



UIFlow Reference

REV 1.6.3

Written By
Adam Bryant

Introduction	2
M5-Cores	3
Core 2	3
M5 Stick	3
M5 Atoms	3
Stick V / Unit V	4
Timer Camera	4
M5Electron Burner	5
Installing UIFlow Firmware.	6
For M5Stack Core, Gray, Go, Fire and, Core 2	7
The Factory Demo (Excluding Core2).	9
To Exit The Factory Demo.	9
Wifi Setup	10
Connecting to UIFlow	12
Events	16
Loop	17
Button Loop	19
Obtain Button Value,	21
Button A+B Press Loop	23
Button Value	25
Timers (Part 1)	27
Timer Callback Loop.	27
Set timer Period	27
Start Timer	29
Stop Timer	30
Virtual User interface Designer and Graphic Functions.	32
The Show/Hide Blocks.	33
The Show Text Blocks	34
Set Screen Rotate Mode	34
Set Color	35
Set Screen Background Colour	35
Set Screen Brightness	36
Set Colour R G B.	36
Set Title Background Colour	37

Set Width and Height	37
Set X and Y	38
Set Circle Radius.	39
Images.	40
.BMP Files.	41
.jpg files.	44
Displaying Images on screen.	47
Internal Hardware	52
Speaker,	53
Speaker.Beep Frequency,	53
Speaker Volume,	54
Play Tone.	55
RGB,	57
Set RGB Bar Colour,	57
Set RGB Bar Colour R,G,B,	58
Set (Side) RGB Bar Colour,	59
Set (Side) RGB Bar Colour R,G,B,	61
Set (individual) RGB Colour,	63
Set (individual) RGB Bar Colour R,G,B,	65
Set RGB Brightness,	67
IMU.	68
Get X,	68
Get Y,	69
Get X ACC,	70
Get Y ACC,	71
Get Z ACC,	72
Get X Gyro,	73
Get Y Gyro,	74
Get Z Gyro,	75
Power (StickC)	76
Get Charge State	76
Get Battery Voltage	77
Get Battery Current	78
Get Vin Voltage	79

Get Vin Current	80
Get VBus Voltage	81
Get VBus Current	82
Get AXP192 Temperature	83
Power Off	84
Set Battery Charge Current	86
Set LCD Voltage	86
Is Charging	88
Is Charge Full	89
Set Charge	90
Get Battery Level	92
RTC - The Real Time Clock	93
Time and Date configuration block.	93
Get Year,	95
Get Month,	96
Get Day,	97
Get Hour,	98
Get Minute,	99
Get Second,	100
Get Local Time.	101
LED	102
LED On	102
LED Off	102
HATS	105
ENV Hat	106
Get Pressure,	106
Get Temperature,	106
Get Humidity,	106
PIR Hat,	108
Get hat_pir Status,	108
Speaker Hat	109
Hat_spk0 Play Tone	109
Hat_spk0 Speaker.beep	110
Hat_spk0 Speaker Volume	110

NCIR HAT **111**

NCIR Read 111

DAC **112**

hat_dac0 Output Voltage 112

hat_dac0 Output Voltage with Raw Data. 114

ADC 115

Hat Servo **116**

Rotate Servo to Degree. 116

Write Milliseconds 118

Servo8 Hat. **120**

Set (num) Servo Rotate 120

Set Servo Millisecond 122

Set RGB Bar Colour 124

Set RGB Bar Colour (R,G,B,) 124

Puppy C **127**

Set Servo position 127

Set servo position 0 legs 0, 1, 2, 3 129

Joystick **131**

Get hat_Joystick0 X 131

Get hat_Joystick0 Y 131

Get hat_Joystick0 is pressed. 131

Get hat_Joystick0 Reverse X 131

Get hat_Joystick0 Reverse Y 131

BugC **132****Finger** **133****BeetleC** **134****YUN** **135****JoyC** **136****RoverC** **137****UIFlow Blocks.** **138****RoverC** **138**

Set Wheel Pulse 138

Set Wheel Pulse (four wheels) 138

Set RoverC Speed	139
Movement	139
Turn Left.	139
Turn Right	141
Move Forwards	142
Move Backwards	143
Rotate Clockwise	144
Rotate Anti-Clockwise	145
Slide Left	146
Slide Right	147
Slide Forward Left	148
Slide Forward Right	149
Slide Backwards Left	150
Slide Backwards Right	151
TOF	152
Thermal Camera	153
Card KB	154
Units	155
AC Socket	156
Set AC Socket Value	156
Environmental,	158
Get Pressure,	158
Get Temperature,	159
Get Humidity,	160
Angle,	161
Get Angle,	161
Get PIR Status,	162
RGB LED,	163
Set Index RGB Colour,	164
Set RGB Range to colour,	165
Set WS2812b Range to RGB colour,	167
Set all WS2812b Colour,	169
Set WS2812b Brightness,	170

Set WS2812b Hex Colour,	172
Set WS2812b Hex (Variable) Colour,	175
<u>Joystick,</u>	178
<u>Light,</u>	179
Get Light Analogue Value,	179
Get Light Digital Value,	179
<u>Earth,</u>	180
<u>Makey,</u>	181
<u>Servo,</u>	182
<u>Weight,</u>	183
<u>Button,</u>	184
Button Loop,	184
<u>Duel Button,</u>	185
Duel_Button Loop,	185
<u>RGB,</u>	186
<u>Relay,</u>	187
<u>ADC,</u>	188
ADC Read,	188
<u>Colour,</u>	189
<u>IR,</u>	190
Get IR State,	190
<u>Thermal,</u>	191
TOF,	193
Get Distance,	193
EXT I/O,	194
RFID,	195
<u>PAHub</u>	196
Set position state	196
Set position	196
Set port value	196
<u>Heart Unit</u>	198
<u>Modules</u>	199

<u>Stepper Motor,</u>	200
<u>Servo,</u>	201
<u>Bala,</u>	202
<u>Bala Motor,</u>	203
<u>Lego +,</u>	204
<u>Lidarbot</u>	205
Lidarbot set front with neopixel colour,	205
Lidar bot Set Individual LED Colour,	206
Lidarbot Move,	206
Lidarbot Set Speed,	206
Lidarbot Set Servo,	206
Lidarbot Draw Map,	207
<u>LoRaWan,</u>	209
<u>PM2.5</u>	210
<u>Cellular,</u>	211
<u>Base X</u>	214
<u>Plus</u>	214
<u>GoPlus</u>	214
<u>GPS</u>	214
<u>Faces Modules</u>	215
<u>Logic Blocks</u>	223
<u>Variables,</u>	224
Create Variable,	224
Set Variable,	224
Change Variable,	224
Variable,	224
<u>Maths,</u>	226
Constant,	226
Common Equation,	226
Special Vaules,	226
Remainder of,	226
Value is even,	226

Sum of list	227
Random Fraction	227
Random Integer,	228
Round,	228
Square Root,	228
Sin,	228
Convert to int,	228
Convert to Float,	228
Reserve Decimal Fraction.	228
Get Bit	229
Set Bit	229
Clear Bit,	229
Reverse Bit,	229
Int From Bytes,	229
Loops	229
Repeat,	229
Repeat while Condition,	229
For Each,	229
Count with,	230
Break out,	230
Logic,	231
If - Do,	231
If - Else - Do,	231
Try/Except	231
Condition True,	231
Condition Equals Condition,	231
Not,	232
Null,	232
Condition and Condition,	232
Test - True/False	232
Switch	232
Graphic,	233
Lcd.clear	233
Change Background Image.	237

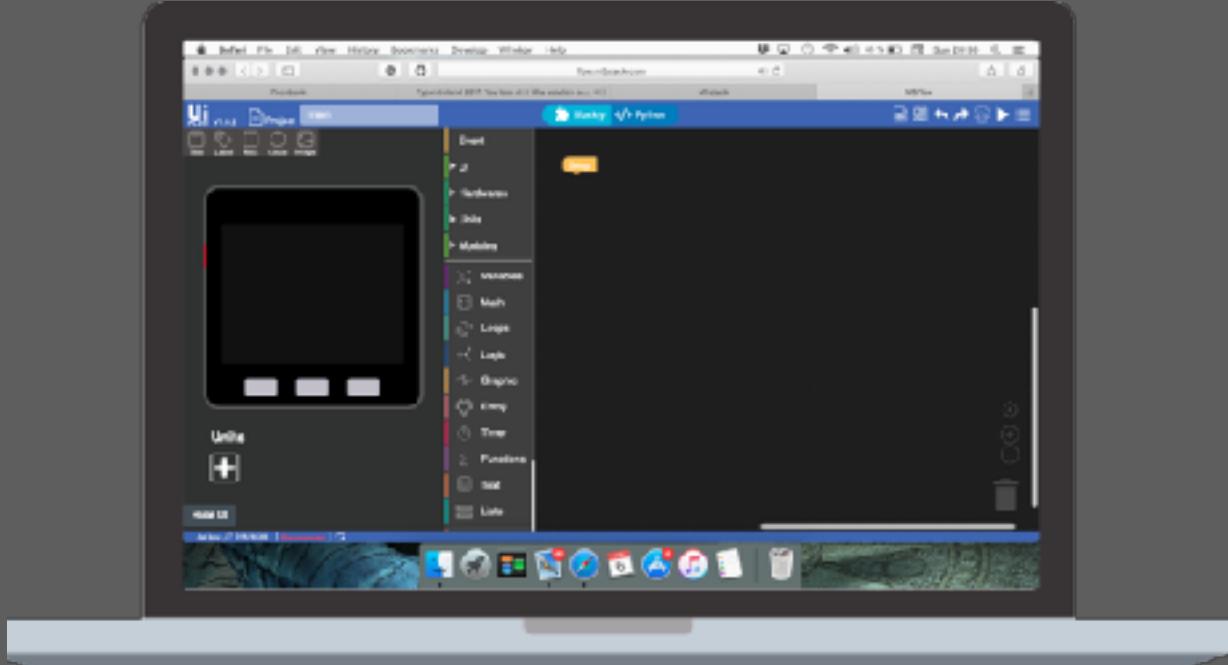
Timers (Part 2),	238
Wait Seconds,	238
Wait Milliseconds,	238
Get Ticks ms	238
Create List With,	245
Advanced Blocks	251
SDCard	252
Open SDCard File	252
File Read all	254
File Read Bytes	255
File Read Line	256
File Write	257
File Seek	258
Easy I/O,	259
Analog Read Pin,	259
UART,	264
Set Uart,	264
Read Uart,	264
I2C,	266
Set I2C Port,	266
Set I2c SDA, SCL,	266
I2C Scan,	266
I2C Available Address in List,	266
I2C Read Reg One Byte,	266
I2C Read Reg One Short,	266
I2C Read Reg One Byte	267
I2C Write Reg One Byte,	267
I2C Read Byte,	267
I2C Write Reg One Short,	267
Execute,	268
Execute Code,	268
Code Mode.	268
Documentation Mode.	268

<u>Network,</u>	270
<u>ESP NOW</u>	271
Get mac Address	271
Add Peer	272
Set PMK	274
Broadcast Data	275
Send Message ID with Data	276
After Send Message Flag	277
Receive MAC Address Data	278
<u>Remote Control</u>	281
Remote,	282
<u>Custom Blocks.</u>	284
<u>Micropython API's</u>	285
<u>Advanced</u>	286
Easy IO	286
Pin	286
Pwm	287
I2C	287
ADC	288
DAC	288
MQTT	289
ESP-NOW	289
ESP-NOW (continued)	290
UART	290
WiFi	291
P2P	291
<u>Display Module</u>	292
Class TFT	292
Colors	292
Drawing	292
Fonts	292
Touch panel	293
Methods	293

Event API's	300
Button	300
Time	301
Faces API's	302
Calculator	302
Encode	302
Finger	303
Gameboy	304
Joystick	304
Keyboard	305
RFID	305
Graphics/LCD API's	306
LCD	306
Internal Hardware API's	310
MPU6050	310
SD	310
Speaker	310
RGB Bar	311
M5UI	312
Colors	312
Title	312
Circle	313
Img	313
Rect	314
TextBox	315
Remote API's	316
Remote	316
Hat API's	317
ENV HAT	317
PIR HAT	317
Speaker	317
NCIR	317
DAC	318
ADC	318

Servo	318
Servo8	318
PuppyC	318
Joystick	319
BugC	319
Finger	320
BeetleC	320
Yun	320
JoyC	321
RoverC	321
TOF	321
Thermal	322
CardKBC	322
units	323
ENV	323
PIR	323
Neopixel	323
Joystick	324
Makey	324
Servo	324
Weight	325
Tracker	325
Button	326
Dual Button	326
RGB	327
Relay	327
ADC	327
DAC	328
Ncir	328
IR 328	328
Extend IO	329
Angle	329
Light	329
Earth	330

Tof	330
Color	330
RFID	331
Finger	331
CardKB	332
Heart	332
<u>Appendix 1</u>	<u>333</u>
<u>Data-sheet Catalogue.</u>	<u>333</u>
<u>Appendix 2</u>	<u>336</u>
<u>I2C Address.</u>	<u>336</u>
<u>Appendix 3</u>	<u>339</u>
<u>Table of symbols available on the CardKB and the CardKB Hat.</u>	<u>339</u>
<u>Index</u>	<u>342</u>



Introduction

UIFlow is the Integrated Development Environment (IDE) created by M5Stack and is the third incarnation of the programming environment.

In various documents references are made to M5Cloud and Moments, these are the two previous versions and are no longer available.

UIFlow contains two environments. Blocky, the default environment is a basic programming aimed at the young and beginners to programming. Micropython, the second environment is a text based environment that is at the core of all the firmware and functions of blocky. Once a programmer has grown beyond blocky, they can use the Micropython environment to dig deeper in to the functions and develop new blocks for Blocky or program the M5Stacks and M5Sticks at a lower level.

Blocks in UIFlow can be separated into three groups, actions, loops and values.

Actions are the blocks that contain command to preform tasks.

Loops are used for code that needs to be repeated.

Values provide numbers that can vary or be defined by the programmer.

I hope that this section will explain what the various blocks available in the UIFlow environment are. In another section I will show examples on how to use the various blocks to make things work.

M5-Cores



Core 2



M5 Stick



M5 Atoms



Stick V / Unit V



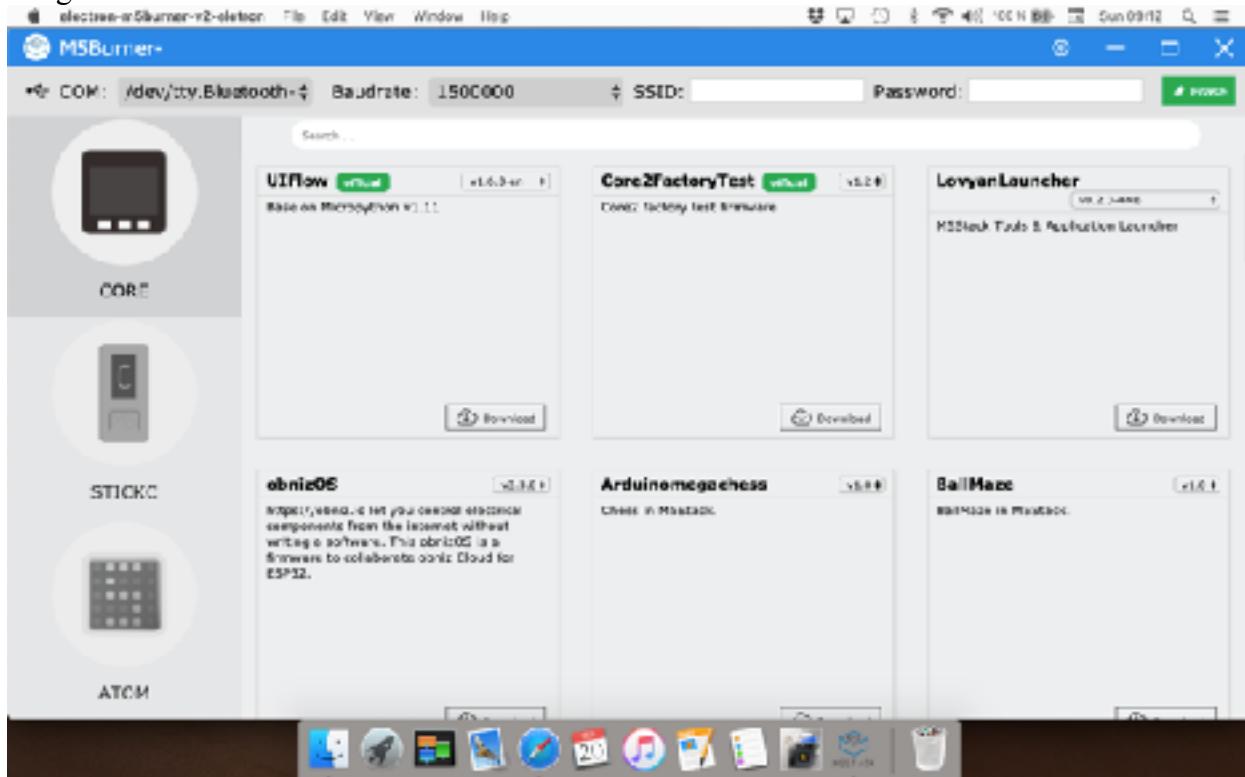
Timer Camera

Currently the Thermal Camera does not have support for UIFlow firmware



M5Electron Burner

The current firmware tool is called the Electron Burner. This version has a new cleaner, easier to navigate layout compared to the previous version and a screen shot can be seen in the following images.



With the new program comes some new setting options. On the left of the new bar we have the familiar Com Port and Baud rate options but now we also have options for pre setting The SSID and password that M5Stack devices will use to connect to the target wifi network.



One important note about WIFI SSID and passwords: SSID's can not contain a space or special characters in the name. If there is a space, or special character then for some reason M5Stack devices will fail to connect to wifi networks.

Pre setting the WIFI SSID and password can save us some setup time but it is not the only way to connect a device to a network. The additional wifi connection methods will be explored after I demonstrate how to install firmware.

Installing UIFlow Firmware.

With the exception of the SSID and Password box's, installation of UIFlow firmware hasn't changed much compared to the previous version.

Before we connect the M5Stack, Stick or Atom to the computer, we first need to download a USB driver that will allow our host computer to talk to our device.

To find the driver we need to visit <https://m5stack.com/pages/download> and in the software menu on the left, click on download and select the operating system that you are using.



Once downloaded, run the install process. Once the install has finished, shut down the computer, wait a few minutes and then restart the computer. There is an issue with OSX and Windows in that if the shut down is not completed, a built in faulty drive will be loaded resulting in M5Stack devices failing to connect over USB.

Important Note: If after this step, if your computer still does not connect to an M5Stack product, try a good quality USB C Data sync cable as the cheap cables suffer with broken conductors and charge cables sometimes don't have the data wires.

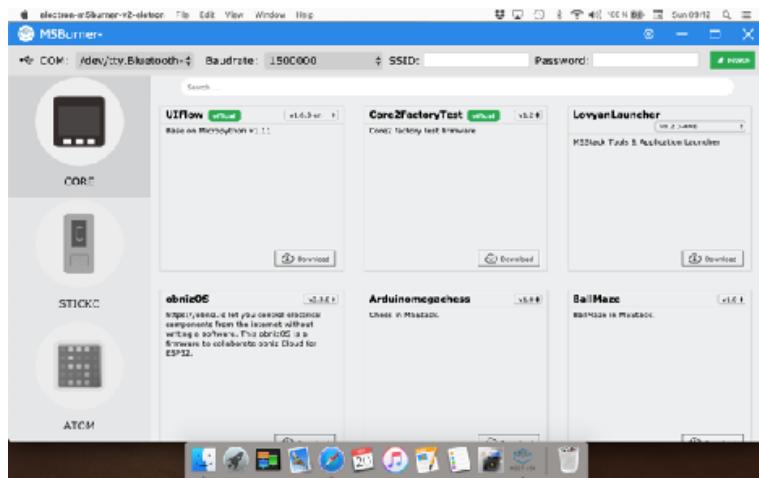
In the left panel we have images of the M5Stack, Stick and Atom, StickV/UnitV and Timer Camera. Click on the image that matches the device you are using and the main panel will now show you the list of available pre built firmwares. Some firmwares will have multiple revisions available (like UIFlow) which can be chosen by click the drop down list next to the firmware name.

Please note that when selecting the firmware revision, there may also be several versions of that available. In the case of UIFlow on the Core there is a version for the Core, Fire, and Core 2 due to the difference in hardware configurations.

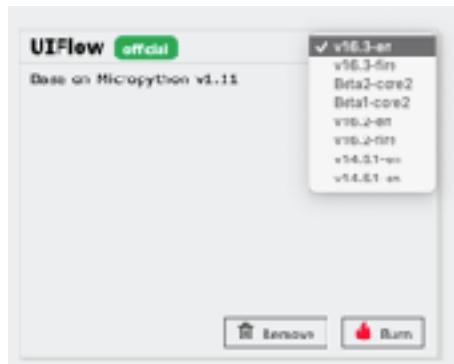
First we select the Com port a M5Stack is connected to. On a Windows computer, the device will be shown as "Com" however, with OSX, BSD and *nix based computers, devices will often appear as /dev/tty *** with a different string for different devices.

For M5Stack Core, Gray, Go, Fire and, Core 2

Click on the Core Icon



Click on UIFlow and then in the dropdown box chose the revision number that ends in “en”.



For the Fire chose the revision ending in “Fire” and for the Core 2 chose the revision ending in “Core2” and then click “Download”.

Next plug the M5Stack into a USB port and select the Com port.



Leave the Baud rate as is, type in your wifi SSID and Password (not shown for privacy). Next press the “Erase” Button and the following will appear on screen.



If everything worked, the window will show the “Erase Successfully”. Press the Close button and we can then move on to install the firmware.



To install the firmware all we need to do is click on the “Burn” button, a new screen will appear and we can wait until you see the message in the following screenshot.
Your device should now restart and show the “Hello” screen of the built in demo.

The Factory Demo (Excluding Core2).

The factor demo has been designed to show off the functions and sensors included with the M5Go Packs and so will have functions that wont work with the Core (Black).

To Exit The Factory Demo.

To exit out of the factory demo press the reset/on off button to reset the core and as soon as the screen show up (shown Below) Press the right hand button with “Setup” above it to go into the settings.

In the setting menu the three white button have the following unctions.

Left Button = Move up.

Centre Button = Select/Enter.

Right Button = Move down.

Press the middle button to open the “Switch Mode” menu and you will get the following options.

Internet Mode = For using the online UIFlow.

USB Mode = For using Offline mode also known as the UIFlow Program.

APP Mode = This mode will auto load the last program downloaded to the memory of the core.

Use the right button to move down to “Internet Mode” and press the middle button. The circle next to it should now be filled in and we can now restart the Core by moving down and pressing the middle button once “Reboot >>” is highlighted.

Wifi Setup

The first time a Core starts up after being set to WIFI Mode, you will be presented with the following screen.



Click on the icon or program you see to select wifi networks and choose the SSID that matched the SSID shown on the screen of the core. (In this case M5-E218)



Open a web browser (Explorer, Safari, Firefox or Chrome) and type in 192.168.4. This should load a configuration web page stored on the core which will allow you to set the network SSID and password of the network you wish to have your M5Stack Core connected to.

Select the SSID you want to use (this normally shows the closest network), Insert your password and press configure.

M5FLOW

WiFi Setup

SSID: SKYWF08D Other

Password:

Configure

If you entered a valid SSID and password and the core connected, you will be presented with the following in the web browser before the Core restarts and the computer automatically reconnected to the wifi.

^_^ WiFi connection success

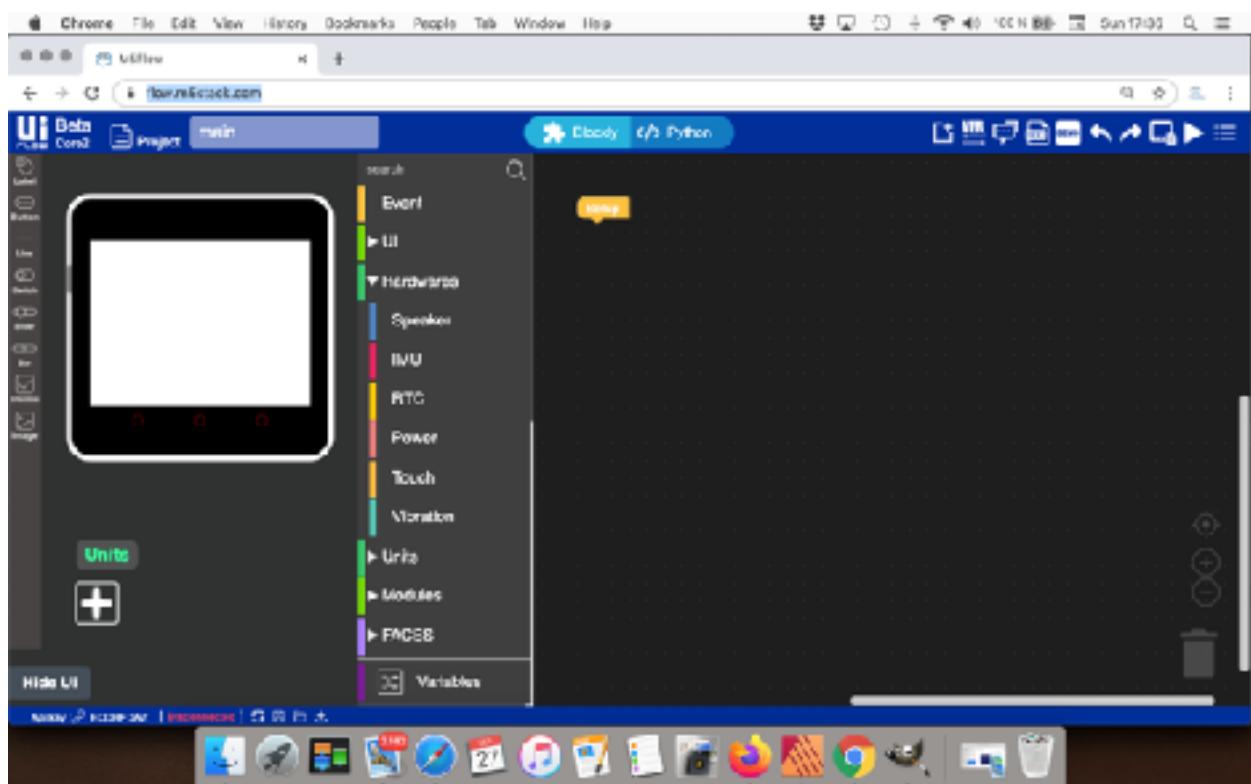
[Reset device now .](#)

Connecting to UIFlow

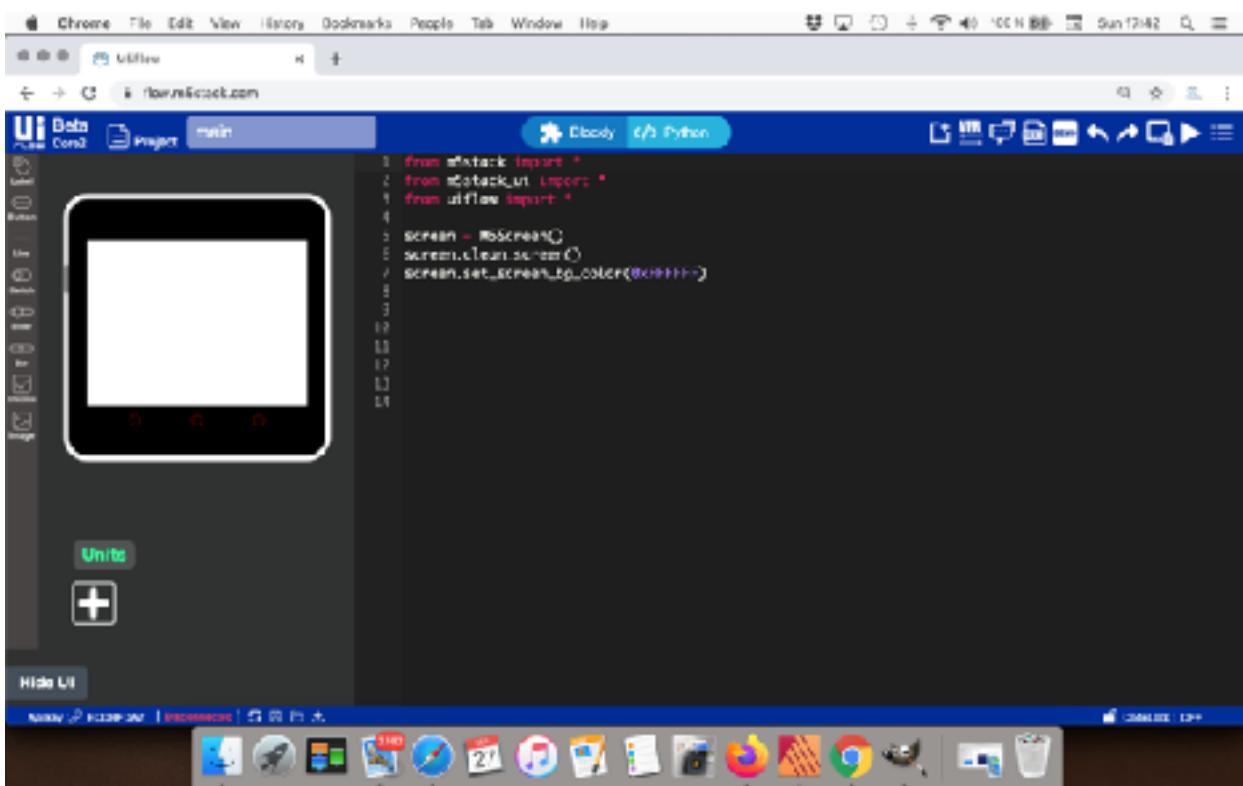
Now that we have our M5Stack Core connected to the internet we need to connect the device to the online UIFlow Server. To link our core to the server, we need the API number that is now shown on the screen after each restart.



