process":"pixels","command":"setnamedcolour","colourname":"green","sensor":"cloc
k","trigger":"timer1"}
```

The above commands would cause the pixels to turn green every three minutes.

Trigger:second

This trigger will raise an event every second. When the event is raised the text value will be set to the current time in the form "hh:mm:ss". This text can be displayed or printed.

```
{"process":"max7219","command":"display","text":"Start..","sensor":"clock","trigge
r":"second","options":"small"}
```

This command creates a max7219 powered clock that ticks once a second. Each time the second trigger fires the clock text is updated. The command uses the small font so that the time display will fit onto a 32 pixel wide display.

Trigger:minute

This trigger will raise an event every minute. When the event is raised the text value is set to the current time in the form "hh:mm"

```
{"process":"max7219","command":"display","text":"Start..","sensor":"clock","trigge
r":"minute"}
```

This command will create a max7219 powered clock which ticks once a minute.

Trigger:hour

This trigger will raise an event every hour. When the event is raised the text value is set to the current time in the form "hh:mm"

```
{"process":"printer","command":"print","pre":"Check:","text":"Start..","sensor":"c
lock","trigger":"hour"}
```

This command will print the message "Check:hh:00" on the printer every hour.

Trigger:day

This trigger will raise an event every hour. When the event is raised the text value is set to the current date in the form "dd:mm:yy"

```
{"process":"printer","command":"print","pre":"Log:","text":"Start..","sensor":"clo
ck","trigger":"day"}
```

This command will print the message "Log:dd:mm:yy" on the printer every day.

Rotary

The rotary sensor is used to provide rotary input to a box. The sensor will generate events when turned which can be used to adjust the relative position of a value. It will also generate events when the button on the sensor is pressed and released. The value of the output is clamped between 0 and 1.00, attempts to turn the value below 9 or beyond 1 will be ignored. The numeric output will range between 0 and 1 in steps of 0.01. The text output is a string from "0.00" to "1.00".

The sensor driver connects to a rotary sensor that provides a clock and a data signal. It has been tested with a KY-040 360 Degree Rotary Encoder without any additional conditioning of the clock and data signals and found to work reasonably well. It is configured using these setting items:

```
rotarysensordatapin=4
rotarysensorclockpin=5
rotarysensorswitchpin=0
rotarysensorfitted=yes
rotarysensorinitial=0.5
```

These are the default values. The pin settings are connected to the encoder. The switch pin is connected to the switch in the encoder. The encoder always provides a value which is relative to the previous position. There is no absolute positioning. The **rotarysensorinitial** setting allows you to set an initial rotary value which is returned when the device starts.

Trigger:turned

The turned event is triggered each time a sensor movement is detected.

```
{"process":"max7219","command":"display","text":"Hello","sensor":"rotary","trigger
":"turned","to":"CLB-2feada"}
```

This command will display the new position of the rotary encoder on the display with the address **CLB-2feada**. Note that a lot of rotary movement will result in a large number of messages being transmitted.

Trigger:pressed

The pressed event is triggered when the user press down on the shaft of the rotary encoder. In this respect the rotary input works in exactly the same way as a push button.

```
{"process":"pixels","command":"setnamedcolour","colourname":"green",
"sensor":"rotary","trigger":"pressed"}
```

The above statement would cause the pixels on the box to turn green when the shaft of the encoder was pressed in.

Trigger:released

The released event is triggered with the user releases a shaft that has been held down.

```
{"process":"pixels",
"command":"setnamedcolour","colourname":"magenta","pixelSpeed":20,
"sensor":"rotary","trigger":"released"}
```

The above statement would cause the pixels on the box to turn magenta when the shaft of the encoder was released.

---

pot

The potentiometer can also be used to read rotary input. It can only be turned through 270 degrees, but the values that are returned are absolute within this range as the potentiometer will be in a particular physical position. The sensor triggers a new value when it detects a change in the potentiometer position. The numeric output will range between 0 and 1 in steps of 0.01. The text output is a string from "0.00" to "1.00". The potentiometer is configured using the following settings:

```
potsensorfitted=no
potsensordatapin=0
potsensormillisbetween=100
potsensordeadzone=10
```

The **potsensordatapin** specifies the analogue data pin to be used to read the value from the potentiometer. This uses a different numbering scheme from the GPIO pins. The **potsensormillisbetween** sets the maximum rate at which position inputs are transmitted to commands that bind to the turned trigger. A setting of 100 milliseconds (the default) means that the sensor will only send 10 readings per second, even if the position signal is changing more rapidly. The **potsensordeadzone** value sets the minimum change in the signal that must be detected for

the sensor to trigger. The default value is 10 in an analogue integer reading in the range 0-1023. If output is found to be noisy it can be filtered by increasing this value.

Trigger:turned

The turned event is triggered each time that the pot is moved.