Open a new page in the internet browser and goto <https://flow.m5stack.com/>



This is UIFlows main screen. Across the top is the title bar which shows the current revision of the UIFlow web server. Next to that is a box that we can use to set our project name. We we want to save the program to a computer the file name will use the project name we specify here.



In the middle of the title bar is the two IDE buttons. Blocky is the default IDE shown above, Python takes us to the Code view where we can directly edit and code in Micropython.



On the right of the title bar we have a set of Icons. These icons and their functions are as follows.

- New File - This will clear the current open program ready to start a new program.
- VER - This will allow you to change between the latest stable version and the latest test version.
- Forum - Takes you to the M5Stack Forum.
- DOC - Takes you to the UIFlow online Documentation.
- Demo - Opens a dropdown menu containing available preinstalled demos.
- Undo - Undo a code change.
- Redo - redo a code change.
- Manager - Allows you to add or remove libraries and files to the onboard storage.
- Run - This downloads the program to the the ram to test the program when when you restart the program is lost as it is not premaritally saved.
- Main Menu - Contains UIFlow settings and options to save the code to a computer or to a device perinatally.



The Settings is where we go to configure UIFlow to connect to and M5Stack, stick or core. The API Key box is where you type in the number shown of the screen of the M5Stack or Stick. Language is to set which language you need UIFlow to be in if English is not your language. Next you click on the icon that represents the device you are connecting to UIFlow. Once these are set you can then click OK to connect UIFlow to your device. Sometimes UIFlow or the device will not connect to the UIFlow server, this is often down to a bad internet connection or the server being overloaded by users trying to connect online.

Event Blocks

The event folder contains the main basic functions that will be used by nearly all of our code in UIFlow.
This folder contains the basic loop, buttons and timer functions.

Events

Setup

The set-up block is required by all programs as it works in the background to import the library required for our programs to run. This block is the equivalent of Arduino's Void Setup function and contains functions that you only want to run once at the beginning of the program. In the Blockly windows we can't see much but if we switch to the </> Python window we see the following Micropython code.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)
```

We can see in this code snippet that this block imports the three core libraries required by all programs and sets the initial screen colour to an almost black colour. By changing the value highlighted in red, we can change the initial screen colour. For more information on setting colours, check out the colour blocks.

Event Blocks

Loop

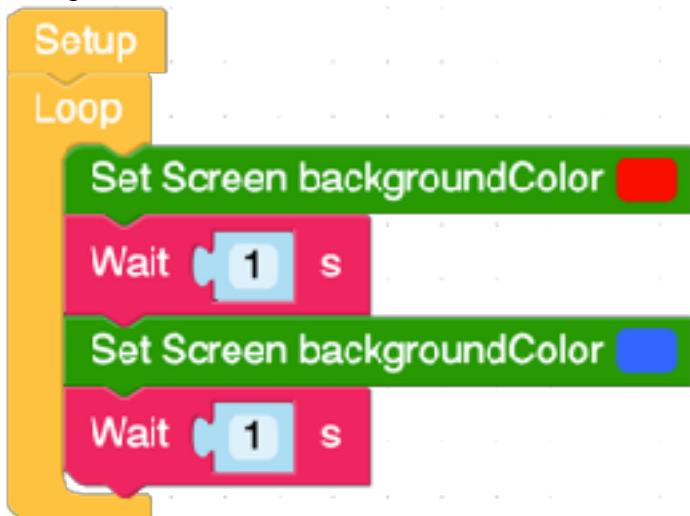


After this we have the main program loop. Inside of this block we place the main program that we want to continuously repeat.

The Micropython code for this block is

while True:

Example



In the following example I am using a loop to continuously step through the Set screen Background Colour block. If this is run without the wait blocks then the screen will look purple because the screen is changing faster than the human eye can see. To be able to see the colour change I have added a wait block to make the program wait one second before moving between the blocks.

Event Blocks

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

while True:
    setScreenColor(0xff0000)
    wait(1)
    setScreenColor(0x3366ff)
    wait(1)
    wait_ms(2)
```

For more information on the wait block, check out the timer section of the book.

Event Blocks

Button Loop



As well as the main loop, we have the button loop that continuously watches the three main buttons on the front of the M5Stacks and then runs the code inside it when it detects a triggering even.

The four events the button loop waits for are as follows.

Was Pressed - This is the normal mode of operation and waits until a single short press of the button is detected before running the code inside it.

Was Released - This waits until the button is pressed and released before running the code placed inside it.

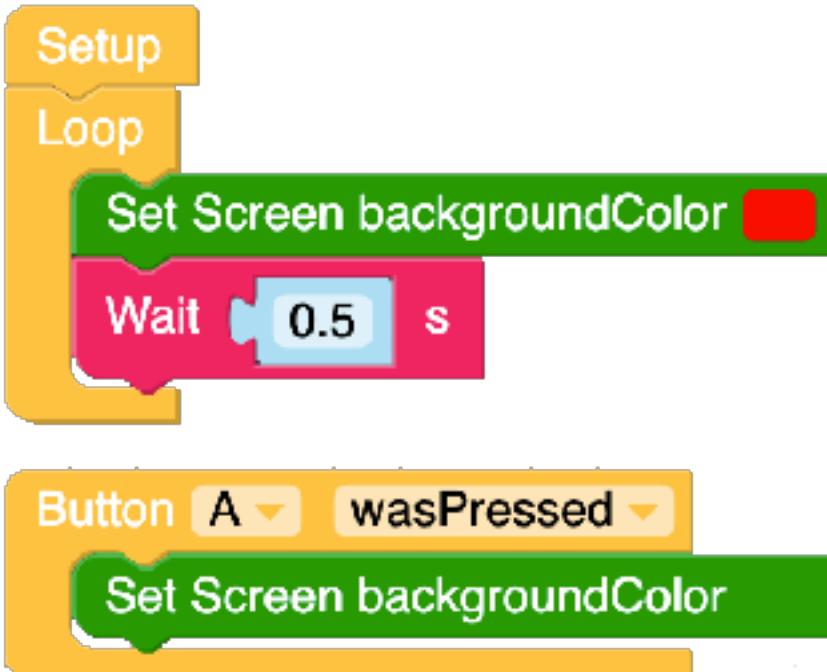
Long Press - Like the "Was Pressed" block, it waits until the button is pressed however, this block on activates if the button is pressed and held down for a few seconds.

Was Double Pressed - This function checks to see if the put-on was pressed and released twice within a few milliseconds before running the code inside it.

The MicroPython code for the Button Loop is as follows.

```
def buttonA_wasPressed():
    # global params
    pass
btnA.wasPressed(buttonA_wasPressed)
```

Example



Event Blocks

In this example I have two loops. The main loop set the screen to red and the button loop changes the screen to green when the button is pressed. The main loop is required to reset the screen colour after the button has been released.

The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

def buttonA_wasPressed():
    # global params
    setScreenColor(0x009900)
    pass
btnA.wasPressed(buttonA_wasPressed)

while True:
    setScreenColor(0xff0000)
    wait(0.5)
    wait_ms(2)
```

Event Blocks

Obtain Button Value,



The obtain button block works in a similar mode to the Button loop but instead of running code placed inside it, it is a value block that returns true or false when one of the button events have been triggered.

The four events the button value block waits for are as follows.

Was Pressed - This is the normal mode of operation and waits until a single short press of the button is detected before running the code inside it.

Was Released - This waits until the button is pressed and released before running the code placed inside it.

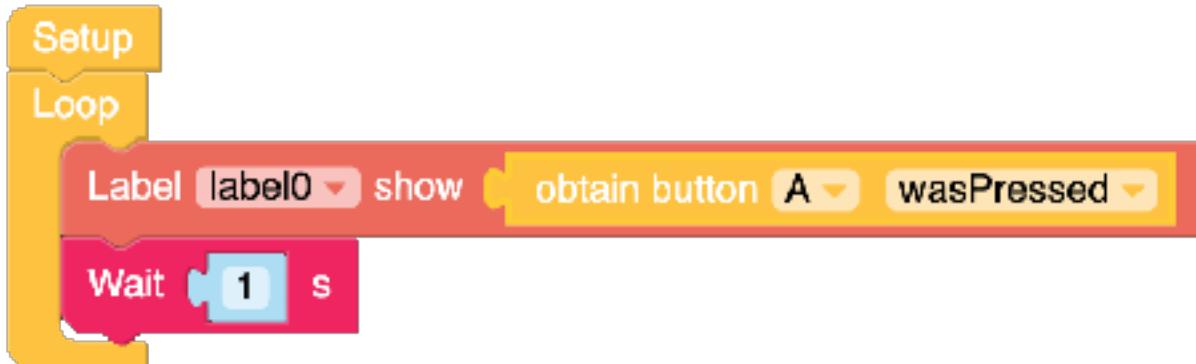
Long Press - Like the "Was Pressed" block, it waits until the button is pressed however, this block activates if the button is pressed and held down for a few seconds.

Was Double Pressed - This function checks to see if the button was pressed and released twice within a few milliseconds before running the code inside it.

The MicroPython code for the Button Loop is as follows.

```
btnA.wasPressed()
```

Example



Event Blocks

In this example I am using a Label block to show the status of the buttons.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(95, 100, "Text", lcd.FONT_Default,0xFFFFFFF, rotate=0)

while True:
    label0.setText(str(btnA.wasPressed()))
    wait(1)
    wait_ms(2)
```

Button A+B Press Loop

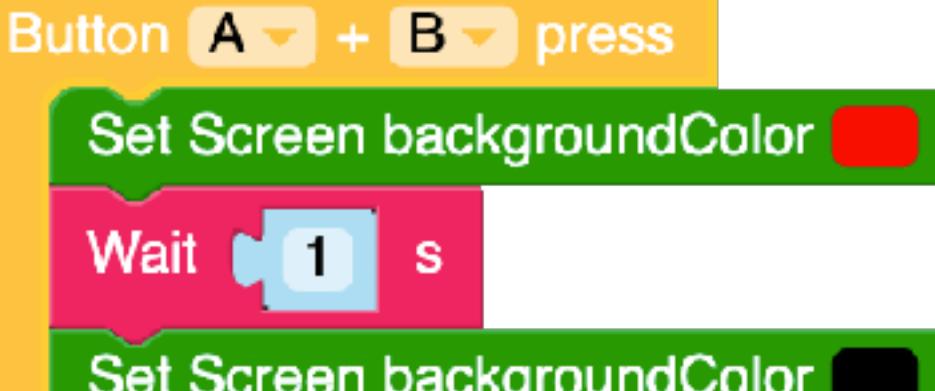
Button A ▾ + B ▾ press

Button A+B press block expands on the function of the button loop by waiting until it detects that any two buttons on the front of the m5stack that have been selected in the drop down boxes have been pressed at the same time.

The Micropython code for the Button Loop is as follows.

```
def multiBtnCb_AB():
    # global params
    pass
btn.multiBtnCb(btnA,btnB,multiBtnCb_AB)
```

Example



In the following example the screen will turn red when both A and B is pressed and then if the buttons have been released for more than a second, will turn black.

The Micropython code for this example is as follows.

Event Blocks

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

def multiBtnCb_AB():
    # global params
    setScreenColor(0xff0000)
    wait(1)
    setScreenColor(0x000000)
    pass
btn.multiBtnCb(btnA,btnB,multiBtnCb_AB)
```

Event Blocks

Button Value



Similar to the Obtain Button function, this block also monitors buttons A, B, and C on the front panel of the M5Stacks but is simplified to only having options for Pressed and Released.

Example 1,



In the following example a label with show false until button A is pressed, the label will show true until A button is released.

The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(95, 100, "Text", lcd.FONT_Default,0xFFFFFFF, rotate=0)

while True:
    label0.setText(str(btnA.isPressed()))
    wait_ms(2)
```

Event Blocks

Example 2,



In this following example a label with show false while button A is pressed, the label will show true when A button is released.

The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(95, 100, "Text", lcd.FONT_Default,0xFFFFF, rotate=0)

while True:
    label0.setText(str(btnA.isReleased()))
    wait_ms(2)
```

Event Blocks

Timers (Part 1)

Timer Callback Loop.

A Scratch-style block labeled "timer callback". Inside the block, the word "timer1" is enclosed in a light orange rounded rectangle, indicating it's a variable that can be renamed.

The timer callback look creates a timer that can contain actions that are set to run under certain time critical events. Click on the light box with "timer1" in it to change the name of the timer. The Micropython code for the loop is as follows:

```
@timerSch.event('timer1')
def ttimer1():
    # global params
    pass
```

Set timer Period

A Scratch-style block labeled "Set Example period". The "period" input has a value of "100". The "ms mode" dropdown is set to "PERIODIC".

A Scratch-style block labeled "Set Example period". The "period" input has a value of "100". The "ms mode" dropdown is set to "ONE_SHOT".

Defines the time the timer will run for. Periodic mode set the timer to continuously run for the defined period whereas "One Shot" will set it run once. The time value can be set with a maths block or a variable block. The operating mode can be changed by clicking the down arrow on the right of the block.

The Micropython code for set timer Period block in periodic mode is

```
timerSch.setTimer('Example', 100, 0x00)
```

and for "One Shot" mode,

```
timerSch.setTimer('Example', 100, 0x01)
```

It is worth noting that the only difference in the code is the 0x00 used to set "Periodic" mode and 0x01 to set "One Shot" mode in Micropython.

Event Blocks

Examples

The following examples show how to set the different modes.

Setup

timer callback Example



The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

@timerSch.event('Example')
def tExample():
    # global params
    timerSch.setTimer('Example', 100, 0x00)
    pass
```

Setup

timer callback Example



The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

@timerSch.event('Example')
def tExample():
    # global params
    timerSch.setTimer('Example', 100, 0x01)
    pass
```

Event Blocks

Start Timer

Start Example period 100 ms mode PERIODIC

Start Example period 100 ms mode ONE_SHOT

Starts the timer for a defined period of time. The time value can be set with a maths block or a variable block. The operating mode can be changed by clicking the down arrow on the right of the block.

The MicroPython code for the stop block is:

```
timerSch.run('Example', 100, 0x00)
```

For "Periodic" mode and

```
timerSch.run('Example', 100, 0x01)
```

For "One Shot" Mode.

Example.

Event Blocks

Stop Timer

Stop Example ▾

Stops the timer selected in the drop down box.

The Micropython code for the stop block is:

```
timerSch.stop('Example')
```

Example.

The following example (as mentioned earlier) has been created to show how the various time blocks react with each other but also rely on each other to work.

The Micropython code for this is quite long and is more advanced than previous demos'

Virtual User interface Designer and Graphic Functions.

The M5Stack and M5Stick models have a selection of text and graphic functions defined for us. Some of these functions share the same code blocks and some have their own unique blocks. In this chapter I will explore the blocks available and teach you how to use them.

User Interface Elements

Virtual User interface Designer and Graphic Functions.

The M5Stack and M5Stick core models have a selection of text and graphic functions defined for us. Some of these functions share the same code blocks and some have their own unique blocks. In this chapter I will explore the blocks available and teach you how to use them.

Functions are not the only things to share code. some of the hardware like the WS2812bs and RGB LEDs also share some of the code blocks.

Code Block	Title	Label	Rectangle	Circle	Image
Show Text	Yes	Yes	Yes	Yes	Yes
Show	Yes	Yes	Yes	Yes	Yes
Hide	Yes	Yes	Yes	Yes	Yes
Set Colour	Yes	Yes	Yes	Yes	Yes
Set Colour RGB	Yes	Yes	Yes	Yes	Yes
Set Background Colour	Yes	Yes	Yes	Yes	Yes
Set Background Colour RGB	Yes	Yes	Yes	Yes	Yes
Set Width and Height	No	No	Yes	No	Yes
Set Width	No	No	Yes	No	Yes
Set Height	No	No	Yes	No	Yes
Set X and Y Position	No	No	Yes	Yes	Yes
Set X Position	No	No	Yes	Yes	Yes
Set Y Position	No	No	Yes	Yes	Yes
Set Radius	No	No	No	Yes	No
Available to M5 Stack	Yes	Yes	Yes	Yes	Yes
Available to M5 Stick	No	Yes	Yes	No	No

The table above shows some of the User Interface (UI) functions and which code blocks they share.

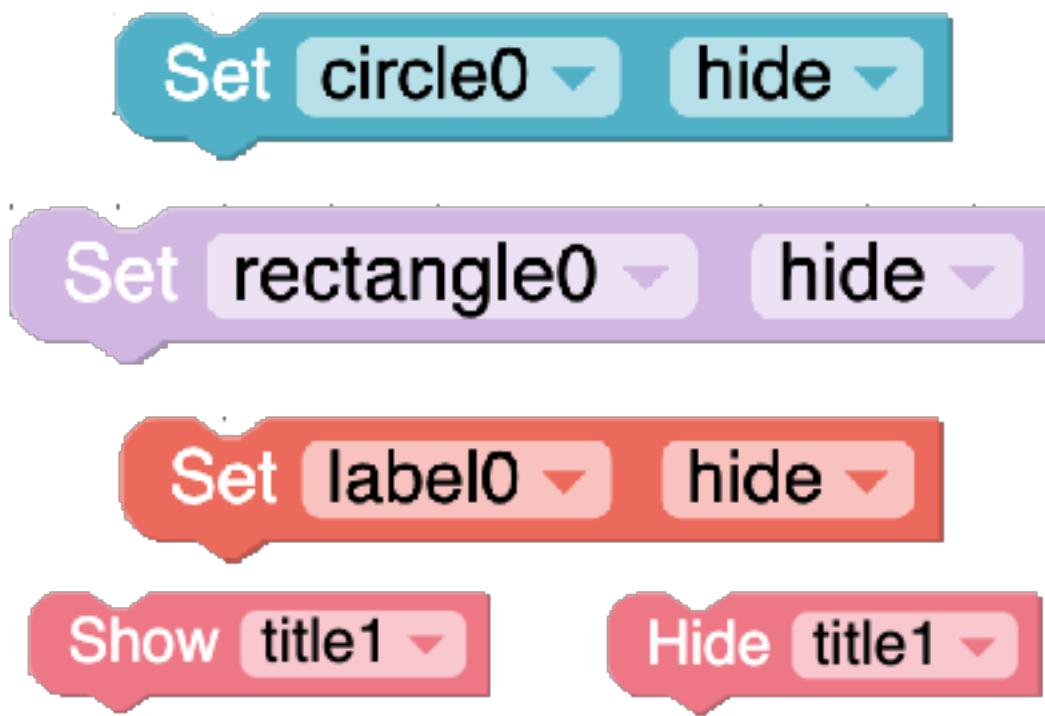
When used in conjunction with other functions we can create functions that can be as simple as displaying the value of a sensor or as complicated as you can imagine.

There are two Show blocks, the first will display the UI element defined on UI builder while the second takes a value from one off the sensors and changes the text to show that value.

Hide, hides the onscreen text.

User Interface Elements

The Show/Hide Blocks.



The Show/Hide Micropython code for the blocks

As you can see in the images above the Show/Hide block is only available to the Label,

```
circle0.hide()  
circle0.show()  
rectangle0.hide()  
rectangle0.show()  
label0.hide()  
label0.show()  
title0.show()  
title0.hide()
```

Rectangle and Circle UI elements. If you have more than one element on the screen you can use the block to control the individual elements by clicking on the arrow beside the element number. The second block is where you choose Hide or Show for the elements visibility. You can use this block to hide elements until certain event or conditions to happen and then set the individual elements to true in the condition or event function.

The Title UI element is slightly different in that its show and hide blocks are separate.

User Interface Elements

The Show Text Blocks



These two blocks are used to show or print a string of text onto the LCD screen. You can see that it has a puzzle-shaped space in it that allows you to use the title and label blocks for showing values returned from sensors.

The MicroPython code for these blocks are

```
title0 = M5Title(title="Title", x=3, fgcolor=0xFFFFF, bgcolor=0x0000FF)
label0 = M5TextBox(26, 85, "Text", lcd.FONT_Default, 0xFFFFF, rotate=0)
label0.setText('Hello M5')
```

Set Screen Rotate Mode



Allows you to rotate the screen using one of the four preset modes shown below.

Mode 0 turns the screen 90 degrees clockwise,

Mode 1 is normal direction,

Mode 2 is 90 degrees anticlockwise,

Mode 3 is 180 degrees clockwise.

The MicroPython code for these blocks are

```
lcd.setRotation(0)
```

User Interface Elements

Set Color

Set title0 color

Set label0 color

Set rectangle0 color

Set circle0 color

The Set Colour block allows you to set each elements colour using the colour picker. Set Screen backgroundColor set the screen or "canvas" colour independent of the other U.I Elements.

The Micropython code for these blocks are:

```
title0.setFgColor(0xff0000)  
label0.setColor(0xff0000)  
rectangle0.setBgColor(0xff0000)  
circle0.setBgColor(0xff0000)
```

Set Screen Background Colour

Set Screen backgroundColor

Sets the background colour of the screen.

The Micropython code for this block is

```
setScreenColor(0x111111)
```

User Interface Elements

Set screen brightness

30

Set Screen Brightness

Sets the screens brightness using a value block.

The Micropython code for this blocks is

```
lcd.setBrightness(30)
```

Set Colour R G B.

Set title0 color by R 0 G 0 B 0

Set label0 color by R 0 G 0 B 0

Set rectangle0 color by R 0 G 0 B 0

Set circle0 color by R 0 G 0 B 0

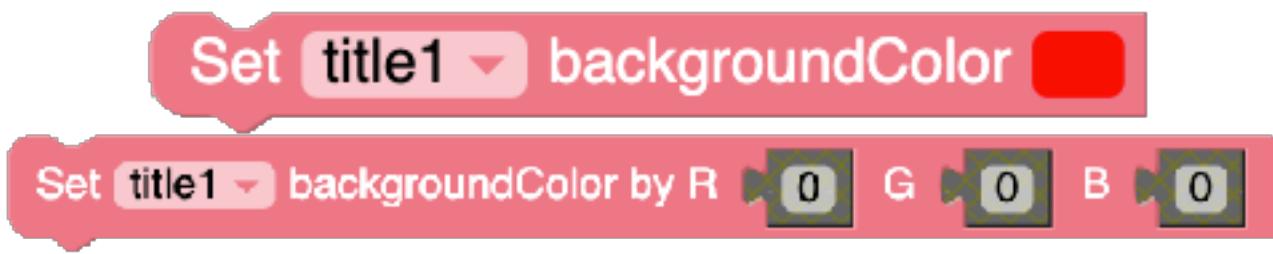
This Set Colour block allows you to set each elements colour by specifying individual value for each of the separate colour channels.

The Micropython code for these blocks are:

```
rectangle0.setBgColor(0x000000)  
circle0.setBgColor(0x000000)  
label0.setColor(0x000000)  
title0.setFgColor(0x000000)
```

User Interface Elements

Set Title Background Colour



Only available to the Title UI element, the **Set backgroundColor** and **Set backgroundColor RGB** block allows us to change the colour off the title bar that appears at the top of the screen when the title element is visible. For more information on colour functions available in UIFlow read chapter 4.3.3.

The MicroPython code for these blocks are:

```
title0.setBgColor(0xff0000)  
title0.setBgColor(0x000000)
```

As you can see, while the blocks operate differently, they produce the same code in MicroPython.

Set Width and Height



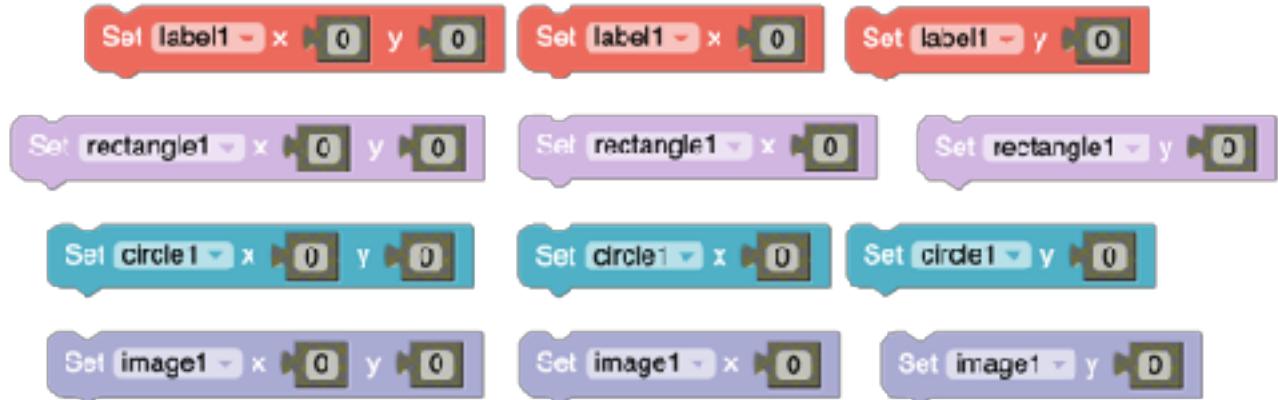
Available only to the Rectangle UI element, the set width and set height blocks are used to control the size.

The MicroPython code for these blocks are:

```
rectangle0.setSize(30, 30)  
rectangle0.setSize(width=30)  
rectangle0.setSize(height=30)
```

User Interface Elements

Set X and Y



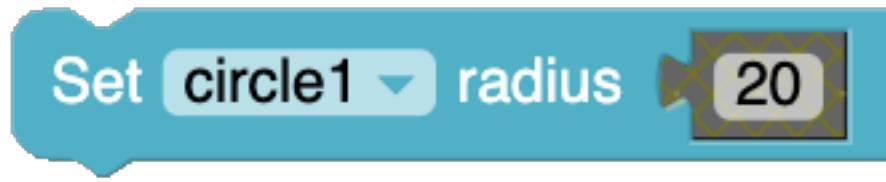
The set X and Y blocks are used to set the position of the UI element on the M5Stacks canvas using math blocks or variable blocks. By using variables as the positions we can move elements around like in games or animations.

The Micropython code for these blocks are as follows:

```
rectangle0.setPosition(0, 0)  
rectangle0.setPosition(x=0)  
rectangle0.setPosition(y=0)
```

As you can see, there are two different ways to set positions of objects in UIFlow. The first line combines the X and Y positions into one command whereas the next two lines have separate X and Y commands.

User Interface Elements



Set Circle Radius.

The Set Circle Radius block is only available to the circle UI element and can be set using maths block or variable blocks.

The MicroPython code for this block is

```
circle0.setSize(20)
```

User Interface Elements

Images.

You may have noticed from the demos shown previously that the M5Stack family of core units and sticks can display images on their screen. This is all well and good but there is a catch. For images to be used they must be created to the specifications listed below.

- **JPG** files must have the **.JPG** extension.
 - **JPG** file must have "Baseline" formatting.
 - Progressive and Lossless JPEG format is not supported.
 - Image size: Up to 65520 x 65520 pixels.
 - Colour space: YCbCr three components only.
 - Gray scale image are not supported.
 - Sampling factor: 4:4:4, 4:2:2 or 4:2:0.
-
- **BMP** images are supported.
 - Uncompressed only.
 - RGB 24-bit.
 - No colour space information.

These images can be stored on the ESP32 internal memory or onto and loaded from a microSD card placed in the built in card reader however, images stored on the SD card will take longer to be loaded and displayed due to the slower read time off the SPI bus.

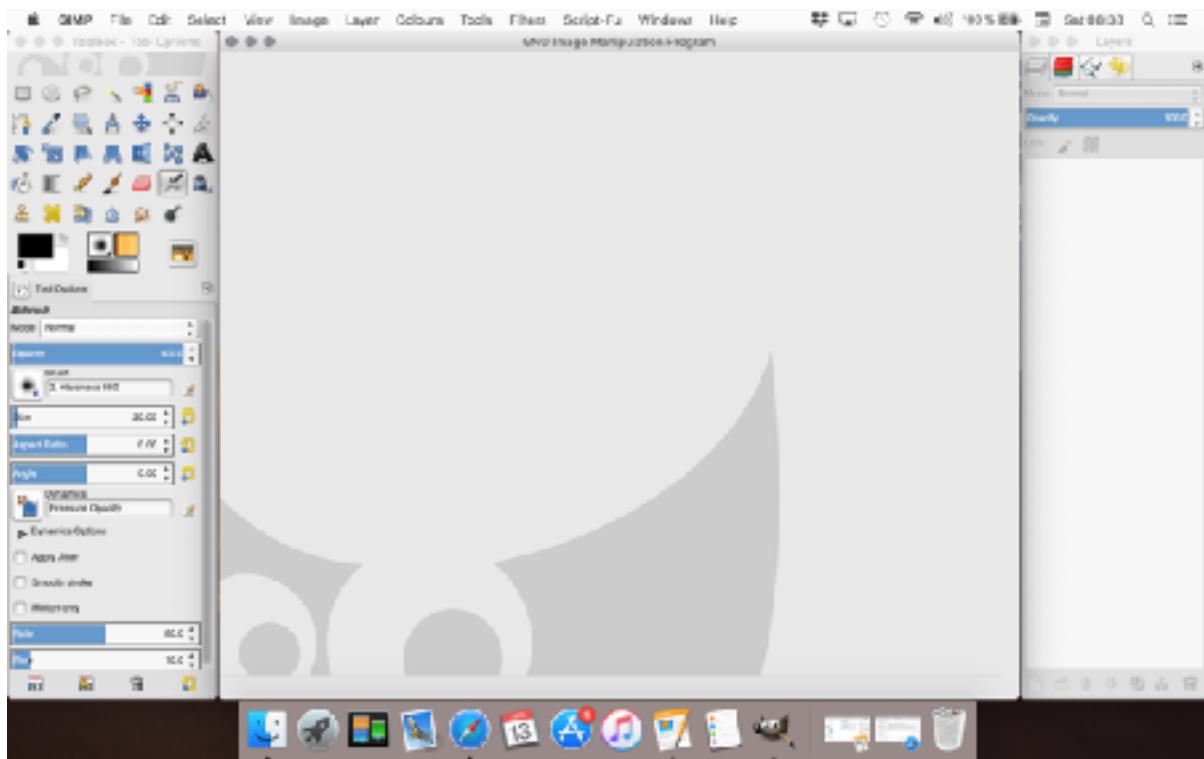
How do we create these files?

There are many image programs that can be used to create images but, not all of them can give us control over the more advance file formatting functions that are kept hidden in basic art programs.

Throughout this book I will be using the open source programs G.I.M.P and Inkscape (in fact, most of the images used in this book have been created and/or edited using G.I.M.P and Inkscape).

User Interface Elements

.BMP Files.



[Screenshot of how G.I.M.P's windows are configured on OSX 10.14.2](#)

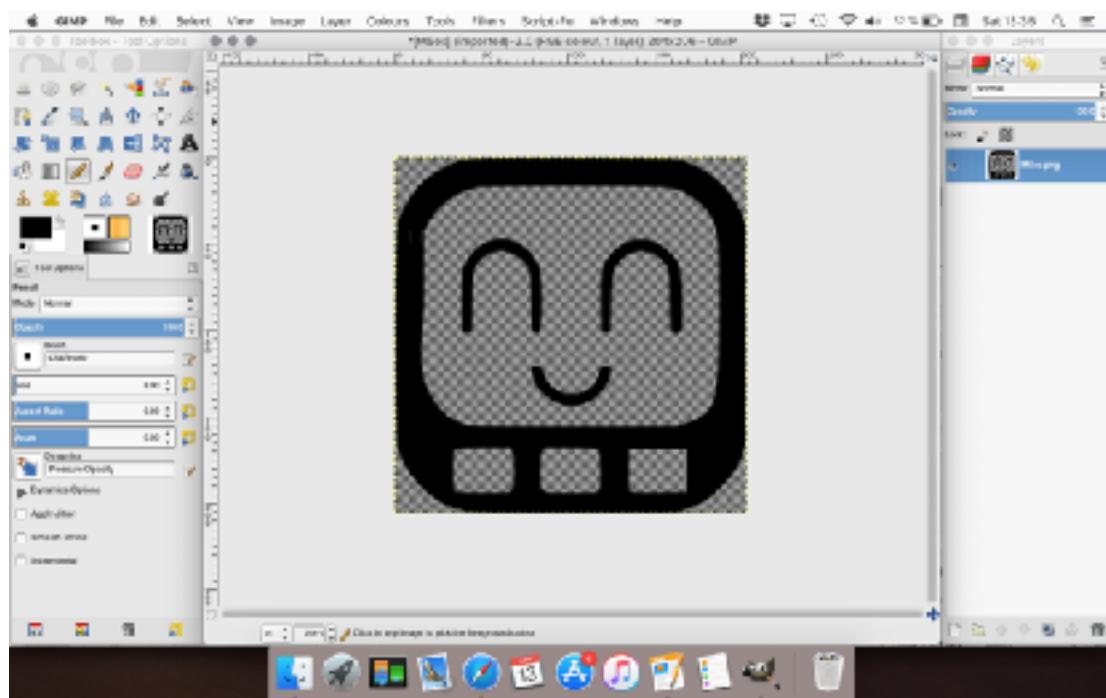
I first became aware of G.I.M.P back in 1998 when it was still in Beta test and in the last twenty years has changed greatly into a program that rivals Photoshop. Being open source, G.I.M.P is free to download from <https://www.gimp.org>

The first time G.I.M.P is loaded up it may not look like the above screen shoot as it is made of three movable windows, I will not go deeply into the operation of G.I.M.P beyond the functions we need for this chapter as that is beyond the scope of this book.

To get started, first we need an image.



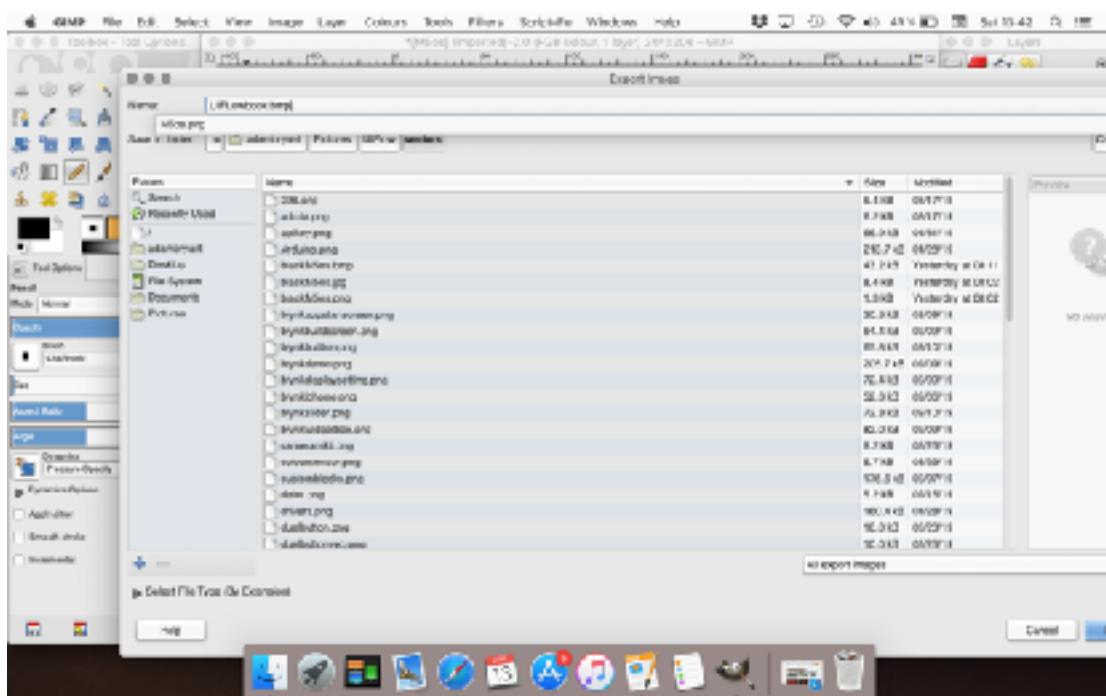
User Interface Elements



The Image I am using above has been provided by Luke from M5Stack.
When loaded up we can see that our image has a transparent background. The available file formats that MicroPython can read do not at present support transparency and when we save the image the transparent background will turn white.

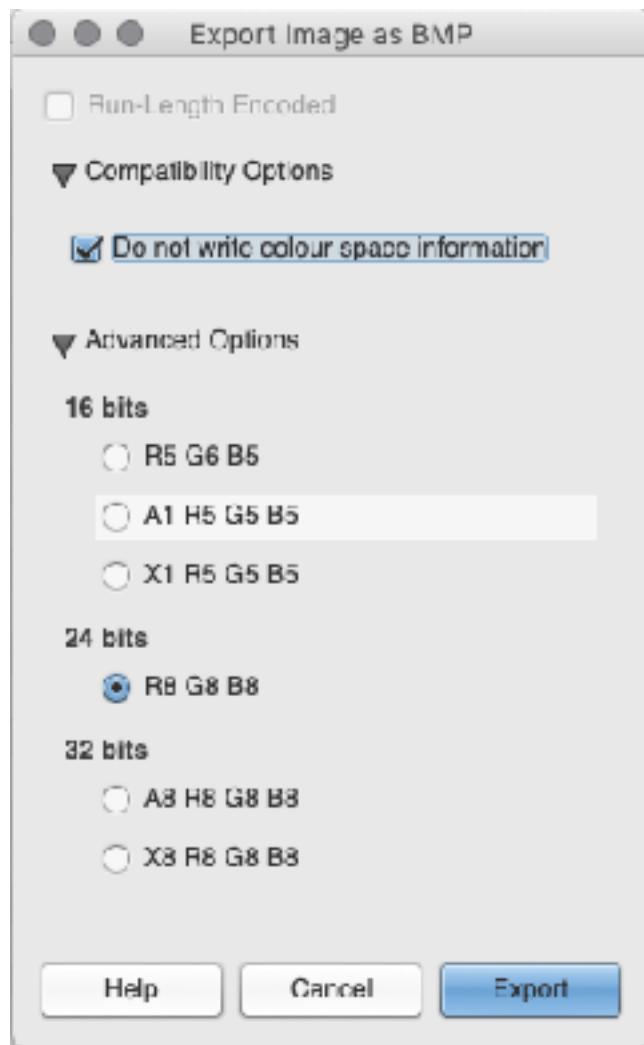
We don't need to do anything here except save the image. Goto "**File>Export As**" to bring up the export dialog.

If we just use "**File>Save AS**" the image will just be saved to G.I.M.P's native file format of **.XCF**.



User Interface Elements

The the Export Image dialogue at the top of this window is the Name: box. Type in a filename under ten characters long and put .BMP after the name. Hit return and the file format dialog appears.



In this window click on the arrow next to **Compatibility Options** and a box will appear with a tick box next to **Do not Write colour space information**.

Click the box to make an arrow appear in it and then click the **Advanced Options** arrow and the above options will appear. Click on the circle under **24 bits** and then click on export to save the file.

If you look at the file in the file manager it will show a file size of **124Kb** this is way over the limit we have of **24Kb** that Micropython can read.

To make it smaller we need to to some editing of the file.

For this go too, **Image>Scale image** and change the image size from **204x206px** to **102x103px** and click Ok. Export the file again and check the file size.

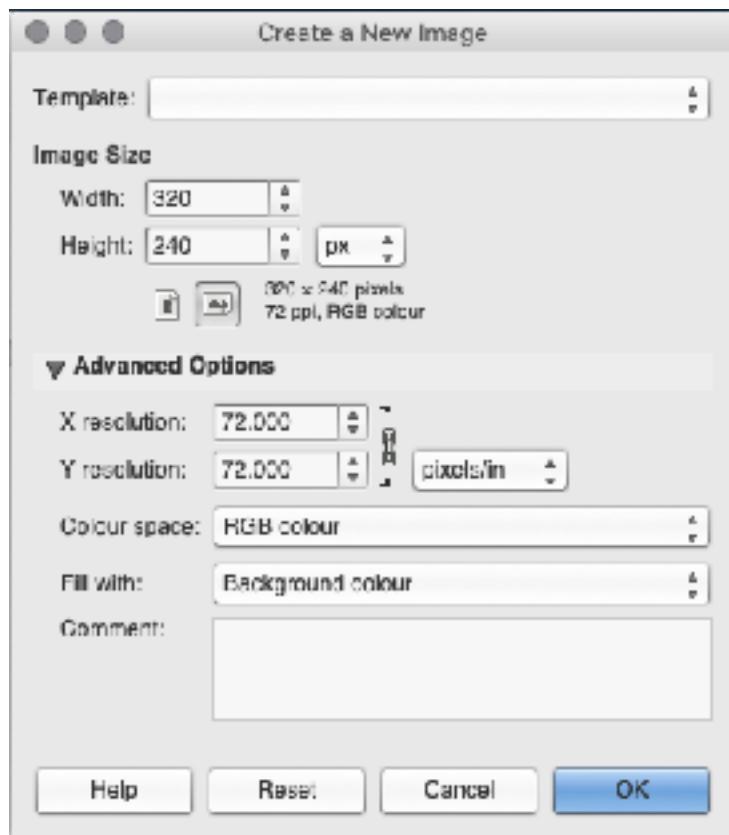
This step has reduced the size from 124Kb down to 32Kb but, it is still too big for Micropython. Goto **Image>Mode** and select **Grayscale**, Save it again and we can see that the file size is now only 12kb.

User Interface Elements

If we upload the file new file to the M5Stack and try to load it noting will happen because grayscale mode just doesn't work. Go back to **Image>Mode** and select and change it back to **24bit mode**. The only thing we have left is to scale the image down a little bit more and report it until we can get the file size low enough.

.jpg files.

Creating a .jpg file is almost the same as creating the .bmp files but we have a few more steps to follow. In G.I.M.P go to **File>New Image** and set the option to the same as in the screen shot

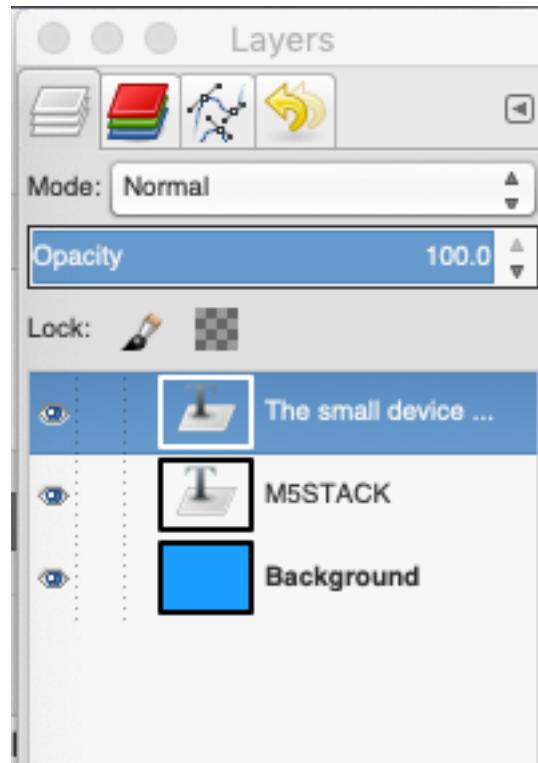


below.



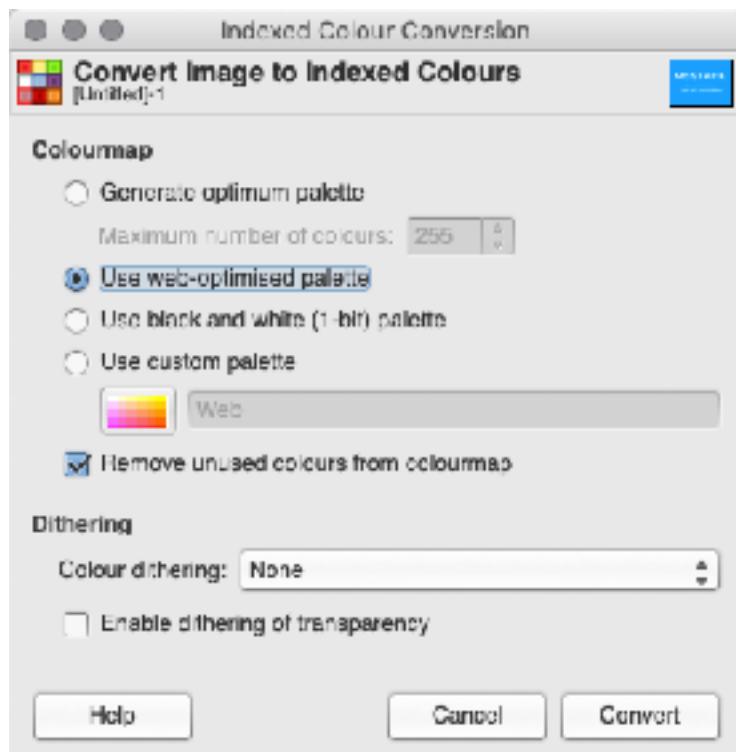
User Interface Elements

Next fill the image with a design. In the next image I filled the background in blue and just added the text in white.



Next look at the Layers panel and you will see that we have three layers.

The .jpg file format does not support layers and so we need to flatten the image into one layer. Go to **Layer>Merge Down** to merge the top layer with the one below it. Repeat the merge down so that we only have one layer.



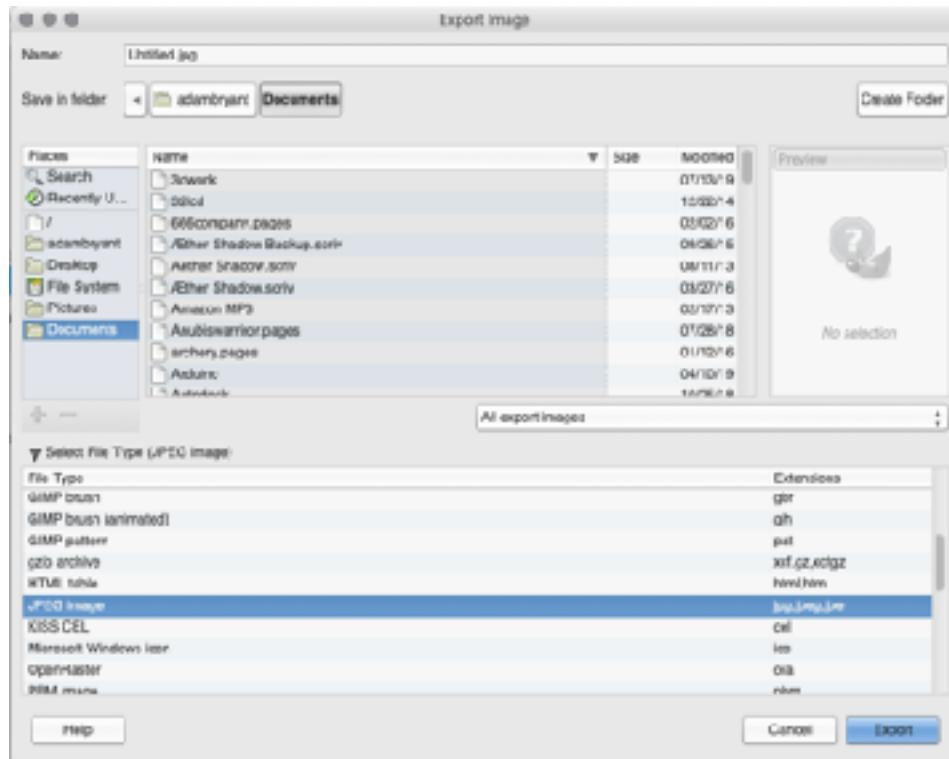
User Interface Elements

Next click on **Image>Mode>Index** and set the options as follows

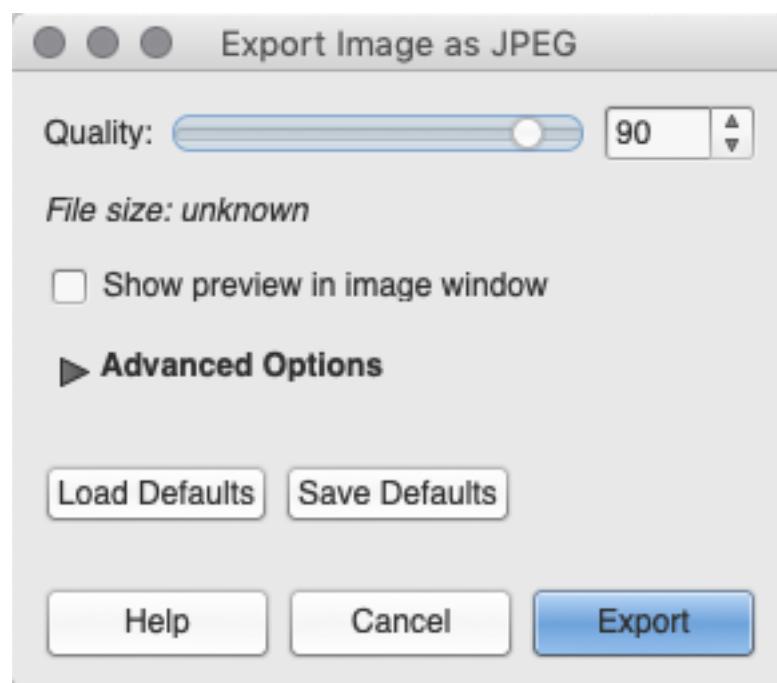
This reduces the images colour data in the colour map saving us some data and reducing the file size as well as making the file easier for the ESP32 to read.

Click on **Convert** and the window will apply the changes and close.

Now we have our image repaired it is time to save and create a .jpg file. The programmers changed the way G.I.M.P saves files so that Save and Save As only save to G.I.M.P's .xcf file



format. To create the .jpg file we need to click on **File>Export** to bring up the export dialog.

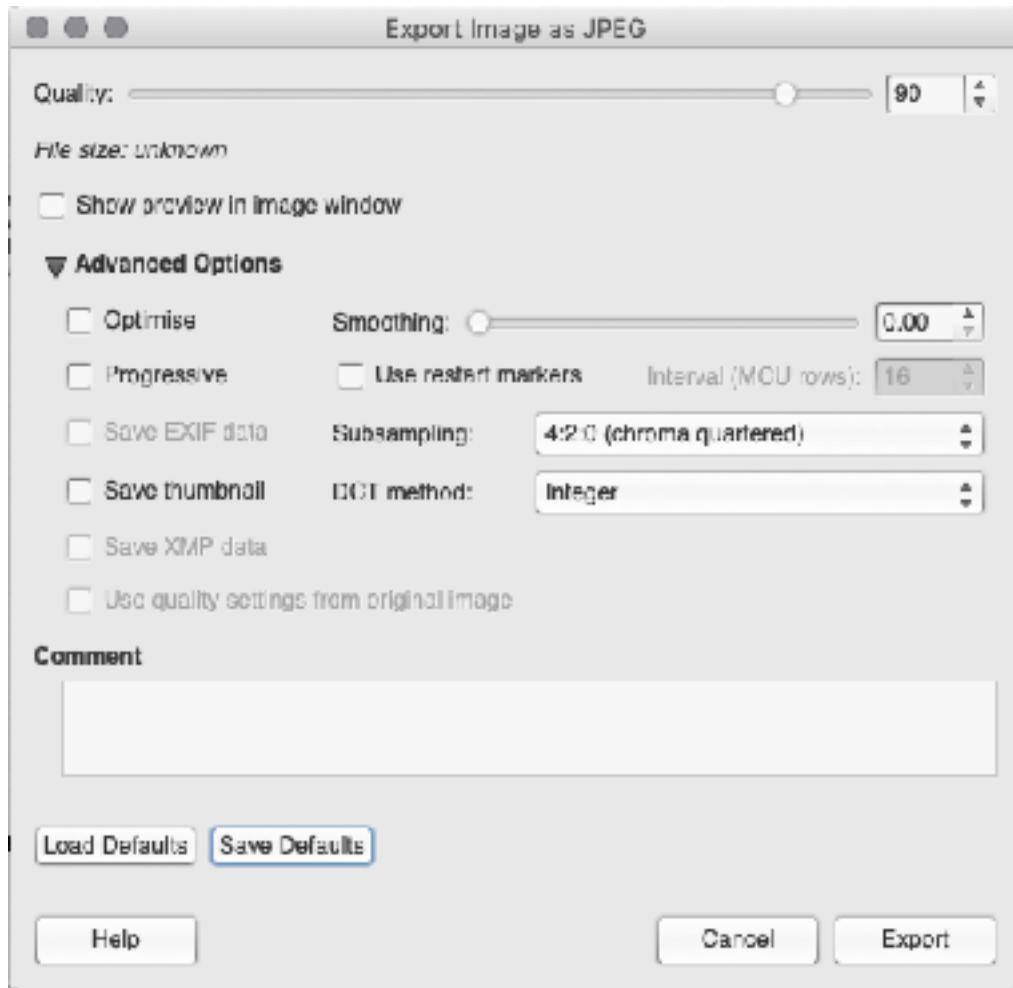


User Interface Elements

The file type box may be collapsed, click on the black arrow next to **Select File Type** to open the box and scroll down to find the JPEG Image Click on JPEG Image to highlight it and the file name at the box will change to Untitled.jpg.

Give the file a name under seven characters long and click **Export**.

In the next dialog, click on the arrow next to **Advanced Options** and change the setting to match



those in the screen shot below and click on **Export** to save the file.

The M5Stack firmware can only handle images up to 24kb in size. The following screen shot shows the same file (saved with a different name to make them stand out) the first has been saved with 90% quality and is to big for the M5Stacks memory. to reduce the file size, slide the Quality slider to the left to reduce the quality of the image. Because my image is simple, there is no loss in the quality of the images appearance after I reduced the quality to 80%.

emoticonbg.jpg	Today at 06:56	39 KB	JPEG Image
emotibg.jpg	Today at 07:00	14 KB	JPEG Image

Displaying Images on screen.

Now that we have our images how do we get them to show up on the screen?

The first method to get the image to display is to directly uploaded it to the M5Stacks internal memory.

User Interface Elements

At the top of the screen you will find UIFlows icon menu



This menu contains several useful functions.



Starting from the left icon we have:

IDE Switcher - This icon when clicked shows a menu that allows us to switch between versions of the UIFlow IDE. Some times we will encounter code that may work with one version but not with another.



Forum link - This icon when clicked will take you to the M5Stack forums.



The Documentation - This will take you to the section of the website where the online documentation is currently stored.



Code Examples - Clicking this icon will cause a window to appear containing demo and sample code.



Undo and Redo allow us to undo changes we made to code or redo the changes if we change our mind of the previous change didn't work.



M5 Resource Manager - this button allows us to connect to the M5Stack or stick and upload or download files from the internal memory.



Play/Test - This button temporarily uploads code onto M5Stack or sticks to test however, the code is lost if the device is reset.



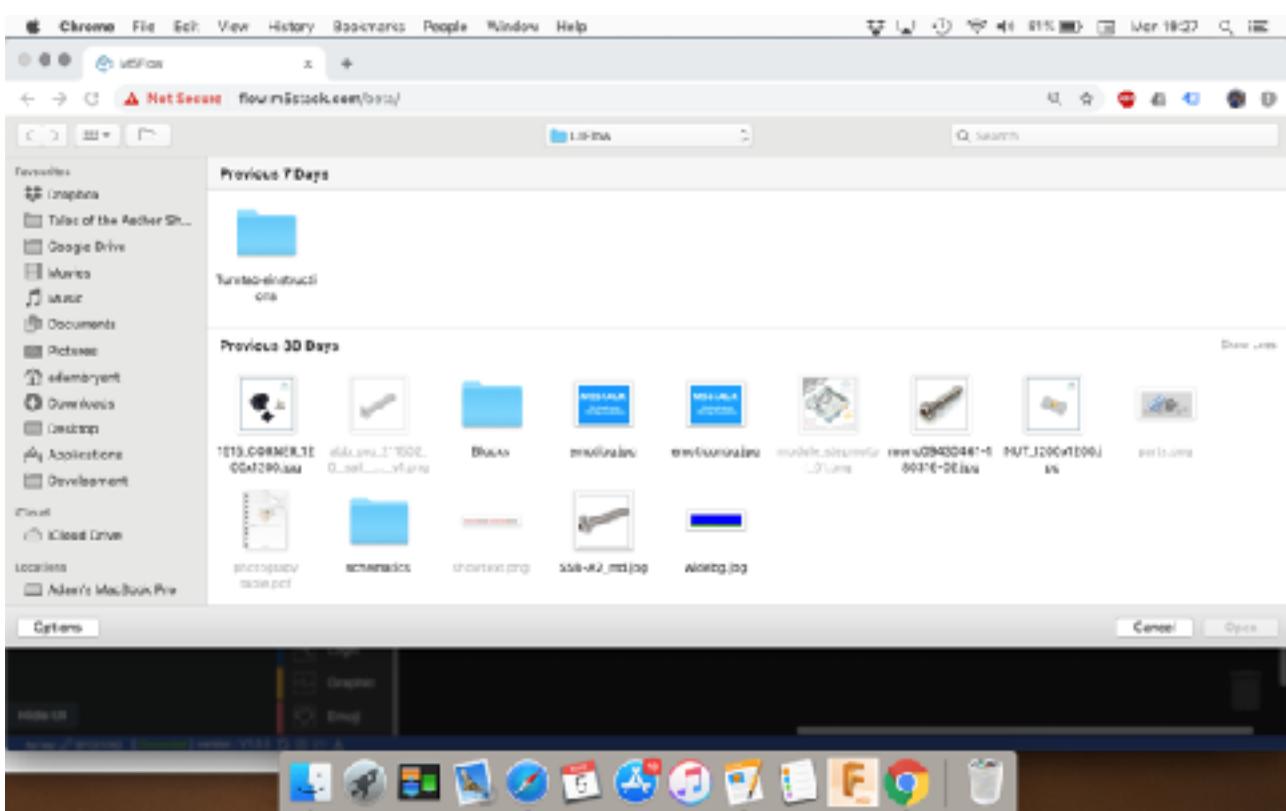
UIFlows Setting Menu - This button brings up the setting menu of UIFlows IDE allowing us to change the settings used by UIFlow to compile code for the M5Stack or the M5Stick. I will explain more about this menu later in the book.

For uploading images we will need to use the M5 File Manager.

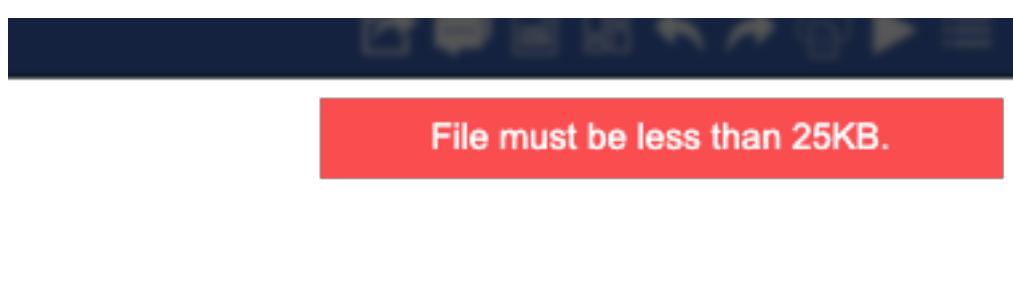
User Interface Elements



Click on the icon shown above and the resource manager panel will be displayed. In this screen shot you can see that I have two images already loaded onto the M5Stack. To delete a file from the internal storage you just need to click on the "Delete" button next to the file. To add a file, click on "Add Images" and the computers operating system file manager will open.



Click on a .jpg image and hit ok. If the image is below the file size of 25kb it will be added to the list, if not you will see the following pop in from the right hand edge of the screen.



User Interface Elements

If the file is the correct size then the gray bar under the file name will start to brighten from the left end to the right end showing that it is in the process of being uploaded. If the image doesn't show up you can try clicking on reload.

Now we have images on the M5Stack we can start writing code to display the stored images.

There are several ways we can do this.

The easiest way to display an image on screen is to use the image loader found in the U.I Builder

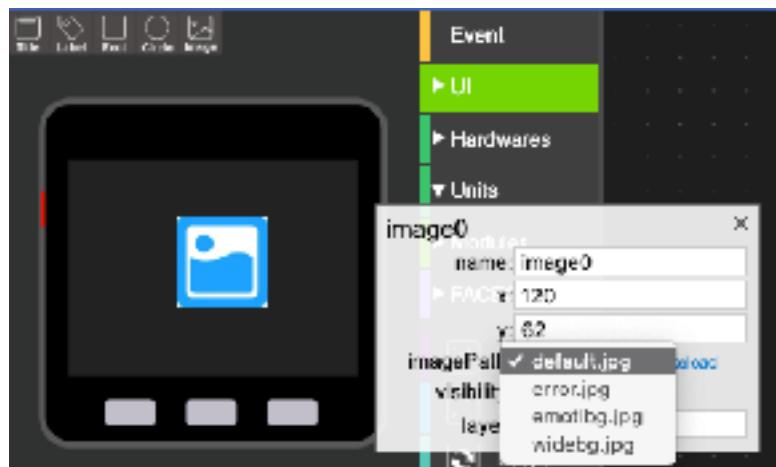


panel above the virtual M5Stack.

Click on it and drag it onto the screen.



User Interface Elements



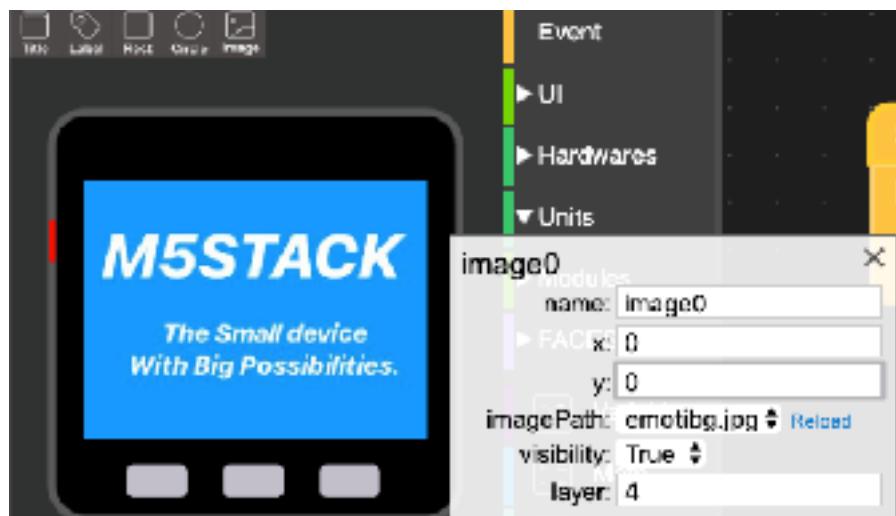
Now we click on the blue and white icon to access the image options and then click on "ImagePath" to drop down a list of files stored on the m5stack

In the following image we can see that the picture is not aligned properly on the screen.



To correct this we need to click on the image again to access the setting and set both X and Y to zero. You will notice that as you type the numbers the image will move around the screen until it reaches the position shown below.

All we need to do now is to press the play/test button and the image will now be shown on screen.



Internal Hardware

The M5Stack and M5Stick family have some differences on the inside due to configuration options available. The blocks in this section allow you to access those functions available in the various hardware

Speaker

Speaker,

Speaker.Beep Frequency,

Speaker.beep freq: 1600 duration: 200 ms

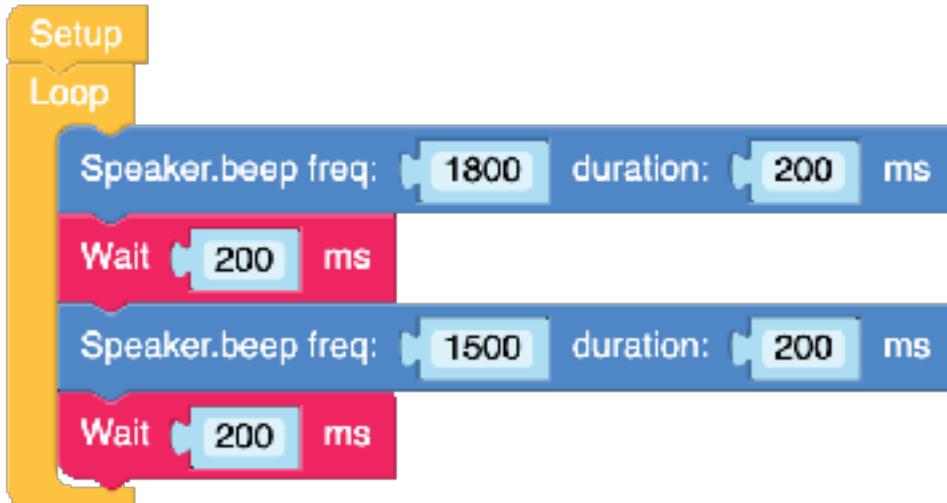
The Speaker Beep Frequency block allows us to make sounds by setting the frequency of the sound and the duration of the sound. We can use the value blocks to manually set the frequency and duration or we can replace these value blocks with a variable block or a value returned from a hardware block.

The Micropython code for this is:

speaker.tone()

Example

In the following example, when run, will play two tones from the M5Stacks internal speaker.



The Micropython code for this example is as follows

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

while True:
    speaker.tone(1800, 200)
    wait_ms(200)
    speaker.tone(1500, 200)
    wait_ms(200)
    wait_ms(2)
```