```
{"process":"pixels","command":"brightness","value":0,"steps":5,
"sensor":"pot","trigger":"turned","to":"CLB-ae894c","store":"mqtt","id":"dim2"}
```

The statement above allows a potentiometer to be used to control the brightness of the leds of the box **CLB-ae894c**.The command would be stored in the device and become active when the device had made an MQTT connection.

BME280

The BME280 provides temperature, humidity and air pressure as values that can be displayed or transmitted. Note that when an event fires the float value is in the range 0-1 (so that it can be used to control another Connected Little Box device directly). However, the text value is set to a string that describes the value and can be used for display, printing or logging. The sensor is configured by the following setting values:

```
bme280sensorfitted=yes
bme280envnoOfAverages=25
bme280tempchangetoxmit=0.500000
bme280tempbasenorm=10.000000
bme280templimitnorm=30.000000
bme280presschangetoxmit=10.000000
bme280pressbasenorm=0.000000
bme280presslimitnorm=100.000000
bme280humidchangetoxmit=2.000000
bme280humidbasenorm=0.000000
bme280humidlimitnorm=100.000000
```

The **bme280envnoOfAverages** setting determines the number of averages of a reading that are used to create each reading. There are three settings for each of the three data types: temperature, pressure and humidity.

The **bme280tempchangetoxmit** setting determines the change in the setting value before the **tempchange** trigger is generated. The **bme280tempbasenorm** and **bme280templimitnorm** control how a temperature value is mapped onto a range that can then be used to control other processes. Any temperature below the base value will be mapped to the value 0. Any temperature above the limit value will be mapped to the value 1. A value in the range between the base and the limit will mapped into the range 0-1 proportionally.

Using the default settings any temperature less than 10 would be mapped to 0. A temperature greater than 30 would be mapped to 1. A temperature in the range 10-30 would be mapped onto the range 0-1. For example, the temperature 20 would be mapped to the value 0.5 because 20 is half way between 10 and 30.

These are the settings for pressure and humidity are used in the same way.

Trigger:tempsec,tempmin,temp5min,temp30min,temphour

The sensor will trigger and generate new temperature values on the given intervals.

Trigger:tempchanged

The sensor will trigger and generate a new humidity reading when the value of the temperature changes outside the **bme280tempchangetoxmit** range.

Trigger:presssec,pressmin,press5min,press30min,presshour

The sensor will trigger and generate new pressure values on the given intervals.

Trigger:presschanged

The sensor will trigger and generate a new humidity reading when the value of the temperature changes outside the **bme280presschangetoxmit** range.

Triggers:humidsec,humidmin,humid5min,humid30min,humidhour

The sensor will trigger and generate new humidity values on the given intervals.

Trigger:humidchanged

The sensor will trigger and generate a new humidity reading when the value of the humidity changes outside the **bme280humidchangetoxmit** range.

Trigger:allsec

The sensor will trigger and generate a summary of all the values as a JSON encoded text string:

```
{"humid":47,"temp":24.5,"press":1014}
```

This text can be displayed or printed as any other text string, but it is intended to be used with the console **reportjson** command to allow a box to send composite environmental information as a single block of data.

```
{"process":"console","command":"reportjson","text":"starting","sensor":"bme280","t
rigger":"allsec","attr":"env"}
```

This would cause a box with an enabled BME280 sensor to send the following message to the console report each second:

```
{"env":{"humid":47,"temp":24.5,"press":1014}}
```

Trigger:allmin,all5min,all30min,allhour

These triggers will cause the sensor output to be generated at the specified intervals.

Command Stores

Commands can be stored in "command stores" inside a Connected Little Box. You can use this to group commands together so that they can all be triggered as a group. When a command is stored the name of the store to use (using the **store** property) and the id of the command to be stored (using the **id** property) are specified.

```
{"process":"pixels","command":"setnamedcolour","colourname":"red",
"store":"start","id":"red"}
```

The command above creates a command that sets the pixels to red and stores this command in a store called **start**. The id for the stored command is **red**. The **start** command store is special, in that any commands in the start store are automatically performed when a Connected Little Box is started. The result of the above command would be that the box would always start up displaying a red colour.

Display command store contents

The stores command will show the contents of all the command stores.