Speaker

Speaker Volume,



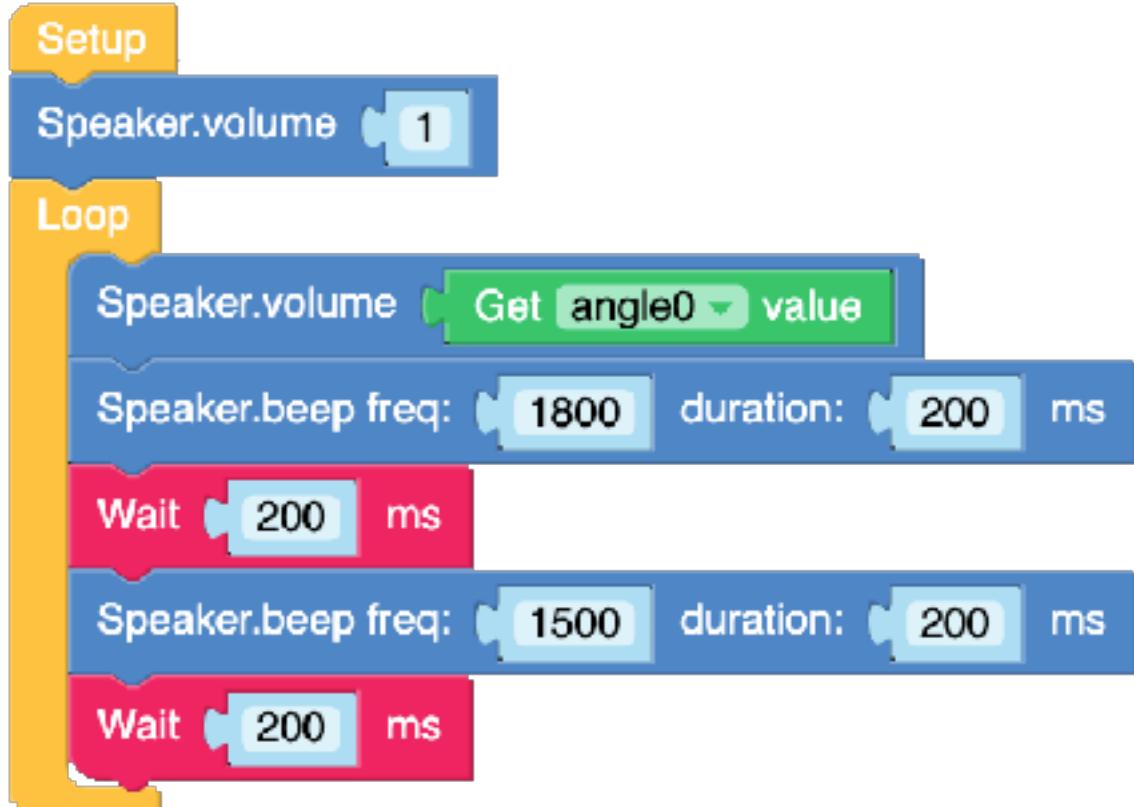
Speaker volume allows us to set the volume of the speaker sounds. The volume has to be initially set before the main loop and can be changed later in the loop. We can use the value block to manually set the volume or we can replace these value blocks with a variable block or a value returned from a hardware block.

The Micropython code for this is:

```
speaker.setVolume()
```

Example

In the following example, I have just added the volume block that gets its value from the angle sensor connected to port B.



Speaker

The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x222222)
angle0 = unit.get(unit.ANGLE, unit.PORTB)

speaker.setVolume(1)
while True:
    speaker.setVolume((angle0.read()))
    speaker.tone(1800, 200)
    wait_ms(200)
    speaker.tone(1500, 200)
    wait_ms(200)
    wait_ms(2)
```

Play Tone.

play tone Low A for 1 beat

The Play Tone block allows users to play musical notes and set the duration in beats or fractions of Beats.

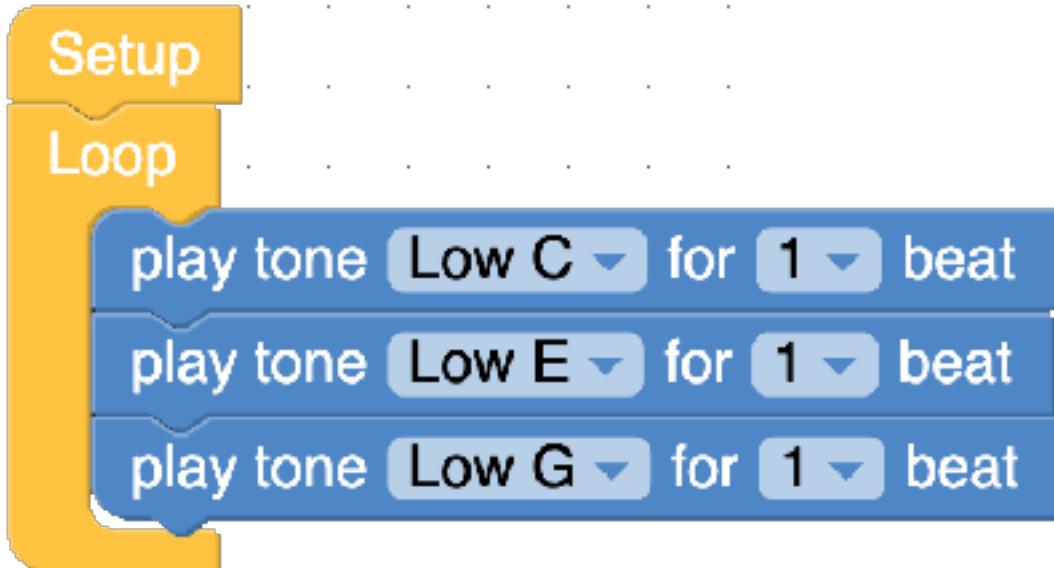
The Micropython code for this is:

```
speaker.sing()
```

Speaker

Example

The following example plays three note in the loop continuously.



And the Micropython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

while True:
    speaker.sing(131, 1)
    speaker.sing(165, 1)
    speaker.sing(196, 1)
    wait_ms(2)
```

RGB Bar Colour

**RGB,
Set RGB Bar Colour,**

Set RGB Bar color



Sets all the WS2812b LED colour in the M5Go base to the same colour using the colour selector.
The MicroPython code for this block is:

```
rgb.setColorAll(0xff0000)
```

Example

In this example, all the RGB LED's in the bases left and right bars are lit up in red.

Setup

Set RGB Bar color



The MicroPython code for this is:

```
from m5stack import *
from m5ui import *
from uiflow import *
setScreenColor(0x222222)
rgb.setColorAll(0xff0000)
```

RGB Bar Colour

Set RGB Bar Colour R,G,B,

Set RGB Bar color R

0

0

0

Sets all the WS2812b LED colour using separate number defined using maths number blocks or variables.

The Micropython code for this block is:

```
rgb.setColorAll(0xff0000)
```

Example:

In this example, all the RGB LED's in the bases left and right bars are lit up by defining the values

Setup

Set RGB Bar color R

255

0

0

for Red, Green and, Blue using numbers.

The Micropython code for this demo is:

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)
rgb.setColorAll(0xff0000)
```

As you can see, while the values are entered into the blocks using numbers, the Micropython code values are generated in hexadecimal values.

RGB Bar Colour

Set (Side) RGB Bar Colour,



Sets all the WS2812b LED colour on the left or the right side of the M5Go base to the same colour using the colour selector.

The Micropython code for this block is:

```
rgb.setColorFrom(6, 10, 0xff0000)
```

As you can see, the Micropython code doesn't have a left or right option instead, it uses a version of the RGB Range block. For more information on the RGB Range block, please view the RGB LED section later in the book.

Example.



In this example I alternate which side lights up.

RGB Bar Colour

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

rgb.setColorAll(0x000000)
while True:
    rgb.setColorFrom(6 , 10 ,0xff0000)
    wait(1)
    rgb.setColorFrom(1 , 5 ,0xff0000)
    wait(1)
    wait_ms(2)
```

RGB Bar Colour

Set (Side) RGB Bar Colour R,G,B,

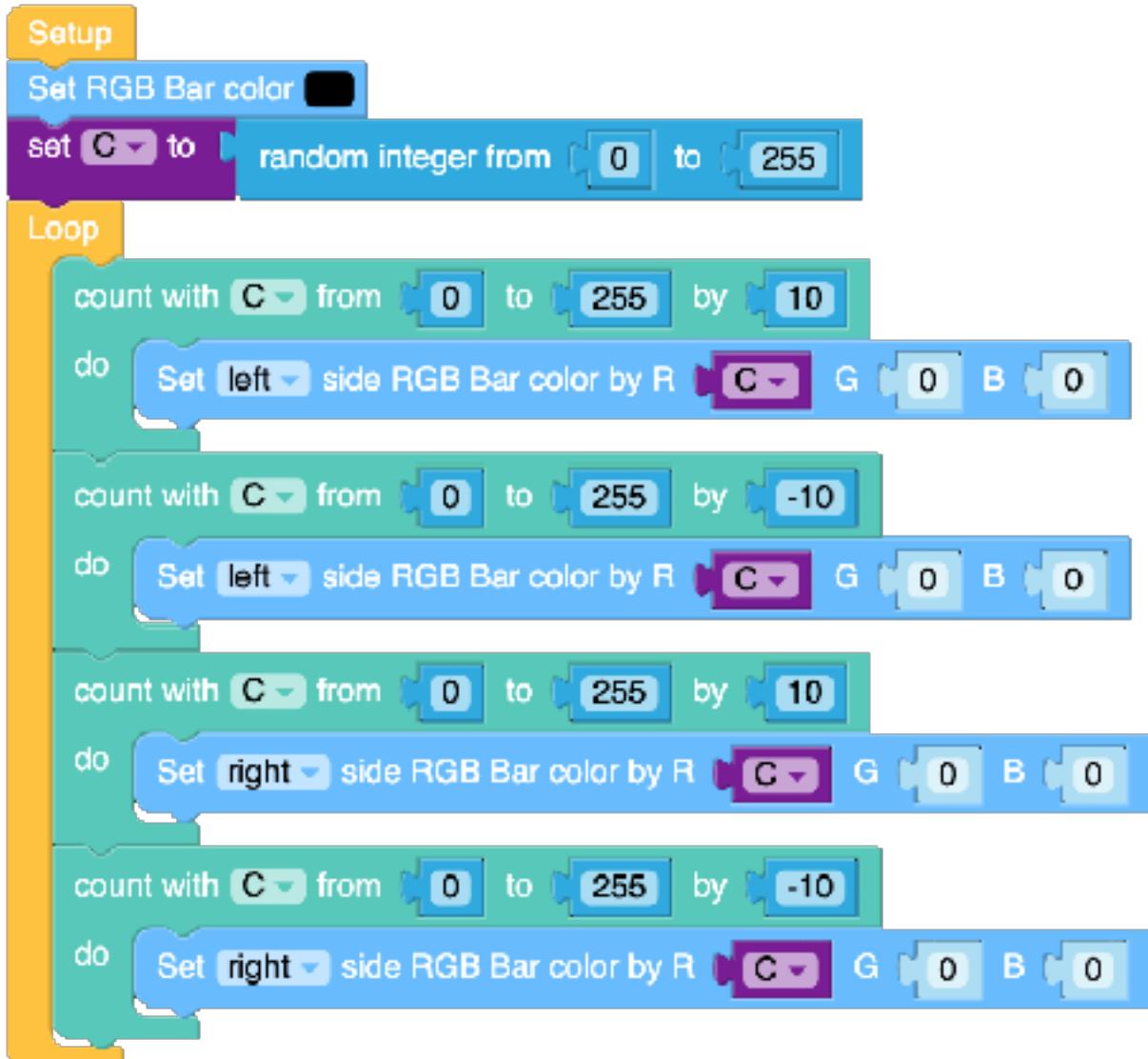
Set left side RGB Bar color by R 0 G 0 B 0

Sets all the WS2812b LED colour on the left or the right side of the M5Go base to the same colour using individual Red, Green and Blue colour values set by maths number blocks or variables..

As you can see, the Micropython code for this block is the same as the previous block:

```
rgb.setColorFrom(6, 10, 0xff0000)
```

Example.



RGB Bar Colour

In this example I am increasing and decreasing the colour of the bars one side at a time.
The Micropython for this is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

import random

C = None

rgb.setColorAll(0x000000)
C = random.randint(0, 255)
while True:
    for C in range(0, 256, 10):
        rgb.setColorFrom(6 , 10 ,(C << 16) | (0 << 8) | 0)
    for C in range(0, 256, 10):
        rgb.setColorFrom(6 , 10 ,(C << 16) | (0 << 8) | 0)
    for C in range(0, 256, 10):
        rgb.setColorFrom(1 , 5 ,(C << 16) | (0 << 8) | 0)
    for C in range(0, 256, 10):
        rgb.setColorFrom(1 , 5 ,(C << 16) | (0 << 8) | 0)
    wait_ms(2)
```

RGB Bar Colour

Set (individual) RGB Colour,



Sets one of the ten individual WS2812B LEDs colour on the M5Go base using the colour Picker.
The position can be set using a maths block or a variable block.

The MicroPython code for this is:

```
rgb.set_color(1, 0xff0000)
```

Example



In this example I am lighting each of the ten WS2812B LEDs in the Go base one after the other using a variable to chose which WS2812b should light up.

RGB Bar Colour

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

C = None

rgb.setColorAll(0x000000)
C = 0
while True:
    C = (C if isinstance(C, int) else 0) + 1
    rgb.setColor(C, 0xff0000)
    wait(1)
    wait_ms(2)
```

RGB Bar Colour

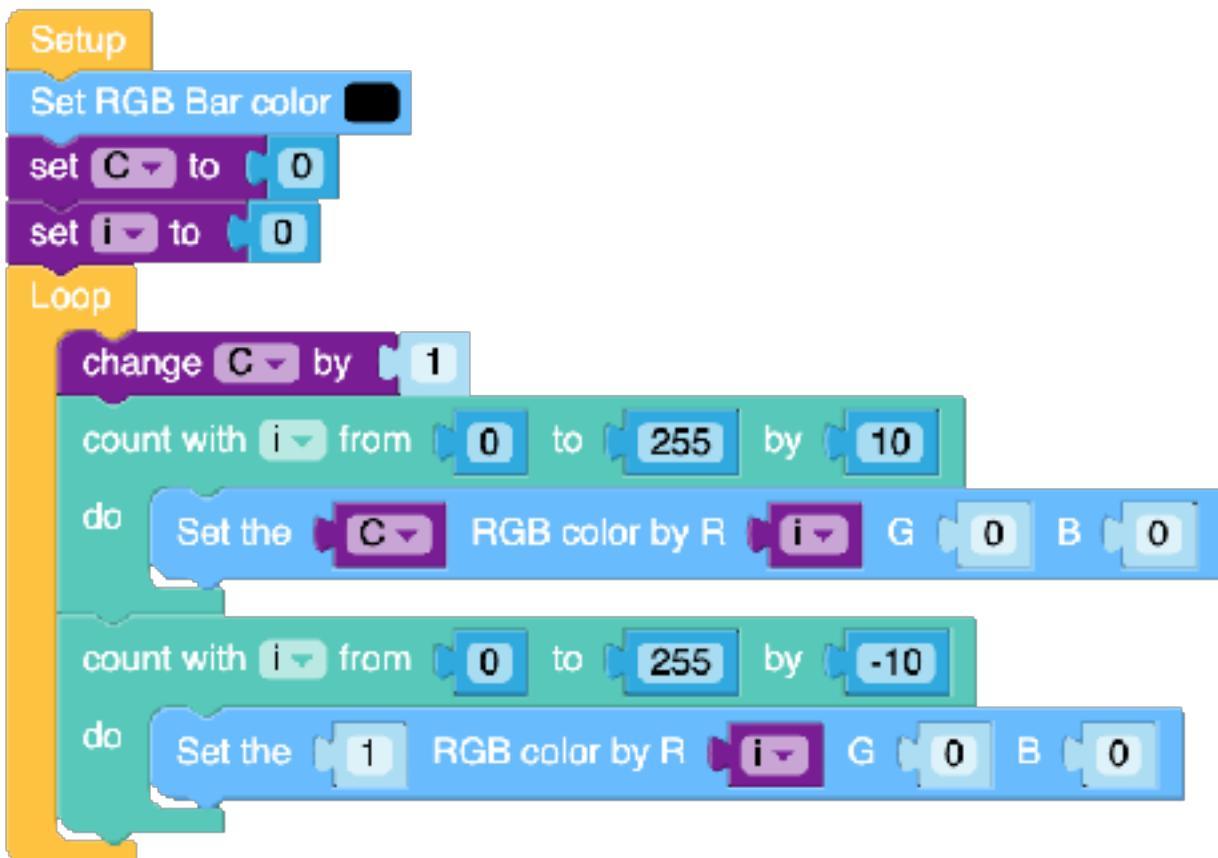
Set (individual) RGB Bar Colour R,G,B,

Set the [1] RGB color by R [0] G [0] B [0]

Sets one of the ten individual WS2812B LEDs colour on the M5Go base using individual Red, Green and Blue colour values using maths blocks or variable blocks.

The MicroPython code for this is:

Example



RGB Bar Colour

Using the previous example, we can expand it to also alter the colours as well as which WS2812b LED illuminates.

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x222222)
angle0 = unit.get(unit.ANGLE, unit.PORTB)

C = None
i = None

rgb.setColorAll(0x000000)
C = 0
i = 0
while True:
    C = (C if isinstance(C, int) else 0) + 1
    for i in range(0, 256, 10):
        rgb.setColor(C,(i << 16) | (0 << 8) | 0)
    for i in range(0, 256, 10):
        rgb.setColor(1,(i << 16) | (0 << 8) | 0)
    wait_ms(2)
```

RGB Bar Colour

Set RGB Brightness,

Set RGB brightness



Sets the brightness level of all the WS2812B LEDs in the M5Go baseplate. We can use Maths blocks or variables to control the brightness of the WS2812b LED's and if you have the Angle sensor, we can also use that as a "dimmer" like control.

The MicroPython code for this is:

```
rgb.setBrightness(10)
```

Example



In this example I have created a simple dimmer control for the LEDs

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x222222)
angle0 = unit.get(unit.ANGLE, unit.PORTB)

rgb.setColorAll(0xff0000)
while True:
    rgb.setBrightness((angle0.read()))
    wait_ms(2)
```

IMU

IMU.

The M5StickC has a built in gyro which can be used for movement tracking. In order to control the IMU (Inertial Measurement Unit) we use the following blocks which return numbered strings.

Get X,

Get X

Gets the X Value from the IMU and returns it as a number that can be used instead of a maths value block or a variable block.

The Micropython code for this block is:

(imu0.ypr[1])

Example



In this example I am using the GetX block to set the X position of a circle while a Label show the numerical value of X.

The Micropython code for this example is.

```
from m5stack import *
from m5ui import *
from uiflow import *
import imu

setScreenColor(0x222222)

imu0 = imu.IMU()
label0 = M5TextBox(93, 223, "Text", lcd.FONT_Default,0xFFFFFFF, rotate=0)
circle0 = M5Circle(164, 115, 15, 0xFFFFFFF, 0xFFFFFFF)

while True:
    label0.setText(str(imu0.ypr[1]))
    circle0.setPosition(x=(imu0.ypr[1]))
    wait_ms(2)
```

Get Y,

Get Y

Gets the Y Value from the IMU and returns it as a number that can be used instead of a maths value block or a variable block.

The Micropython code for this block is

(imu0.ypr[2])

Example



In this example I am using the GetY block to set the Y position of a circle while a Label show the numerical value of Y.

The Micropython code for this example is.

```
from m5stack import *
from m5ui import *
from uiflow import *
import imu

setScreenColor(0x111111)

imu0 = imu.IMU()
label0 = M5TextBox(93, 223, "Text", lcd.FONT_Default,0xFFFFFFF, rotate=0)
circle0 = M5Circle(164, 115, 15, 0xFFFFFFF, 0xFFFFFFF)

while True:
    label0.setText(str(imu0.ypr[2]))
    circle0.setPosition(x=(imu0.ypr[2]))
```

IMU

Get X ACC,

Get X ACC

Gets the X acceleration value from the IMU.

The Micropython code for this block is:

`imu0.acceleration[0]`

Example



In this example I have just used a label to show the X accelerations reading from the M5Sticks IMU.

The Micropython code for this demo is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import imu

setScreenColor(0x111111)

imu0 = imu.IMU()
label0 = M5TextBox(14, 36, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(imu0.acceleration[0]))
    wait_ms(2)
```

Get Y ACC,

Get Y ACC

Gets the Y acceleration value from the IMU.

The Micropython code for this block is:

```
imu0.acceleration[1]
```

Example



In this example I have just used a label to show the Y accelerations reading from the M5Sticks IMU.

The Micropython code for this demo is as follows.

```
ffrom m5stack import *
from m5ui import *
from uiflow import *
import imu

setScreenColor(0x111111)

imu0 = imu.IMU()
label0 = M5TextBox(14, 36, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(imu0.acceleration[1]))
    wait_ms(2)
```

IMU

Get Z ACC,

Get Z ACC

Gets the Z acceleration value from the IMU.

The Micropython code for this block is:

`imu0.acceleration[2]`

Example

In this example I have just used a label to show the Z accelerations reading from the M5Sticks IMU.



The Micropython code for this demo is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import imu

setScreenColor(0x111111)

imu0 = imu.IMU()
label0 = M5TextBox(14, 36, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(imu0.acceleration[2]))
    wait_ms(2)
```

It is worth noting that X, Y and, Z are not used in the code but instead use 0, 1 and, 2 respectively to represent X, Y, and, Z.

Get X Gyro,

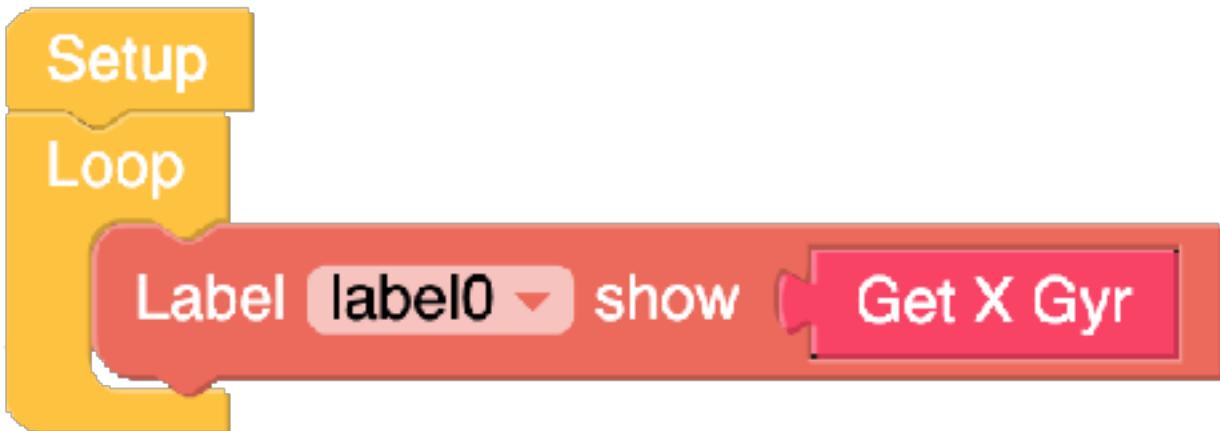
Get X Gyr

Gets the X value from the IMU's Gyro.
The Micropython code for this block is:

imu0.gyro[0]

Example

In this example I have just used a label to show the X rotation reading from the M5Sticks IMU.



The Micropython code for this demo is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import imu

setScreenColor(0x111111)

imu0 = imu.IMU()
label0 = M5TextBox(14, 36, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(imu0.gyro[0]))
    wait_ms(2)
```

IMU

Get Y Gyro,

Get Y Gyr

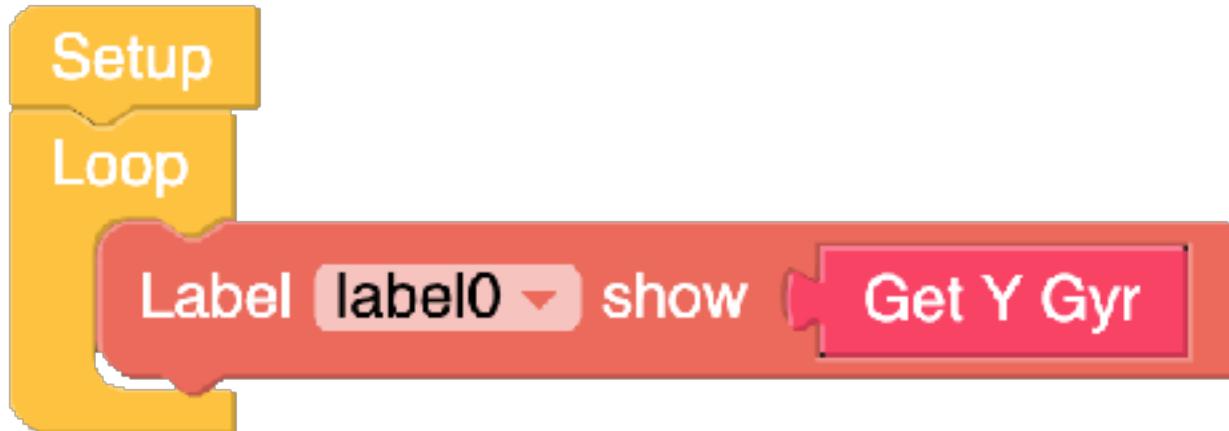
Gets the Y value from the IMU's Gyro.

The Micropython code for this block is:

Example

imu0.gyro[1]

In this example I have just used a label to show the X rotation reading from the M5Sticks IMU.



The Micropython code for this demo is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import imu

setScreenColor(0x111111)

imu0 = imu.IMU()
label0 = M5TextBox(14, 36, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(imu0.gyro[1]))
    wait_ms(2)
```

Get Z Gyro,

Get Z Gyr

Gets the Z value from the IMU's Gyro.
The Micropython code for this block is:

imu0.gyro[2]

Example

In this example I have just used a label to show the X rotation reading from the M5Sticks IMU.



The Micropython code for this demo is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import imu

setScreenColor(0x111111)

imu0 = imu.IMU()
label0 = M5TextBox(14, 36, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(imu0.gyro[2]))
    wait_ms(2)
```

Power P1 - AXP192

Power (StickC)

The following power blocks are only available to units that are fitted with a modified version of the power control chip which allows the ESP32 to communicate with the power management chip.

Get Charge State

Get charge state

The Get Charge State block is a value block which returns the status from the power management chip. The block reports false if there is no USB power connected or True if a USB cable is plugged in and connected to a supply.

The Micropython code for this is.

```
axp.getChargeState()
```

Example



In this example I have just created a label that shows if a USB cable is plugged in and the M5Stick is charging. The Micropython code is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(19, 30, "Text", lcd.FONT_Default, 0xFFFF, rotate=0)

while True:
    label0.setText(str(axp.getChargeState()))
    wait_ms(2)
```

Get Battery Voltage

Get battery voltage

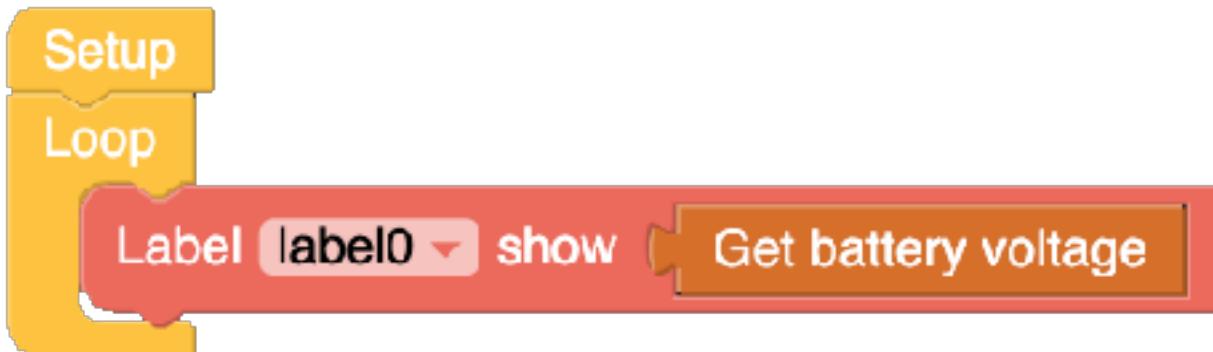
Returns the current voltage from the internal battery.

The Micropython code for this block is.

AXP.GETBATVOLTAGE()

Example

In the following example I am just using a label to show the voltage of the internal battery.



The Micropython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(10, 88, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(axp.getBatVoltage()))
    wait_ms(2)
```

Power P1 - AXP192

Get Battery Current

Get battery current

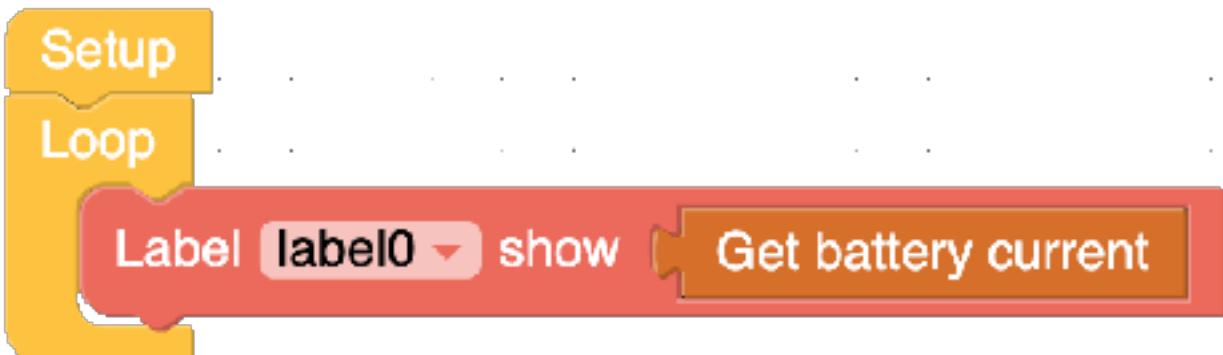
Returns the current in millamps from the internal battery.

The Micropython code for this block is

```
axp.getBatCurrent()
```

Example

In this example I have just created a label that show internal battery current use on the screen.



The Micropython code is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(21, 57, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(axp.getBatCurrent()))
    wait_ms(2)
```

Get Vin Voltage



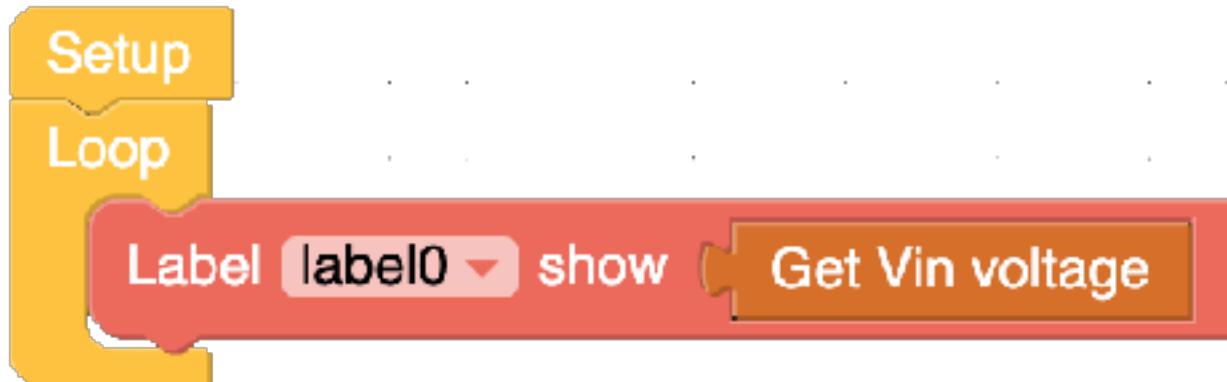
Returns the voltage on the input pin of the HAT connector on the top of the M5StickC

The Micropython code for this block is

```
axp.getVinVoltage()
```

Example

The following example uses a label show the voltage on the screen.



The Micropython code is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(14, 36, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(axp.getVinVoltage()))
    wait_ms(2)
```

Power P1 - AXP192

Get Vin Current

Get Vin current

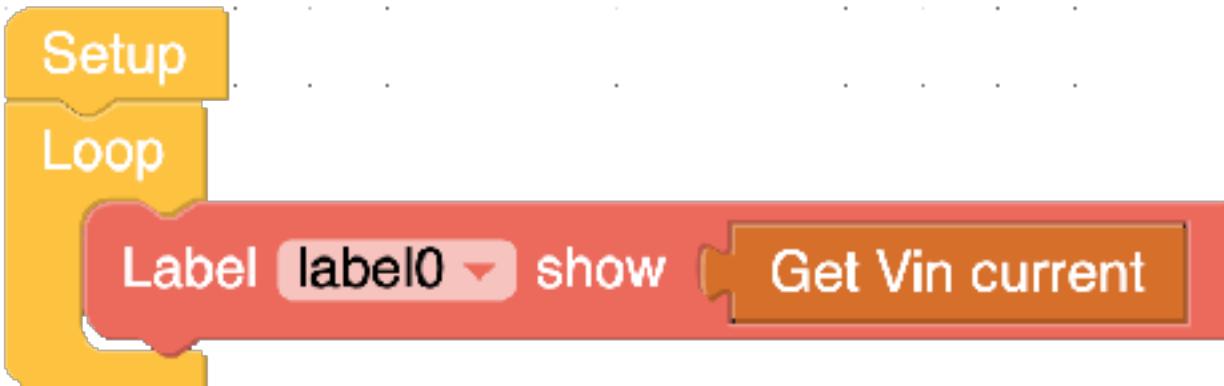
Returns the current on the input pin of the HAT connector on the top of the M5StickC

The Micropython code for this block is

```
axp.getVinCurrent()
```

Example

This example is the same as the Vin Voltage demo but with the voltage block changed for the Vin Current Block.



The Micropython code for this example is

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(5, 40, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(axp.getVinCurrent()))
    wait_ms(2)
```

Get VBus Voltage

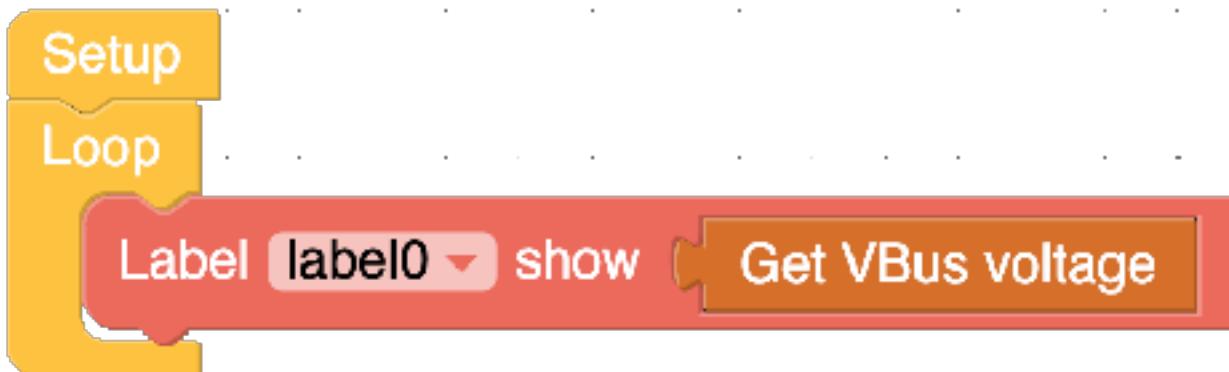
Get VBus voltage

The Micropython code for this block is

```
axp.getVBusVoltage()
```

Example

This example is the same as the Vin Voltage demo but with the voltage block changed for the VBus Voltage Block.



The Micropython code for this example is

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(5, 36, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(axp.getVBusVoltage()))
    wait_ms(2)
```

Power P1 - AXP192

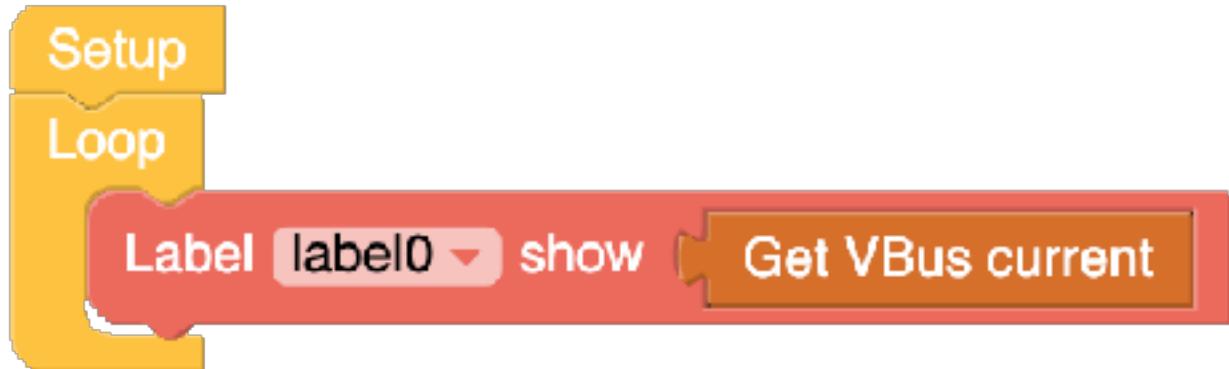
Get VBus Current

Get VBus current

The Micropython code for this block is

```
axp.getVBusCurrent()
```

Example



The Micropython code for this block is

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(11, 26, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(axp.getVBusCurrent()))
    wait_ms(2)
```

Get AXP192 Temperature

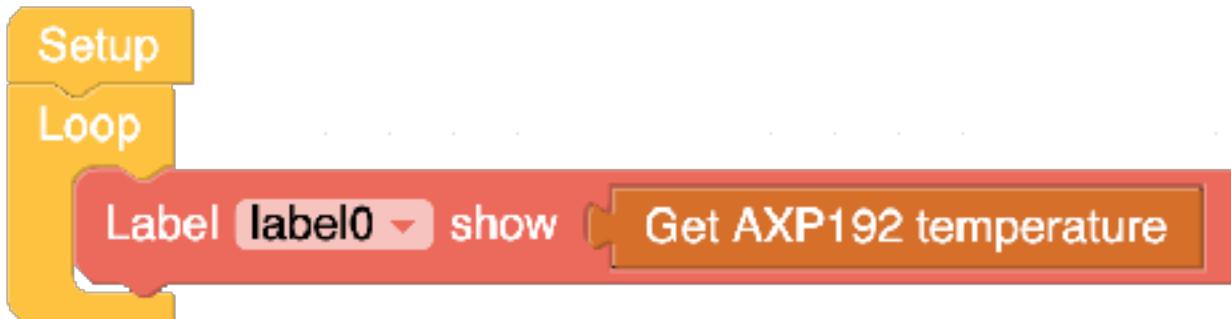
Get AXP192 temperature

Returns the temperature from the AXP192's internal temperature sensor.

The Micropython code for this block is

```
axp.getTempInAXP192()
```

Example



The Micropython code for this block is

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(16, 40, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(axp.getTempInAXP192()))
    wait_ms(2)
```

Power P1 - AXP192

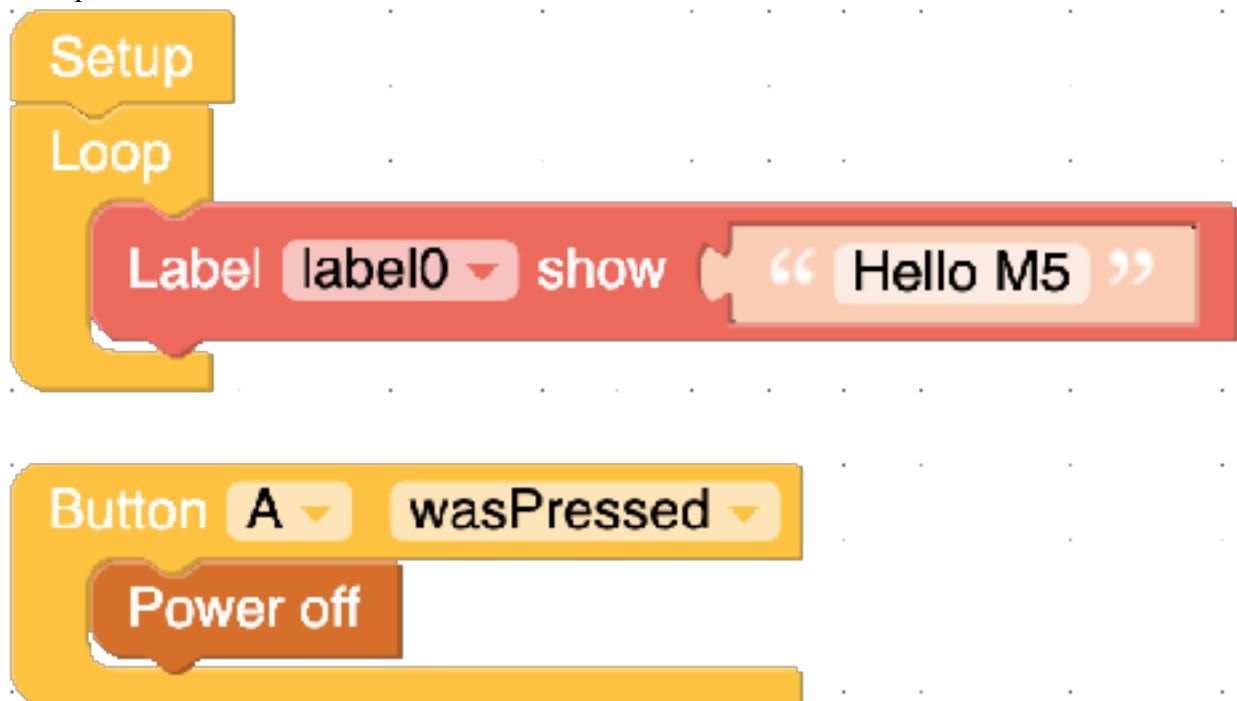
Power Off



Can be used to power down the AXP192 and the M5Stack or M5Stick Connected to it.
The MicroPython code for this block is

axp.powerOff()

Example.



This demo when run will show hello M5 on an M5StickC screen and then switch off when the button on the front is pressed.

To switch back on, you have to use the power/reset button on the side of the M5StickC.

Power P1 - AXP192

The Micropython code for this demo is shown below.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(20, 33, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

def buttonA_wasPressed():
    # global params
    axp.powerOff()
    pass
btnA.wasPressed(buttonA_wasPressed)

while True:
    label0.setText('Hello M5')
    wait_ms(2)
```

Power P1 - AXP192

The next two blocks are for advanced control of the AXP192's functions and not recommended for normal use.

Incorrect use of these two blocks could result in damage of your M5Stack or M5Stick.

Set Battery Charge Current

Set battery charge current to 100mA ▾

The Set Battery Charge Current block is used to control how much current is fed to the battery for charging. The current available options are 100mA, 190mA, 280mA, 360mA, 450mA, 550mA, 630mA and 700mA.

Again, I must mention that playing with these setting can be dangerous as the batteries used are lithium based and could explode or catch fire if incorrectly charged.

The Micropython code for this block is

```
axp.setChargeCurrent(axp.CURRENT_100MA)
```

Set LCD Voltage

Set LCD voltage to 2.4

The Set LCD Voltage block can be used to control the voltage that the LCD runs at. Voltage is used using a variable block or a mathematical value block. Setting the voltage lower will reduce the brightness but if set too low then the screen will stop working. Setting the voltage higher runs the risk of damaging the screen.

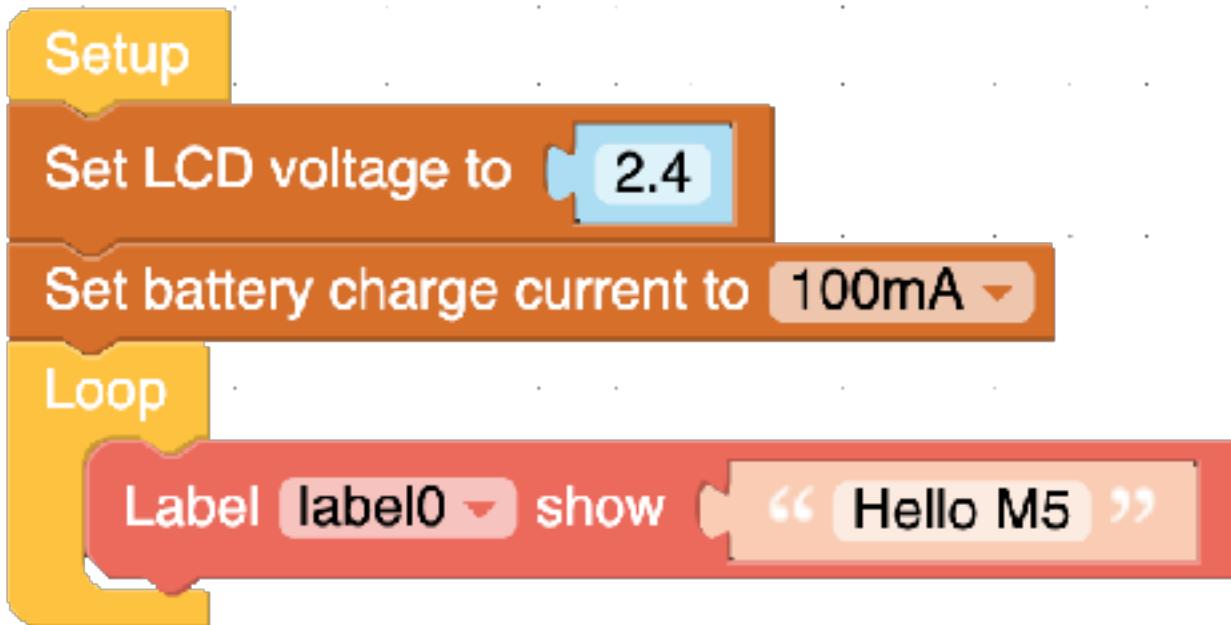
The Micropython code for this block is

```
axp.setLDO2Volt(0)
```

Power P1 - AXP192

Example

In this example I have used the blocks in the setup section of the program but the Set LCD Screen can be used in a loop if you know how to use it correctly.