```
stores
```

Each of the stores and the commands in the stores will be displayed. If the red command had been created as above the display would be as follows:

```
Processing: stores
 Store:start
   Command:red
      {"process":"pixels","command":"setnamedcolour","colourname":"red"}
```

Performing store contents

A store can contain many commands, all of which will be performed when that store is executed. The controller process can be used to perform commands from a given store. The **perform** command requests that a store be performed; the **store** property identifies the store to use.

```
{"process":"controller","command":"perform","store":"start"}
```

The command above would perform all the commands in the store called **start**.

Delete stored commands

You can delete a stored command by using the console command **deletecommand**. The command is followed by the id of the command to be deleted. The command searches all the command stores for the specified command and removes it.

```
deletecommand red
```

The command above would delete the command with the id "red". The following message would be displayed:

```
Removing:/start/red
```

Command store events

There are five "special" stores where you can put commands that you want to be performed whenever a particular event occurs on the device.

Store name	Event description
`start`	runs when the Connected Little Box starts
`wifion`	runs when the Connected Little Box has a WiFi connection.
`wifioff`	runs when the Connected Little Box loses the WiFi connection.
`clock`	Commands in this store will run as soon as the Connected Little Box has acquired the correct date and time from the network
`mqtt`	Commands in this store will run as soon as the Connected Little Box has connected to MQTT.

You can use these events to create initial settings and allow the box to respond to changes in status. For example, a box could turn green when it gets a network connection and red when the network fails.

```
{"process":"pixels","command":"twinkle","options":"timed",
"sensor":"clock","trigger":"minute","store":"start","id":"twinkle"}
```

The above command displays random twinkling colours. It has been placed in the "start" store so the colour display will begin when the device starts running.

Things to remember about command stores

Command stores are a very powerful feature that are still being developed.

- It is possible for a command store to contain commands that run commands in other command stores. This is powerful but potentially confusing.
- You don't know the order in which the commands in a store will be performed.
- Commands in a command store can be directed to other boxes.
- You can direct a remote box to perform the commands in a particular store.
- If a command in a store directs the box to perform that store the device will then get stuck in a loop. Which will not end well.

Appendix 1 – Settings List

```
pirsensorfitted=no
pirsensorinputpin=4
piractivehigh=yes
buttoninputpin=14
buttoninputgroundpin=16
pushbuttonfitted=no
timezone=Europe/London
alarm1hour=7
alarm1min=0
alarm1enabled=no
alarm1timematch=yes
alarm2hour=12
alarm2min=0
alarm2enabled=no
alarm2timematch=yes
alarm3hour=19
alarm3min=0
alarm3enabled=no
alarm3timematch=yes
timer1=30
timer1enabled=no
timer1singleshot=yes
timer2=30
timer2enabled=no
timer2singleshot=yes
rotarysensordatapin=4
rotarysensorclockpin=5
rotarysensorswitchpin=0
rotarysensorfitted=no
rotarysensorinitial=0.500000
potsensordatapin=0
potsensorfitted=no
potsensormillisbetween=100
potsensordeadzone=10
bme280sensorfitted=no
bme280tempchangetoxmit=0.500000
bme280tempbasenorm=10.000000
bme280templimitnorm=30.000000
bme280presschangetoxmit=10.000000
bme280pressbasenorm=0.000000
bme280presslimitnorm=100.000000
bme280humidchangetoxmit=2.000000
bme280humidbasenorm=0.000000
bme280humidlimitnorm=100.000000
bme280envnoOfAverages=25
accesspointtimeoutsecs=60
pixelcontrolpin=0
noofxpixels=5
noofypixels=1
pixelconfig=1
pixelbrightness=1.000000
statusledoutputpin=2
statusledoutputactivelow=yes
statusledactive=yes
switchinputpin=5
switchinputgroundpin=-1
switchinputactivelow=yes
switchinputwificonfig=0
messagesactive=no
echoserial=yes
autosavesettings=yes
wifiactive=yes
wifissid1=ZyXEL56E8A7
wifipwd1=*****
wifissid2=myhomewifi
wifipwd2=*****
wifissid3=
wifipwd3=*****
wifissid4=
wifipwd4=*****
wifissid5=
wifipwd5=*****
mqttdevicename=CLB-d1eea3
mqttactive=yes
mqtthost=mqtt.connectedhumber.org
mqttport=1883
mqttsecure=no
mqttuser=littleboxes
mqttpwd=*****
mqttpre=lb
mqttpub=data
mqttsub=command
mqttreport=report
mqttsecsperupdate=360
mqttsecsperretry=10
controlleractive=yes
servooutputpin=12
servoinitialangle=0.000000
servoactive=no
friendlyName=
max7219Messagesactive=no
max7219defaultMessage=Hello!
max7219datapin=4
max7219cspin=5
max7219clockpin=15
max7219xdev=4
max7219ydev=1
max7219stickymessageMS=30000
max7219frametimeMS=1000
max7219framedelay=0.100000
max7219brightness=0.500000
printeron=no
printerbaud=19200
printerdatapin=16
hullosactive=no
Hullosprogram=
otaupdateurl=
otaupdateprodkey=
```

These are the default settings for a device. You can change the setting values by assigning values to them.

Appendix 2 Error Codes

The command line has a number of error codes if an invalid command is entered. These are the code meanings.

```
#define WORKED_OK 0
#define INVALID_HEX_DIGIT_IN_VALUE 1
#define INCOMING_HEX_VALUE_TOO_BIG_FOR_BUFFER 2
#define INCOMING_HEX_VALUE_IS_THE_WRONG_LENGTH 3
#define JSON_MESSAGE_COULD_NOT_BE_PARSED -1
#define JSON_MESSAGE_MISSING_COMMAND_NAME -2
#define JSON_MESSAGE_COMMAND_NAME_INVALID -3
#define JSON_MESSAGE_INVALID_DATA_TYPE -4
#define JSON_MESSAGE_INVALID_DATA_VALUE -5
#define INVALID_OR_MISSING_TARGET_IN_RECEIVED_COMMAND -6
#define COMMAND_FOR_DIFFERENT_TARGET -7
#define JSON_MESSAGE_PROCESS_NAME_MISSING -8
#define JSON_MESSAGE_PROCESS_NAME_INVALID -9
#define JSON_MESSAGE_COMMAND_MISSING_COMMAND -10
#define JSON_MESSAGE_COMMAND_COMMAND_NOT_FOUND -11
#define JSON_MESSAGE_COMMAND_ITEM_NOT_FOUND -12
#define JSON_MESSAGE_COMMAND_ITEM_NOT_INT -13
#define JSON_MESSAGE_COMMAND_ITEM_INVALID_TYPE -14
#define JSON_MESSAGE_COMMAND_ITEM_INVALID -15
#define JSON_MESSAGE_INVALID_COLOUR_NAME -16
#define JSON_MESSAGE_SERVO_VALUE_TOO_LOW -17
#define JSON_MESSAGE_SERVO_VALUE_TOO_HIGH -18
#define JSON_MESSAGE_NO_ROOM_TO_STORE_LISTENER -19
#define JSON_MESSAGE_NO_MATCHING_SENSOR_FOR_LISTENER -20
#define JSON_MESSAGE_NO_MATCHING_LISTENER_IN_SELECTED_SENSOR -21
#define JSON_MESSAGE_SENSOR_MISSING_TRIGGER -22
#define JSON_MESSAGE_SENSOR_ITEM_NOT_FOUND -23
#define JSON_MESSAGE_SERVO_NOT_AVAILABLE -24
#define JSON_MESSAGE_SENSOR_TRIGGER_NOT_FOUND -25
#define JSON_MESSAGE_DESTINATION_STRING_TOO_LONG -26
#define JSON_MESSAGE_LISTENER_COULD_NOT_BE_CREATED -27
#define JSON_MESSAGE_INVALID_CONSOLE_COMMAND -28
#define JSON_MESSAGE_INVALID_REGISTRATION_COMMAND -29
#define JSON_MESSAGE_MAX7219_NOT_ENABLED -30
#define JSON_MESSAGE_SETTINGS_NOT_FOUND -31
#define JSON_MESSAGE_MAX7219_INVALID_DEFAULT -32
#define JSON_MESSAGE_PRINTER_NOT_ENABLED -33
#define JSON_MESSAGE_SENSOR_NOT_FOUND_FOR_LISTENER_DELETE -34
#define JSON_MESSAGE_COULD_NOT_CREATE_STORE_FOLDER -35
#define JSON_MESSAGE_FILE_IN_PLACE_OF_STORE_FOLDER -36
#define JSON_MESSAGE_STORE_ID_MISSING_FROM_STORE_COMMAND -37
#define JSON_MESSAGE_STORE_FILENAME_INVALID -38
#define JSON_MESSAGE_STORE_FOLDERNAME_INVALID -39
#define JSON_MESSAGE_STORE_FOLDER_DOES_NOT_EXIST -40
```