The Micropython code for this demo is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(20, 33, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

axp.setLDO2Volt(2.4)
axp.setChargeCurrent(axp.CURRENT_100MA)
while True:
    label0.setText('Hello M5')
    wait_ms(2)
```

Power P2 - IP5306

The following power blocks are only available on certain M5Stack models that if the customised version of the IP5306 chip.

Is Charging

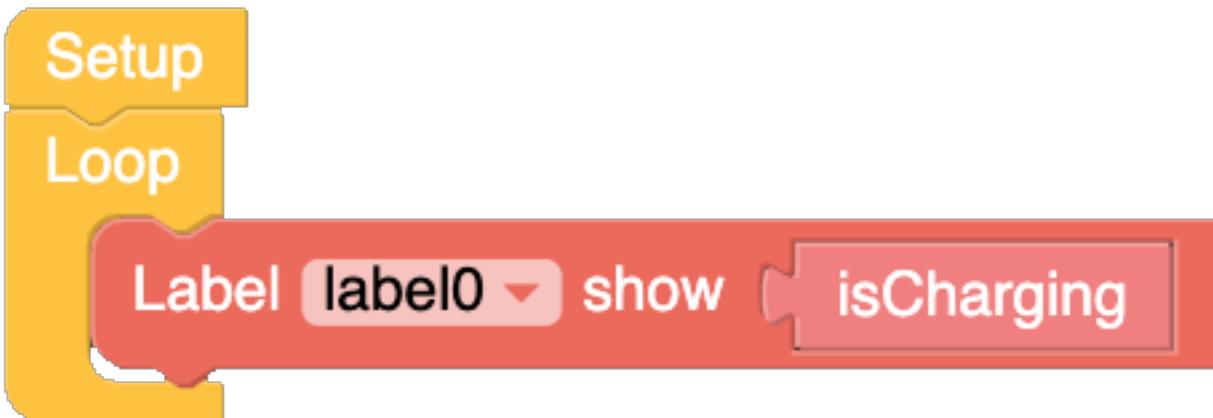


The Is Charging block is a value block which returns the status from the power management chip. The block reports false if there is no USB power connected or True if a USB cable is plugged in and connected to a supply.

The MicroPython code for this block is

```
power.isCharging()
```

Example



As with previous examples, here I am just using a label to show the status returned from the Is Charging Block.

The MicroPython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(108, 74, "Text", lcd.FONT_Default, 0xFFFF, rotate=0)

while True:
    label0.setText(str(power.isCharging()))
    wait_ms(2)
```

Is Charge Full

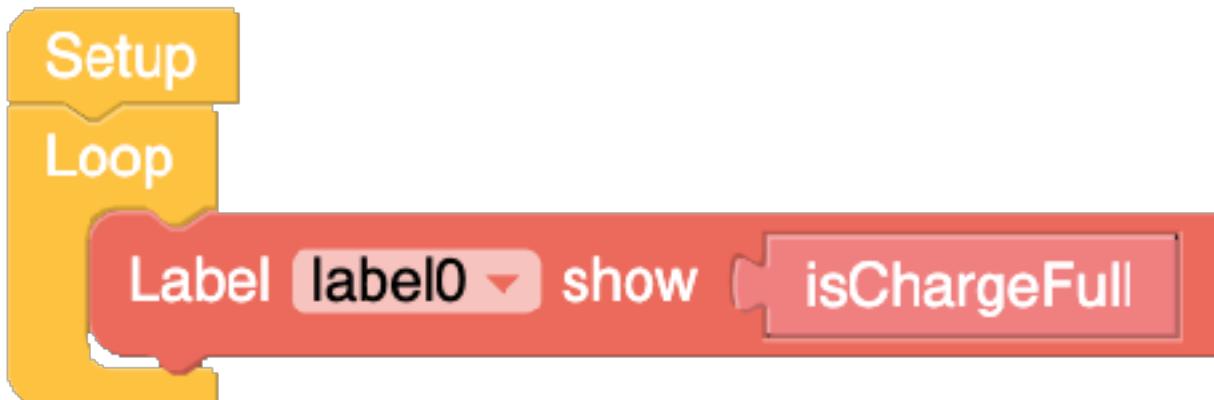


The Is Charge Full block is a value block which returns the status from the power management chip. The block reports false if there is the battery is not fully charged or True if the battery is fully charged.

The Micropython code for this block is

```
power.isChargeFull()
```

Example



As with previous examples, here I am just using a label to show the status returned from the Is Charge Full Block.

The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(108, 74, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(power.isChargeFull()))
    wait_ms(2)
```

Power P2 - IP5306

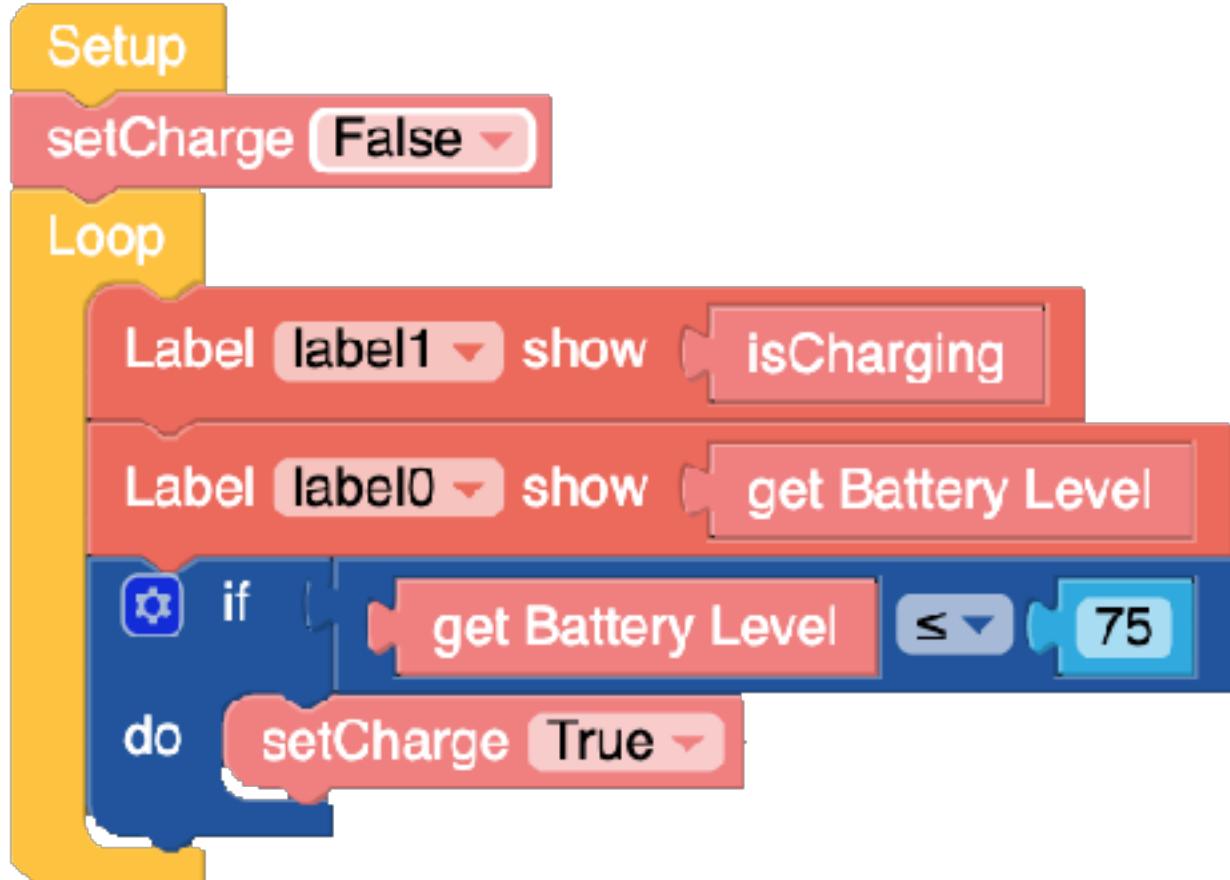
Set Charge



The Set Charge block allows you to control whether the battery is charging or not. Unlike the other blocks in this group, the Set Charge Block is a function block and not a value block. The MicroPython code for this block is

```
power.setCharge(True)
```

Example



In this example I have initially set the charging to false. In the loop the two labels check the status of the battery to see the charge level and charge status. I have then added a logic loop that checks if the battery is under 75 and if so set the battery charging. I have done this because my battery is flat as I forgot to charge it.

The MicroPython code for this example is as follows.

Power P2 - IP5306

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(108, 74, "Text", lcd.FONT_Default,0xFFFF, rotate=0)
label1 = M5TextBox(115, 38, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

power.setCharge(False)
while True:
    label1.setText(str(power.isCharging()))
    label0.setText(str(power.getBatteryLevel()))
    if (power.getBatteryLevel()) <= 75:
        power.setCharge(True)
    wait_ms(2)
```

Power P2 - IP5306

Get Battery Level



The Get Battery Lever block is a value block that returns the charge level of the battery as read by the IP5306.

The MicroPython code for this block is

```
power.getBatteryLevel()
```

Example



As with previous examples, here I am just using a label to show the status returned from the Get Battery Level Block.

The MicroPython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(108, 74, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

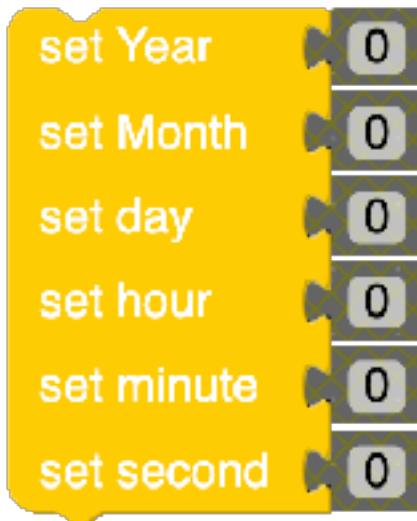
while True:
    label0.setText(str(power.getBatteryLevel()))
    wait_ms(2)
```

Real Time Clock

RTC - The Real Time Clock

At Present, the real time clock is only available to M5Sticks. The blocks in this section allow you to configure and make use of the RTC for time and date critical function.

Time and Date configuration block.

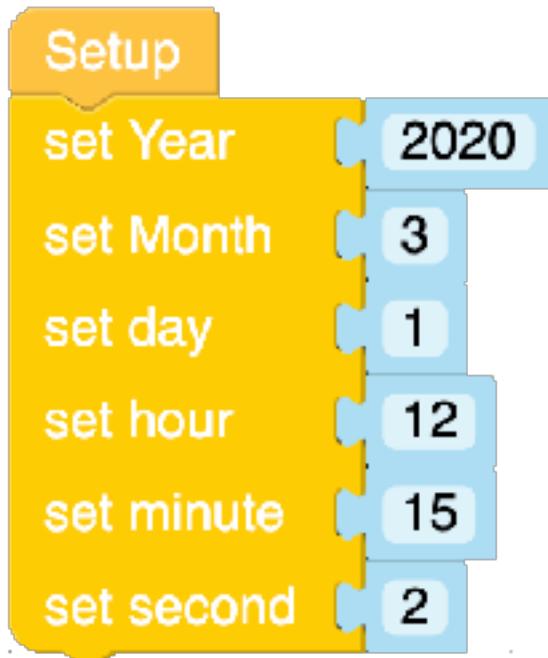


Used to configure the initial time and date settings for the internal clock. We can manually set the values by typing them in or we can use other blocks like variables.

The Micropython code for this block is

```
rtc.setTime(0, 0, 0, 0, 0, 0)
```

Example



To set the time and date options, you would configure this block and place it in the setup stage before the main loop.

The Micropython code for this example is as follows.

Real Time Clock

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

rtc.setTime(2020, 3, 1, 12, 15, 2)
```

The number in the brackets are from left to right and represent the values of the blocks from top to bottom.

This block doesn't show anything on the screen but sets these values in the code background. To view these values on screen, we need to use the following blocks.

So far there is no ability to receive the time and date from the global radio clock signal to automatically set time and date on the M5Stick or M5Stacks.

Real Time Clock

Get Year,

Year

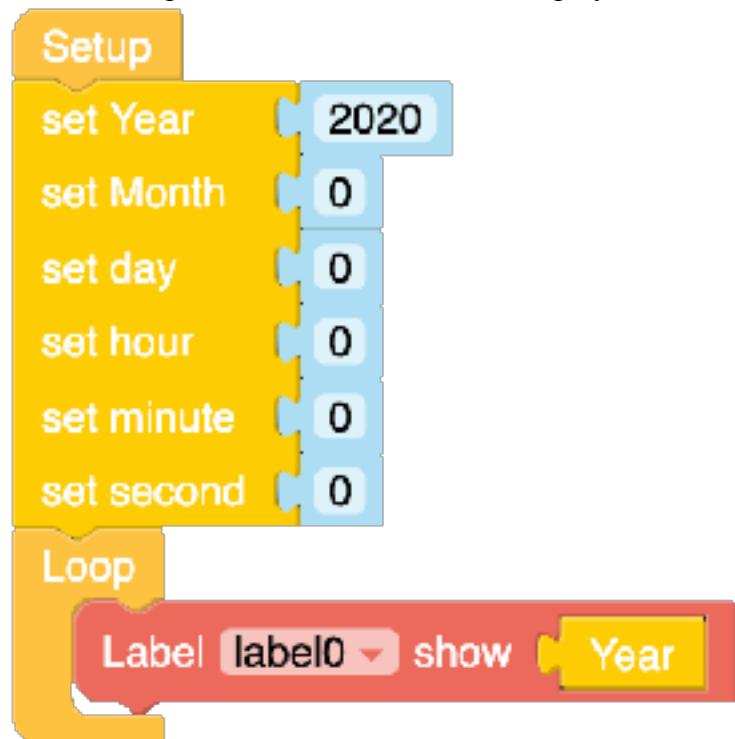
Gets the current year held in the real time clocks memory.

The micropython code for this block is

`rtc.now()[0]`

Example

In this example the code uses a label to display the date configured using the setup block.



The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(18, 53, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

rtc.setTime(2020, 3, 25, 14, 21, 0)
while True:
    label0.setText(str(rtc.now()))
    wait_ms(2)
```

Real Time Clock

Get Month,

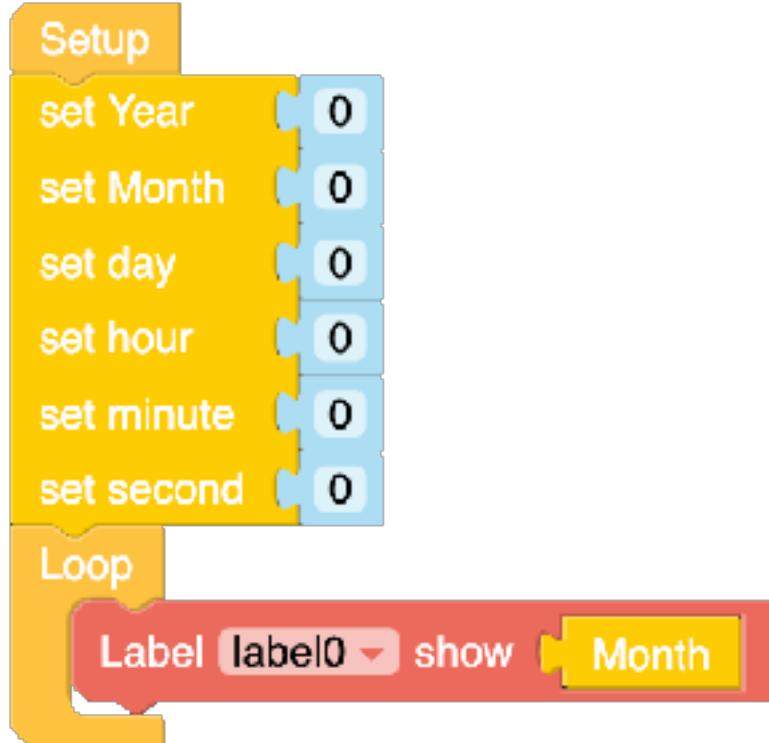
Month

Gets the month from the RTC.

The micropython code for this block is

`rtc.now()[1]`

Example



The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(22, 57, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

rtc.setTime(0, 0, 0, 0, 0, 0)
while True:
    label0.setText(str(rtc.now()[1]))
    wait_ms(2)
```

Real Time Clock

Get Day,

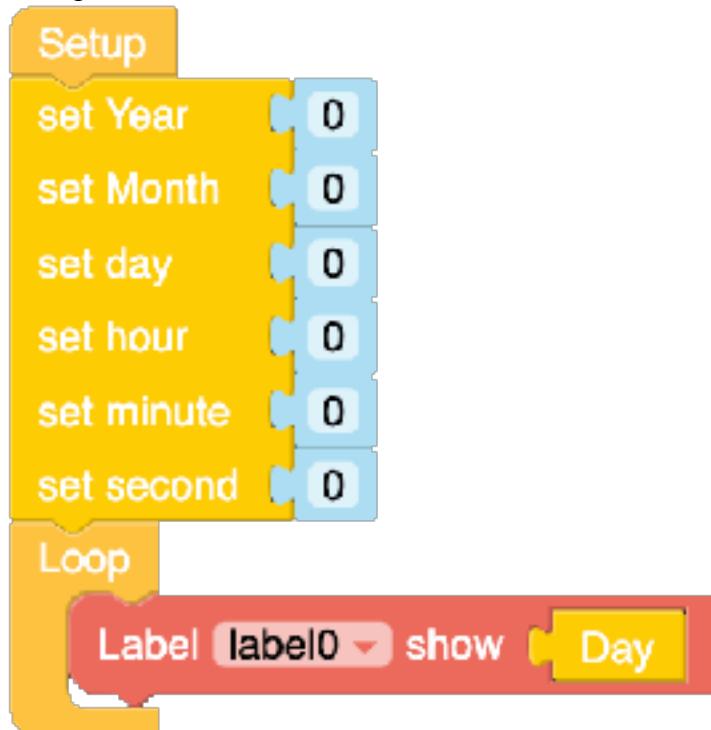
Day

Gets the day from the RTC.

The micropython code for this block is

`rtc.now()[2]`

Example



The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(22, 57, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

rtc.setTime(0, 0, 0, 0, 0, 0)
while True:
    label0.setText(str(rtc.now()[2]))
    wait_ms(2)
```

Real Time Clock

Get Hour,

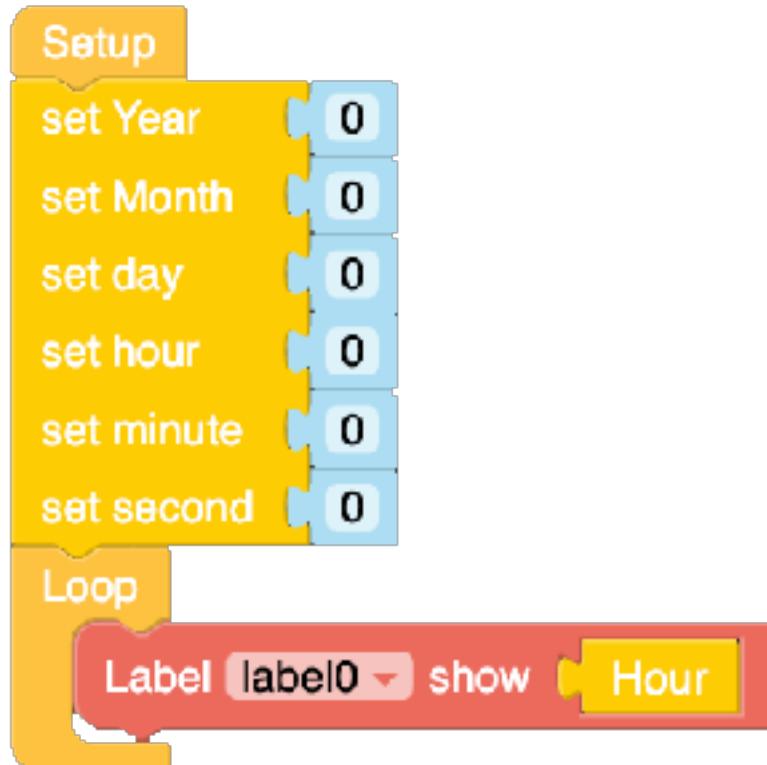


Gets the hour from the RTC.

The micropython code for this block is

`rtc.now()[3]`

Example



The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(22, 57, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

rtc.setTime(0, 0, 0, 0, 0, 0)
while True:
    label0.setText(str(rtc.now()[3]))
    wait_ms(2)
```

Real Time Clock

Get Minute,

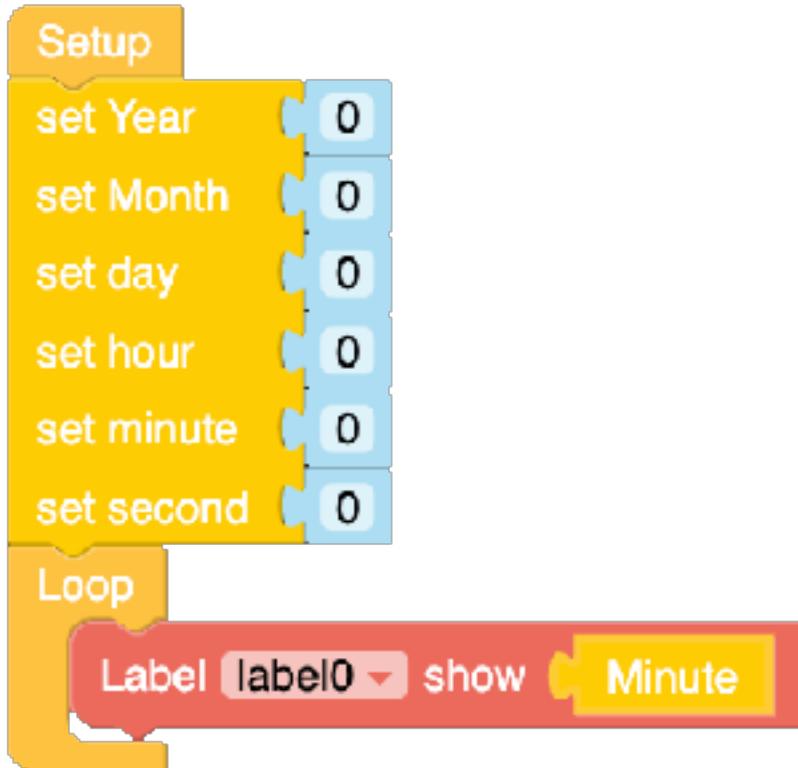


Gets the minute from the RTC.

The micropython code for this block is

`rtc.now()[4]`

Example



The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(22, 57, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

rtc.setTime(0, 0, 0, 0, 0, 0)
while True:
    label0.setText(str(rtc.now()[4]))
    wait_ms(2)
```

Real Time Clock

Get Second,

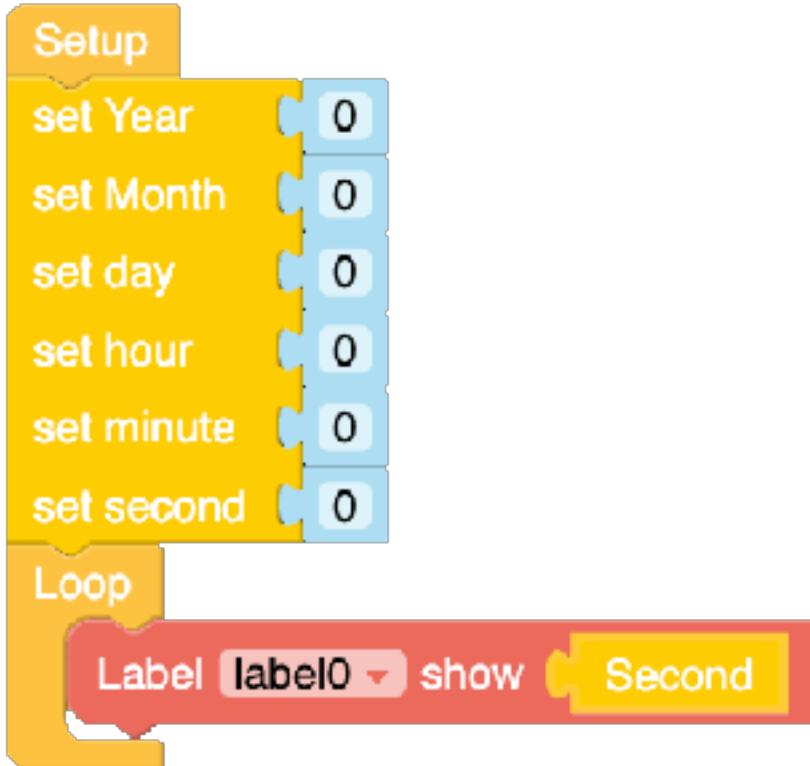
Second

Gets the seconds from the RTC.

The micropython code for this block is

`rtc.now()[5]`

Example



The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(22, 57, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

rtc.setTime(0, 0, 0, 0, 0, 0)
while True:
    label0.setText(str(rtc.now()[5]))
    wait_ms(2)
```

Real Time Clock

Get Local Time.

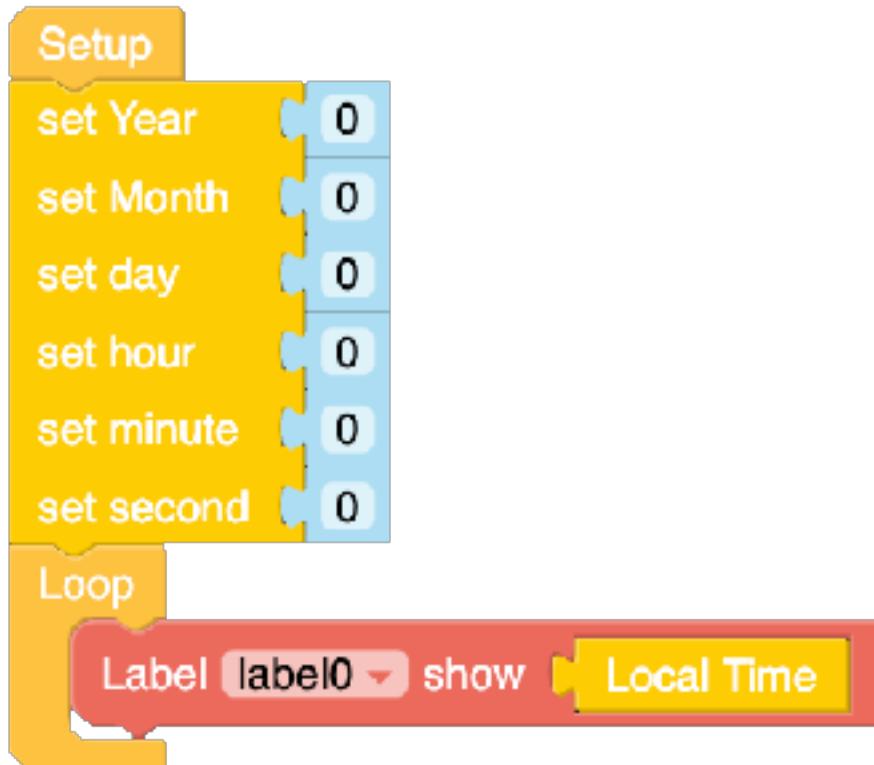
Local Time

Gets the local from the RTC.

The micropython code for this block is

`rtc.now()`

Example



The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)

label0 = M5TextBox(22, 57, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

rtc.setTime(0, 0, 0, 0, 0, 0)
while True:
    label0.setText(str(rtc.now()))
    wait_ms(2)
```

LED

LED

On the top left hand corner of the M5Stick C is a red led. To control this led you use the following blocks.

LED On



Turns the red led on the top of the M5Stick C on.

The Micropython code for this is,

M5Led.on()

LED Off



Turns the red led on the top of the M5Stick C off.

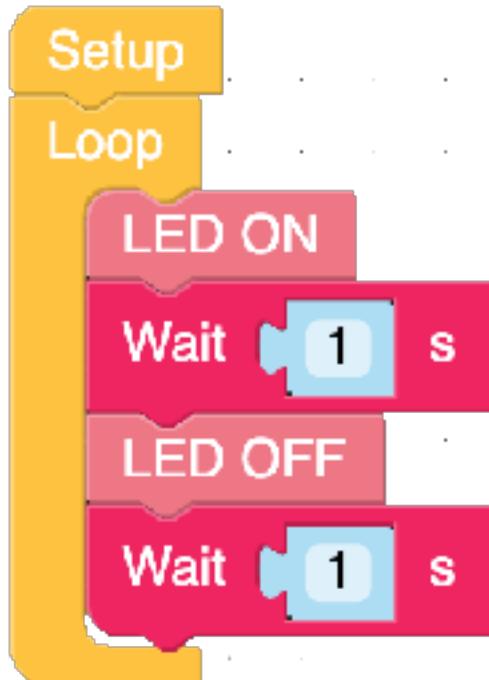
The Micropython code for this is,

M5Led.oFF()

LED

Example.

The following example turns the LED on for one second and then off for one second.



The Micropython code for this example is

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x111111)
while True:
    M5Led.on()
    wait(1)
    M5Led.off()
    wait(1)
    wait_ms(2)
```


HATS

First coined by the Raspberry Pi community, H.A.T. stands for Hardware Attached on Top and was a loose specification for solder less hardware add-ons that are connected on top of development boards. Due to the small size of the M5Sticks, M5Stack modules and only a limited number of units can be used. To get around the hardware limitations of the M5Sticks, a dedicated range of H.A.Ts have been developed to expand the capabilities of the M5Stick.

ENV Hat

ENV Hat

Get Pressure,

Get hat_env0 ▾ Pressure

Gets the current air pressure.

The Micropython code for this is,

`hat_env0.pressure`

Get Temperature,

Get hat_env0 ▾ Temperature

Gets the current temperature.

The Micropython code for this is,

`hat_env0.temperature`

Get Humidity,

Get hat_env0 ▾ Humidity

Gets the current Humidity.

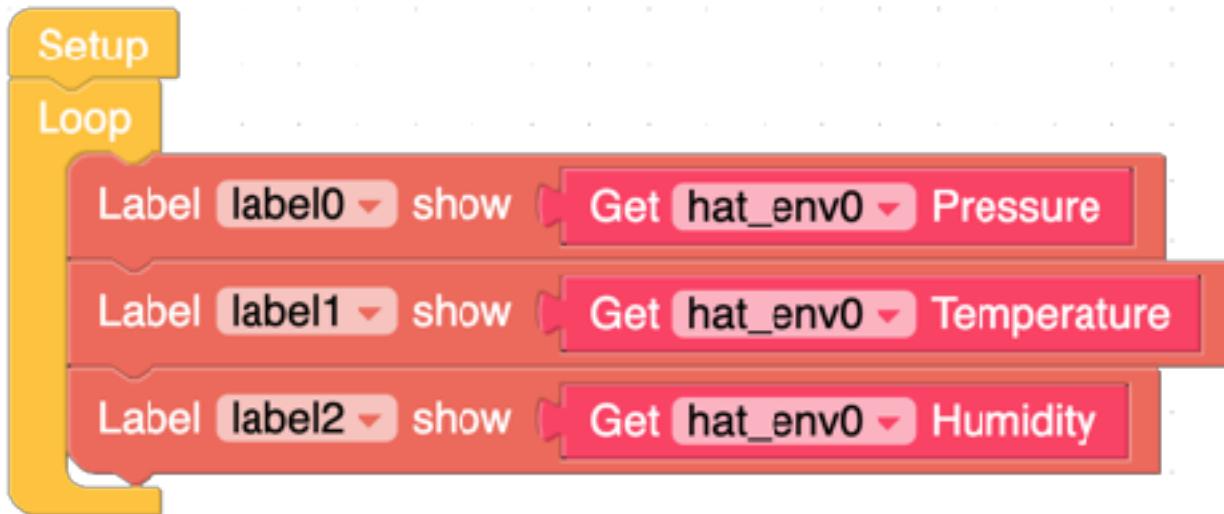
The Micropython code for this is,

`hat_env0.humidity`

ENV Hat

Example

In this example I am using three labels to show the Pressure, temperature and, humidity from the environmental sensor.



The Micropython code for this is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat
import hat

setScreenColor(0x111111)

hat_env0 = hat.get(hat.ENV)

label0 = M5TextBox(4, 43, "Text", lcd.FONT_Default,0xFFFF, rotate=0)
label1 = M5TextBox(3, 88, "Text", lcd.FONT_Default,0xFFFF, rotate=0)
label2 = M5TextBox(1, 124, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(hat_env0.pressure))
    label1.setText(str(hat_env0.temperature))
    label2.setText(str(hat_env0.humidity))
    wait_ms(2)
```

PIR Hat

PIR Hat,
Get hat_pir Status,

Get hat_pir0 status

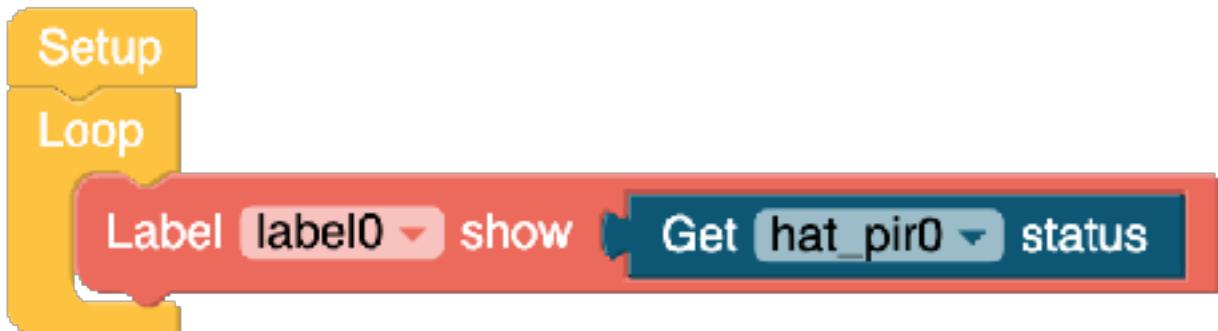
Returns True or False if the PIR detects anything.

The MicroPython code for this block is.

hat_pir0.state

Example

In the following example I am using a label to show the status of the P.I.R on the Stick C's screen.



The MicroPython code for this example is as follows:

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat
import hat

setScreenColor(0x111111)

hat_pir0 = hat.get(hat.PIR)

label0 = M5TextBox(4, 43, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(hat_pir0.state))
    wait_ms(2)
```

Speaker Hat

Speaker Hat Hat_spk0 Play Tone

hat_spk0 play tone Low A for 1 beat

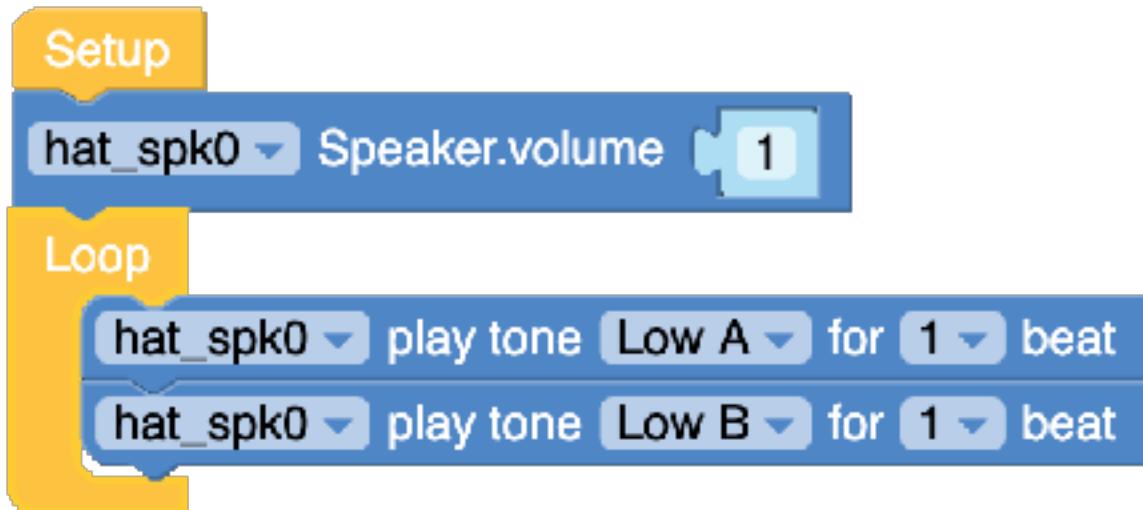
The Play Tone block allows users to play musical notes and set the duration in beats or fractions of Beats.

The Micropython code for this block is.

hat_spk0.sing(220, 1)

Example

The following demo is an example of the two tone alarms often found on emergency vehicles.



The Micropython code for this demo is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat
import hat

setScreenColor(0x111111)

hat_spk0 = hat.get(hat.SPEAKER)

hat_spk0.setVolume(1)
while True:
    hat_spk0.sing(220, 1)
    hat_spk0.sing(247, 1)
    wait_ms(2)
```

Speaker Hat

Hat_spk0 Speaker.beep



hat_spk0 [Speaker.beep freq: 1800 duration: 200 ms]

The hat_spk0 speaker Beep block allows us to make sounds by setting the frequency of the sound and the duration of the sound.

The Micropython code for this block is.



```
hat_spk0.tone(1600, 200)
```

Example

Hat_spk0 Speaker Volume



hat_spk0 [Speaker.volume 1]

The hat_spk0 Speaker volume allows us to set the volume of the speaker sounds using a set value or by using an input device like the angle unit.



```
Setup: hat_spk0 [Speaker.volume v1] v1
Loop: Get angle0 [value] v1
```

The Micropython code for this block is.



```
hat_spk0.setVolume(1)
```

Example

NCIR HAT NCIR Read



Returns a value from the N.C.I.R Hat Sensor.

The MicroPython code for this block is.

ncir0.temperature

Example

In this example I have added a label that gets its text value from the NCIR unit.



The MicroPython code for this example is

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x222222)
ncir0 = unit.get(unit.NCIR, unit.PORTA)

label0 = M5TextBox(132, 61, "Text", lcd.FONT_Default, 0xFFFF, rotate=0)

while True:
    label0.setText(str(ncir0.temperature))
    wait_ms(2)
```

DAC Hat

DAC hat_dac0 Output Voltage

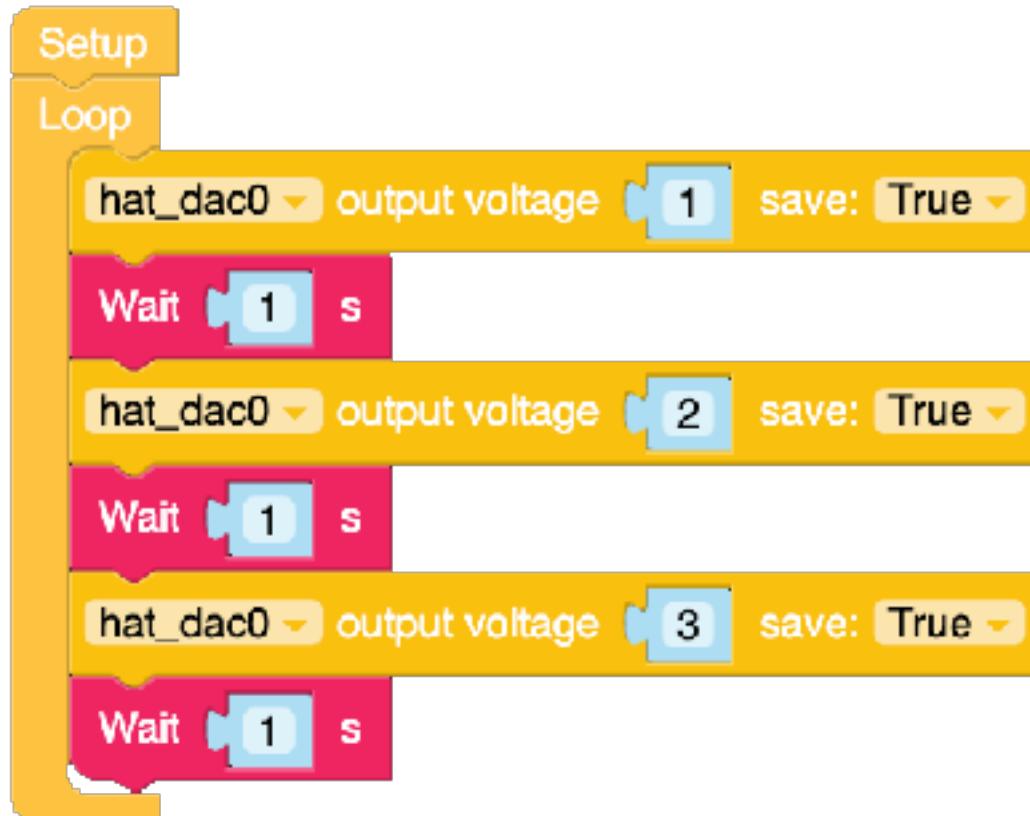
```
hat_dac0 output voltage 1 save: True
```

Creates an output voltage from 0 to 3.3 volts using the mathematical blocks where the number shown is the actual reading. If save is set to true then the data will be written to the M5Sticks E.E.P.R.O.M.

The MicroPython code for this block is.

```
hat_dac0.setVoltage(1,save=True)
```

Example



This simple example sets the output voltage between 1 volt and 3 volts.

The MicroPython code for this example is as follows.

DAC Hat

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat
import hat

setScreenColor(0x111111)

hat_dac0 = hat.get(hat.DAC)

while True:
    hat_dac0.setVoltage(1,save=True)
    wait(1)
    hat_dac0.setVoltage(2,save=True)
    wait(1)
    hat_dac0.setVoltage(3,save=True)
    wait(1)
    wait_ms(2)
```

DAC Hat

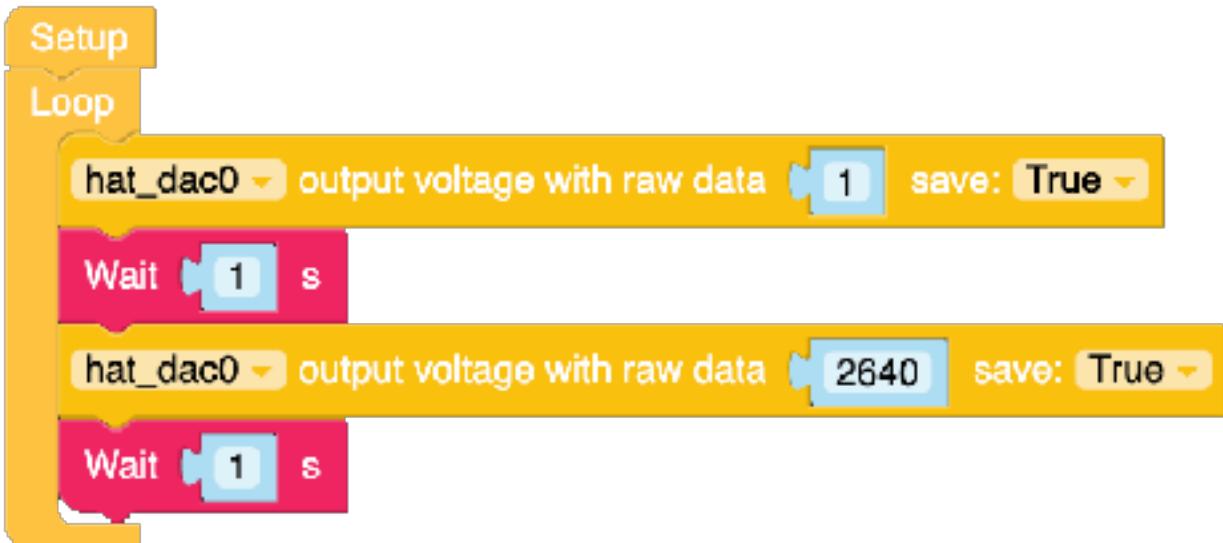
hat_dac0 Output Voltage with Raw Data.

hat_dac0 output voltage with raw data **1** save: **True**

Creates a raw data output of 0 to 4096 with 2460 being close to the 3.3v max voltage. If save it set to true then the data will be written to the M5Sticks E.E.P.R.O.M.
The Micropython code for this block is.

hat_dac0.writeData(1,save=True)

Example



The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat
import hat

setScreenColor(0x111111)

hat_dac0 = hat.get(hat.DAC)

while True:
    hat_dac0.writeData(1,save=True)
    wait(1)
    hat_dac0.writeData(2640,save=True)
    wait(1)
    wait_ms(2)
```

ADC Hat

ADC

read hat_adc0 voltage

Returns an analogue reading from the Analogue to Digital Converter Hat.
The MicroPython code for this block is.

hat_adc0.voltage

Example

In this example I am using a label to display the voltage measured from the ADC.



The MicroPython code for the sensor is

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat
import hat

setScreenColor(0x111111)

hat_adc0 = hat.get(hat.ADC)

label0 = M5TextBox(31, 63, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(hat_adc0.voltage))
    wait_ms(2)
```

Servo Hat

Hat Servo

Rotate Servo to Degree.

Set hat_servo0 ▾ rotate to degree 0

Rotates the servo to a position defined with a maths number block.
The MicroPython code for this block is.

`hat_servo0.write_angle(0)`

Example



This demo sweeps the arm of the servo hat from 0 to 180 degrees and back again, pausing for a second between moves.

Servo Hat

The Micropython code for this is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_servo0 = hat.get(hat.SERVO)

while True:
    hat_servo0.write_angle(0)
    wait(1)
    hat_servo0.write_angle(90)
    wait(1)
    hat_servo0.write_angle(180)
    wait(1)
    hat_servo0.write_angle(90)
    wait(1)
    hat_servo0.write_angle(0)
    wait_ms(2)
```

Servo Hat

Write Milliseconds

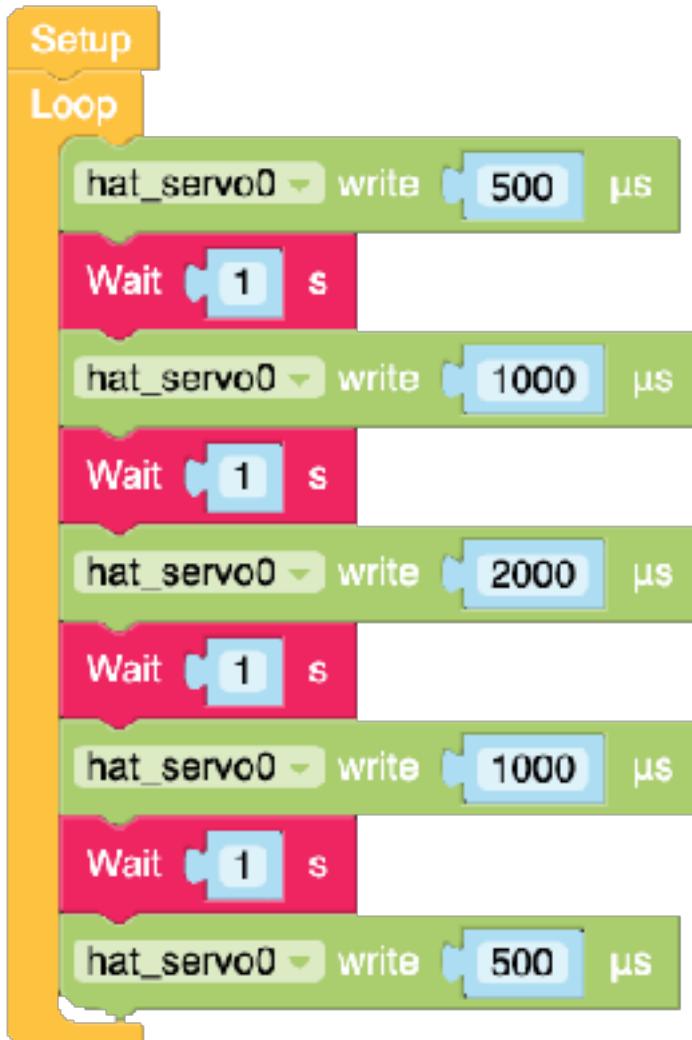


Not all servos are made the same and there are times where the servos rotation position will not match the angle set in the "Rotate to Degree" block. To fine tune this position we can use this block as it sets the servo position in microseconds with a maths number block. In order to find the position in milliseconds we will need to look at the individual servo specifications and data sheets for the numbers. The valid range is from 500 to 2500 milliseconds.

The MicroPython code for this block is.

```
hat_servo0.write_us(600)
```

Example



This example moves the servo horn by setting how long the PWM signal is high. The MicroPython code for this example is as follows.

Servo Hat

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_servo0 = hat.get(hat.SERVO)

while True:
    hat_servo0.write_us(500)
    wait(1)
    hat_servo0.write_us(1000)
    wait(1)
    hat_servo0.write_us(2000)
    wait(1)
    hat_servo0.write_us(1000)
    wait(1)
    hat_servo0.write_us(500)
    wait_ms(2)
```

Servo8 Hat

Servo8 Hat.

The Servo8 Hat is a robot chassis that allows you to control up to eight servos at the same time.

Set (num) Servo Rotate



The Set Servo Rotate block allows individual control of the eight servo ports on the board. The servo port is selected by clicking the arrow and deleting the port number. The position value can be set between 0 and 180 using a maths value block, a variable block or a sensor value block.

The MicroPython code for this block is.

```
hat_servos0.SetAngle(1, 0)
```

The first number in brackets is the port number and the second number is the position.

Example

The example on the next page set the first three servos to 90 degrees then back to 0 degrees.

The MicroPython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_servos0 = hat.get(hat.SERVOS)

while True:
    hat_servos0.SetAngle(1, 90)
    wait_ms(1)
    hat_servos0.SetAngle(2, 90)
    wait_ms(1)
    hat_servos0.SetAngle(3, 90)
    wait_ms(1)
    hat_servos0.SetAngle(1, 0)
    wait_ms(1)
    hat_servos0.SetAngle(2, 0)
    wait_ms(1)
    hat_servos0.SetAngle(3, 0)
    wait_ms(1)
    wait_ms(2)
```

Servo8 Hat

The script consists of a **Setup** block and a **Loop** block. The **Setup** block contains a **Set Servos** block. The **Loop** block contains three nested loops. Each nested loop has a **Set Servos** block followed by a **Wait** block.

- Outer Loop:
 - Set 1 servos rotate 90**
 - Wait 1 ms**
- Middle Loop:
 - Set 2 servos rotate 90**
 - Wait 1 ms**
- Inner Loop:
 - Set 3 servos rotate 90**
 - Wait 1 ms**

After the inner loop, the middle loop's **Set Servos** block is executed again, followed by another **Wait 1 ms**. This pattern repeats until the outer loop is completed.

Servo8 Hat

Set Servo Millisecond

Set 1 servos write 600 μ s

As stated in previous servo examples, the Set Servo Millisecond block is used to control servos that don't have a 180 degree movement range.

The MicroPython code for this block is.

```
hat_servos0.SetPulse(1, 600)
```

Example



Servo8 Hat

The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_servos0 = hat.get(hat.SERVOS)

while True:
    hat_servos0.SetPulse(1, 1800)
    wait_ms(1)
    hat_servos0.SetPulse(1, 600)
    wait_ms(1)
    wait_ms(2)
```

Servo8 Hat

The following two blocks provide two different ways to change the colour of the RGB light on the chassis. The first box is a basic colour setting function where as the second is for those who revere to deal with the separate colour values which gives more colour options. Which ever function is used, the resulted code generated by UIflow results in a hexadecimal colour value.

Set RGB Bar Colour

Set RGB Bar color

A red rounded rectangular block with the text "Set RGB Bar color" in white. To the right of the text is a small red square icon.

The Set RGB Bar Colour is used to set the colour of the RGB LED on the front of the Servo8 module using the colour picker dialog. for more information on the colour picker dialog see the Graphical section.

The Micropython code for this block is.

```
hat_servos0.SetRGB(0xff0000)
```

Example

Set RGB Bar Colour (R,G,B,)

Set RGB Bar color R 0 G 0 B 0

A red rounded rectangular block with the text "Set RGB Bar color R" in white. To the right of the text are three grey boxes containing the letters "0", "G", and "B".

The Set RGB Bar Colour sets the colour using separate values for the Red, Green and blue channels using mathematical values.

The Micropython code for this block is.

```
hat_servos0.SetRGB(0x000000)
```

Example

Servo8 Hat

This Scratch script uses the Servo8 Hat extension. It begins with a yellow **Setup** block, followed by a yellow **Loop** block. Inside the loop, there are three **Set RGB Bar color** blocks: one red (red R, green G, blue B), one green (red R, green 255, blue B), and one blue (red 0, green 0, blue 255). Each color set is preceded by a **Wait** block for 1 second.

```
when green flag is shown
  [Setup v]
end
[Loop v]
  [Set RGB Bar color [red v] [green v] [blue v]]
  [Wait [1] sec]
  [Set RGB Bar color [green v] [255] [blue v]]
  [Wait [1] sec]
  [Set RGB Bar color [red v] [0] [blue 255]]
  [Wait [1] sec]
end
```

This Scratch script uses the Servo8 Hat extension. It begins with a yellow **Setup** block, followed by a yellow **Loop** block. Inside the loop, there are three **Set RGB Bar color** blocks setting the bar to red (R 255, G 0, B 0), green (R 0, G 255, B 0), and blue (R 0, G 0, B 255) respectively. Each color set is preceded by a **Wait** block for 1 second.

```
when green flag is shown
  [Setup v]
end
[Loop v]
  [Set RGB Bar color [R 255 G 0 B 0] v]
  [Wait [1] sec]
  [Set RGB Bar color [R 0 G 255 B 0] v]
  [Wait [1] sec]
  [Set RGB Bar color [R 0 G 0 B 255] v]
  [Wait [1] sec]
end
```

Servo8 Hat

The Micropython code for both block examples are exactly the same.

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_servos0 = hat.get(hat.SERVOS)

while True:
    hat_servos0.SetRGB(0xff0000)
    wait(1)
    hat_servos0.SetRGB(0x00ff00)
    wait(1)
    hat_servos0.SetRGB(0x0000ff)
    wait(1)
    wait_ms(2)
```

PuppyC Hat

Puppy C

The Puppy C chassis is a walking robot chassis designed for the M5StickC and comes with four SG90 servos and a 16340 battery.

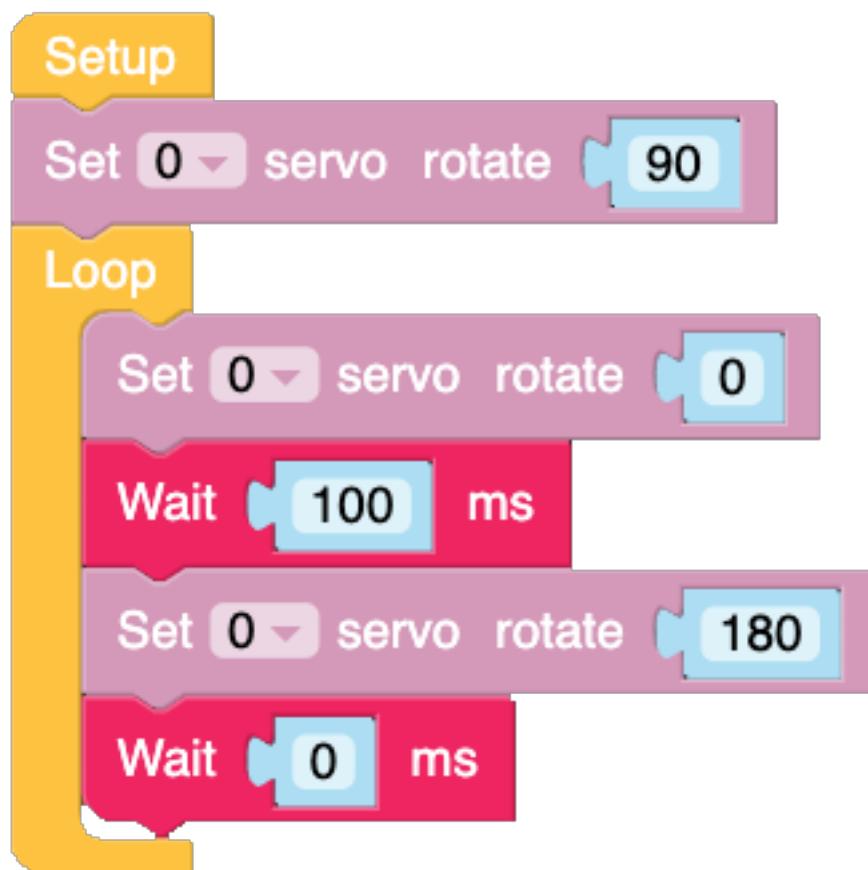
Set Servo position



The MicroPython code for this block is.

```
hat_puppy0.SetAngle(0, 0)
```

Example



This example moves one leg back and forth through 180 degrees.

PuppyC Hat

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_puppy0 = hat.get(hat.PUPPY)

hat_puppy0.SetAngle(0, 90)
while True:
    hat_puppy0.SetAngle(0, 0)
    wait_ms(100)
    hat_puppy0.SetAngle(0, 180)
    wait_ms(0)
    wait_ms(2)
```

PuppyC Hat

Set servo position 0 legs 0, 1, 2, 3

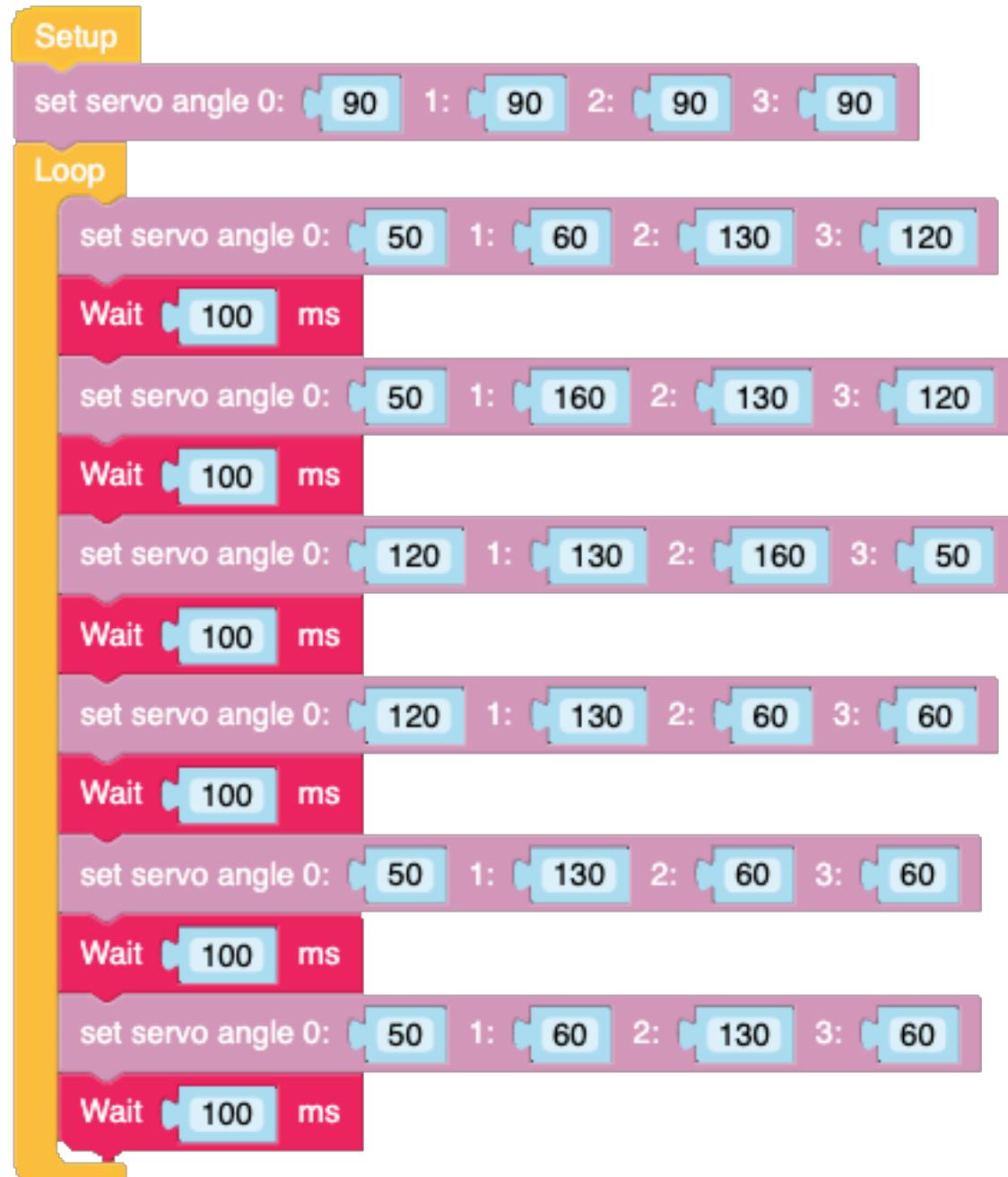
set servo angle 0: 0 1: 0 2: 0 3: 0

The Micropython code for this block is.

```
hat_puppy0.SetAllAngle(, , )
```

Example

This example is based on the demo code found on the M5Stack web site.



PuppyC Hat

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_puppy0 = hat.get(hat.PUPPY)

hat_puppy0.SetAllAngle(90, 90, 90, 90)
while True:
    hat_puppy0.SetAllAngle(50, 60, 130, 120)
    wait_ms(100)
    hat_puppy0.SetAllAngle(50, 160, 130, 120)
    wait_ms(100)
    hat_puppy0.SetAllAngle(120, 130, 160, 50)
    wait_ms(100)
    hat_puppy0.SetAllAngle(120, 130, 60, 60)
    wait_ms(100)
    hat_puppy0.SetAllAngle(50, 130, 60, 60)
    wait_ms(100)
    hat_puppy0.SetAllAngle(50, 60, 130, 60)
    wait_ms(100)
    wait_ms(2)
```

JoystickC

Joystick

JoystickC is a single Joystick Hat that plugs into the connector on the top of the M5StickC and can be used for simple remote control of other M5Stack devices. The Joysticks blocks are value blocks meaning that they need other blocks in order to be used.

Get hat_Joystick0 X

 Get [hat_Joystick0 ▾] X

hat_Joystick0.X

Get hat_Joystick0 Y

 Get [hat_Joystick0 ▾] Y

hat_Joystick0.Y

Get hat_Joystick0 is pressed.

 Get [hat_Joystick0 ▾] is pressed

hat_Joystick0.Press

Get hat_Joystick0 Reverse X

 Get [hat_Joystick0 ▾] Reverse X

hat_Joystick0.InvertX

Get hat_Joystick0 Reverse Y

 Get [hat_Joystick0 ▾] Reverse Y

hat_Joystick0.InvertY

BugC

Finger

BeetleC

YUN

JoyC

RoverC

RoverC



The RoverC is a mobile omnidirectional chassis designed to connect to the M5StickC using the eight pin HAT connector. The Chassis communicates with the M5StickC over I2C and is controlled by its own STM32F030F4 controller.

The chassis features four independently controlled N20 motors connected to Mecanum wheels which can provide up to twelve different modes of movement, a 750mah 16340 battery to provide additional power for the motors, two I2C ports, four M3 screw points, and six mounting holes for adding LEGO compatible parts.

RoverC

UIFlow Blocks.

RoverC

When programming the RoverC in UIFlow there are only three blocks provided to control the chassis. These three blocks are as follows.

Set Wheel Pulse

Set the front-left wheel pulse to 0

A UIFlow block labeled "Set the front-left wheel pulse to 0". The "front-left" part is highlighted in red. The "wheel pulse to" part is in blue. There is a green input port with a value of 0.

Set Wheel Pulse can be used to set a pulse length between 20 and 100 using a maths block or with a variable to only one wheel turning the wheel in a clockwise rotation. Sending a pulse length between -20 and -100 will result in the wheel turning in an anticlockwise rotation. Sending a pulse to only one wheel will result in a normal turning action.

The Micropython code for this block is as follows.

```
hat_roverc0.SetPulse(0, 0)
```

Set Wheel Pulse (four wheels)

Set wheel pulse front-left 0 front-right 0 rear-left 0 rear-right 0

A UIFlow block labeled "Set wheel pulse front-left 0 front-right 0 rear-left 0 rear-right 0". Each wheel has its own green input port with a value of 0.

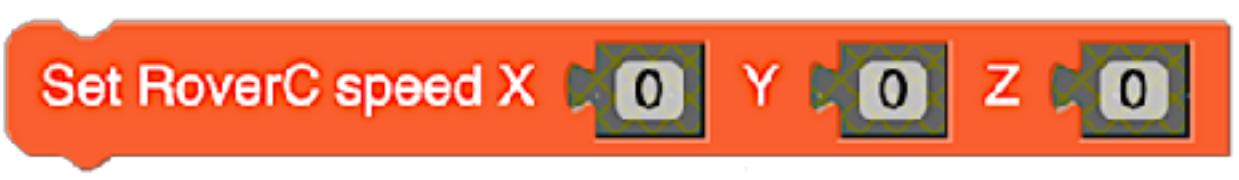
Set Wheel Pulse can be used to control all four wheels at once which give a greater range of movement including turning on the spot and sliding to a direction. As with the previous block, the pulse length can be set between 20 and 100 using a maths block or with a variable turning the wheel in a clockwise rotation. Sending a pulse length between -20 and -100 will result in the wheel turning in an anticlockwise rotation.

The Micropython code for this block is as follows.

```
hat_roverc0.SetAllPulse(0, 0, 0, 0)
```

RoverC

Set RoverC Speed



Set RoverC speed X [0] Y [0] Z [0]

The Set RoverC Speed block is used to control the speed that the RoverC move at. The speed length is between 20 and 100 using a maths block or with a variable.

The MicroPython code for this block is as follows.

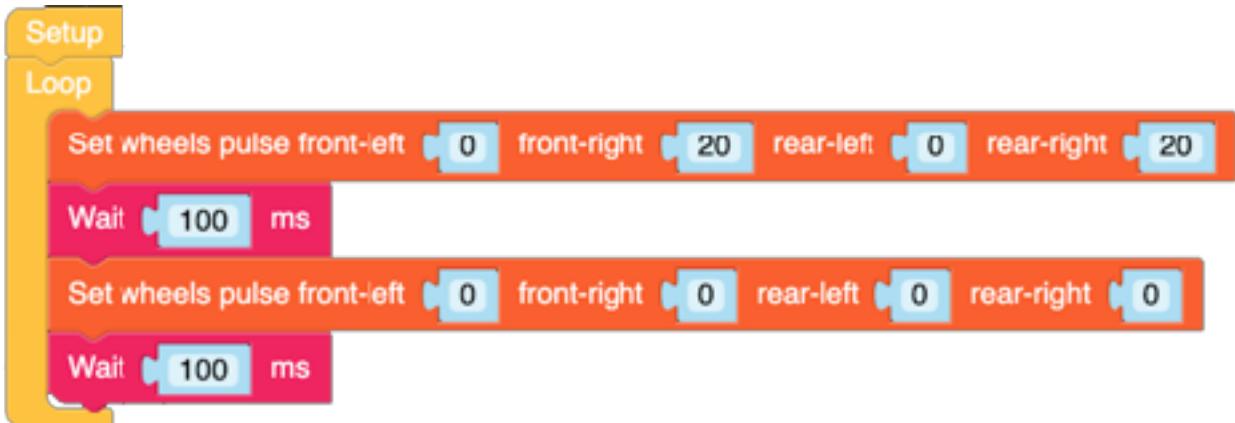
```
hat_roverc0.SetSpeed(0, 0, 0)
```

Movement

Because the RoverC uses Mecanum wheels instead of normal wheels movement needs to be programmed in what would to some look like a peculiar way.

Turn Left.

To make the RoverC turn left or anticlockwise you would use the code blocks as follows.



RoverC

The Micropython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_roverc0 = hat.get(hat.ROVERC)

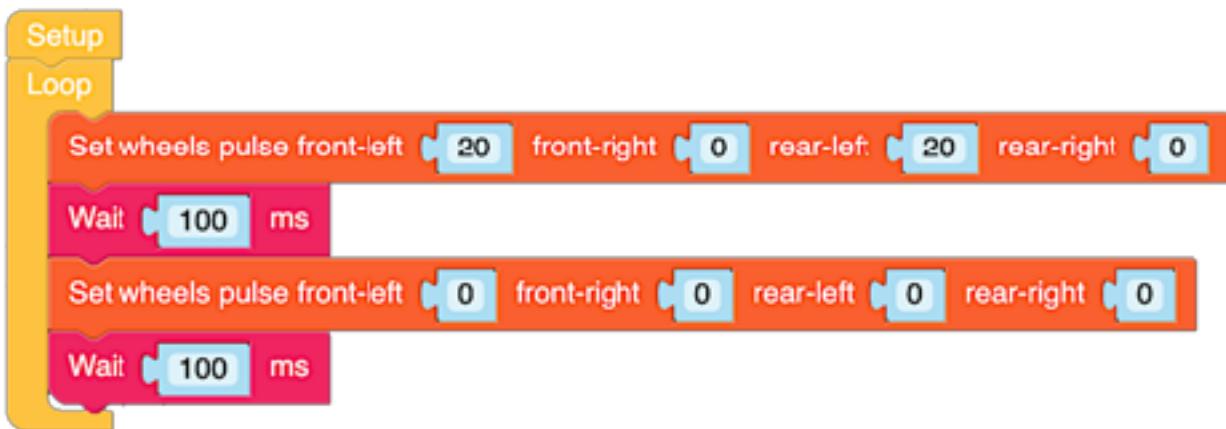
while True:
    hat_roverc0.SetAllPulse(0, 20, 0, 20)
    wait_ms(100)
    hat_roverc0.SetAllPulse(0, 0, 0, 0)
    wait_ms(100)
    wait_ms(2)
```

In this example I send a value of 20 to both the right hand motors and then after 100ms send a value of 0 to stop the motors.

Please note: If the motors do not receive instructions to se the motors to 0 before being switched off, the RoverC will start moving as soon as it is switched on again using the onboard power switch.

RoverC

Turn Right



To make the RoverC turn right or clockwise you send value to the left hand motors.

In this example I send a value of 20 to both the left hand motors and then after 100ms send a value of 0 to stop the motors.

The MicroPython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_roverc0 = hat.get(hat.ROVERC)

while True:
    hat_roverc0.SetAllPulse(20, 0, 20, 0)
    wait_ms(100)
    hat_roverc0.SetAllPulse(0, 0, 0, 0)
    wait_ms(100)
    wait_ms(2)
```

RoverC

Move Forwards



In order to make the RoverC drive forwards all we need to do is set the same value to all four wheels at the same time.

The Micropython code for this is.

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_roverc0 = hat.get(hat.ROVERC)

while True:
    hat_roverc0.SetAllPulse(20, 20, 20, 20)
    wait_ms(100)
    hat_roverc0.SetAllPulse(0, 0, 0, 0)
    wait_ms(100)
    wait_ms(2)
```

RoverC

Move Backwards



To make the RoverC reverse we just need to send a minus value to all four motors at once. The Micropython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

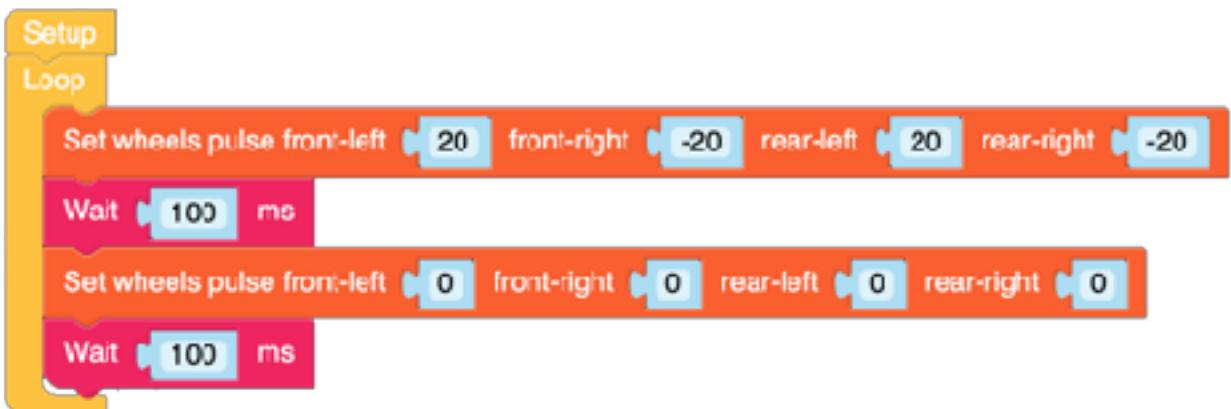
setScreenColor(0x111111)

hat_roverc0 = hat.get(hat.ROVERC)

while True:
    hat_roverc0.SetAllPulse(-20, -20, -20, -20)
    wait_ms(100)
    hat_roverc0.SetAllPulse(0, 0, 0, 0)
    wait_ms(100)
    wait_ms(2)
```

RoverC

Rotate Clockwise



Because the RoverC has four independently motorised wheels the RoverC can rotate on the spot like a tracked vehicle. In order to make the RoverC Rotate clockwise we set the left wheels with a positive value and the right wheels with a negative value. Rotate AntiClockwise The Micropython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

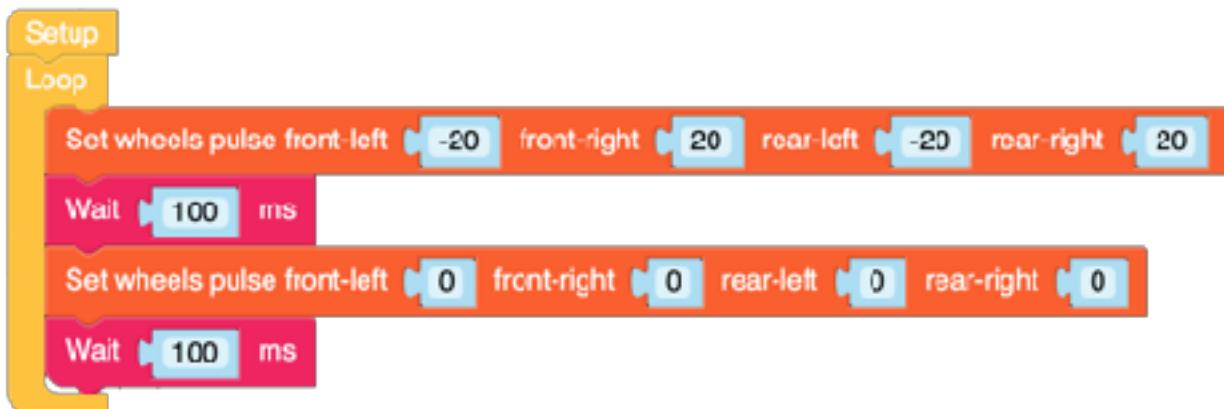
setScreenColor(0x111111)

hat_roverc0 = hat.get(hat.ROVERC)

while True:
    hat_roverc0.SetAllPulse(-20, 20, -20, 20)
    wait_ms(100)
    hat_roverc0.SetAllPulse(0, 0, 0, 0)
    wait_ms(100)
    wait_ms(2)
```

RoverC

Rotate Anti-Clockwise



The Micropython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_roverc0 = hat.get(hat.ROVERC)

while True:
    hat_roverc0.SetAllPulse(20, -20, 20, -20)
    wait_ms(100)
    hat_roverc0.SetAllPulse(0, 0, 0, 0)
    wait_ms(100)
    wait_ms(2)
```

RoverC

As well as rotating clockwise or anticlockwise, the Mecanum wheels also allow the RoverC to Slide around while still facing the same direction.

Slide Left



The MicroPython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

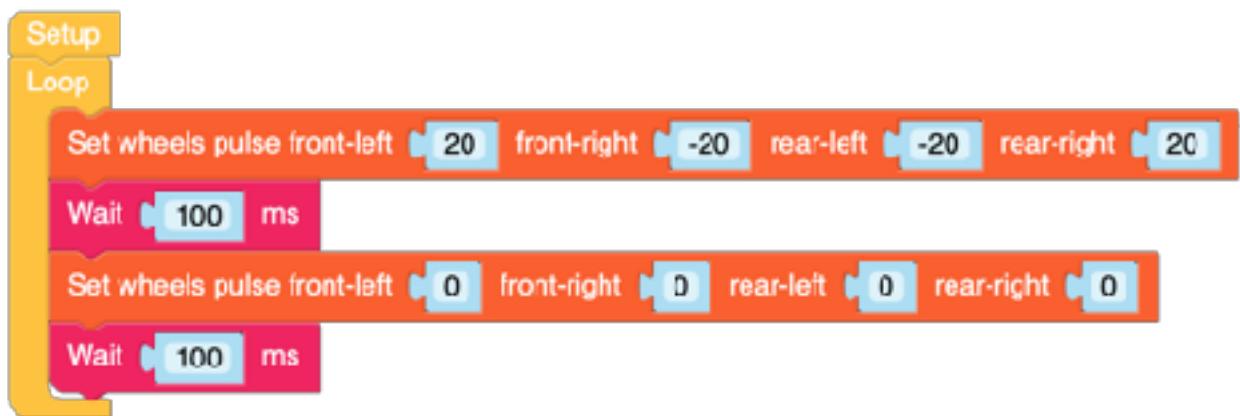
setScreenColor(0x111111)

hat_roverc0 = hat.get(hat.ROVERC)

while True:
    hat_roverc0.SetAllPulse(-20, 20, 20, -20)
    wait_ms(100)
    hat_roverc0.SetAllPulse(0, 0, 0, 0)
    wait_ms(100)
    wait_ms(2)
```

RoverC

Slide Right



The Micropython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

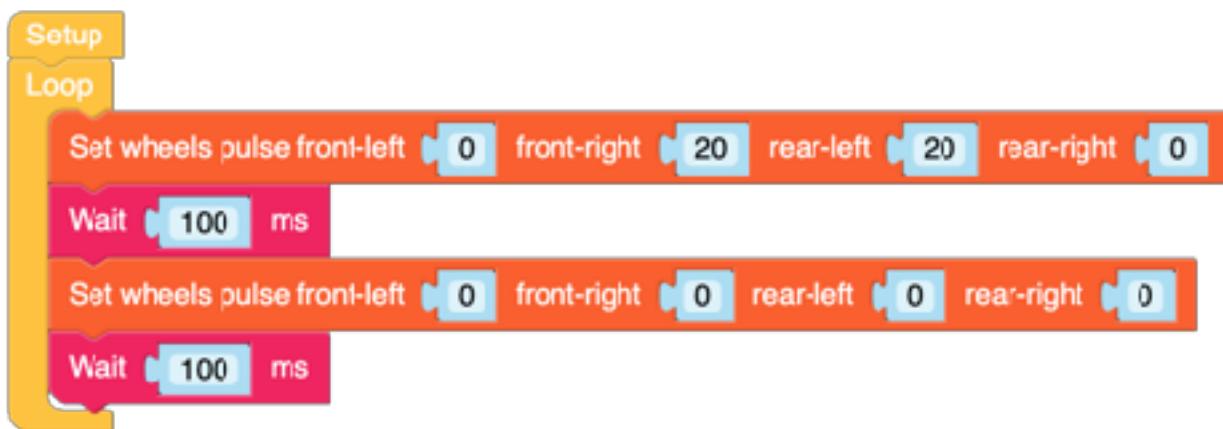
setScreenColor(0x111111)

hat_roverc0 = hat.get(hat.ROVERC)

while True:
    hat_roverc0.SetAllPulse(20, -20, -20, 20)
    wait_ms(100)
    hat_roverc0.SetAllPulse(0, 0, 0, 0)
    wait_ms(100)
    wait_ms(2)
```

RoverC

Slide Forward Left



The Micropython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

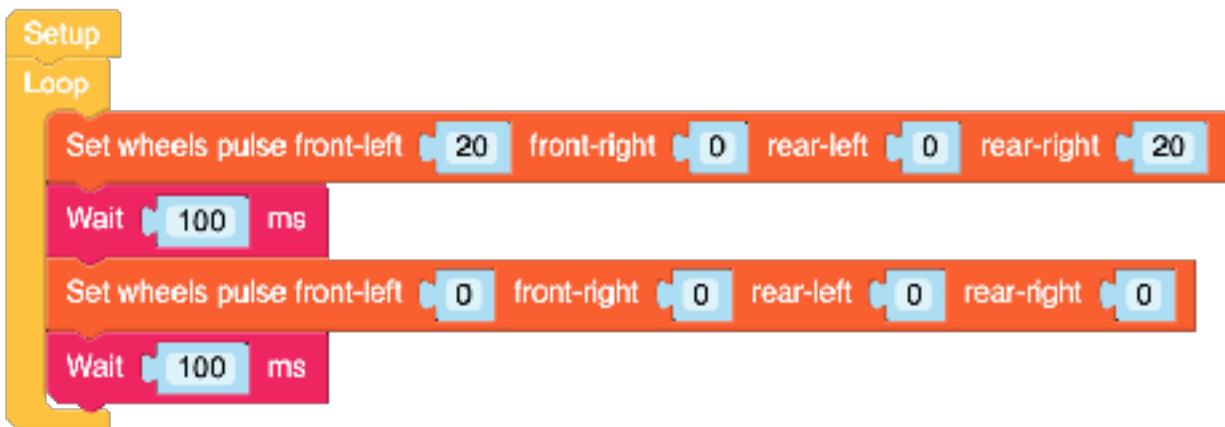
setScreenColor(0x111111)

hat_roverc0 = hat.get(hat.ROVERC)

while True:
    hat_roverc0.SetAllPulse(0, 20, 0, 20)
    wait_ms(100)
    hat_roverc0.SetAllPulse(0, 0, 0, 0)
    wait_ms(100)
    wait_ms(2)
```

RoverC

Slide Forward Right



The Micropython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

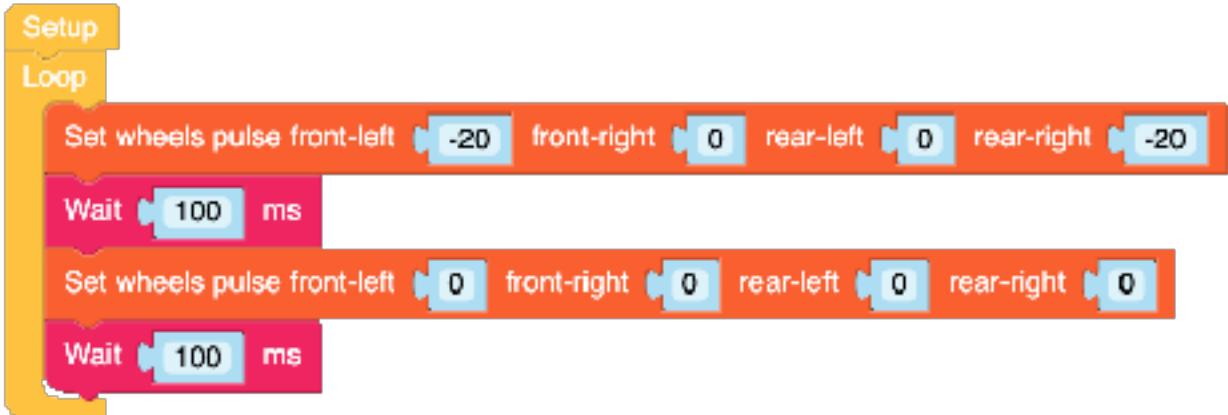
setScreenColor(0x111111)

hat_roverc0 = hat.get(hat.ROVERC)

while True:
    hat_roverc0.SetAllPulse(0, 20, 20, 0)
    wait_ms(100)
    hat_roverc0.SetAllPulse(0, 0, 0, 0)
    wait_ms(100)
    wait_ms(2)
```

RoverC

Slide Backwards Left



The Micropython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_roverc0 = hat.get(hat.ROVERC)

while True:
    hat_roverc0.SetAllPulse(-20, 0, 0, -20)
    wait_ms(100)
    hat_roverc0.SetAllPulse(0, 0, 0, 0)
    wait_ms(100)
    wait_ms(2)
```

RoverC

Slide Backwards Right



The MicroPython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *
import hat

setScreenColor(0x111111)

hat_roverc0 = hat.get(hat.ROVERC)

while True:
    hat_roverc0.SetAllPulse(0, -20, -20, 0)
    wait_ms(100)
    hat_roverc0.SetAllPulse(0, 0, 0, 0)
    wait_ms(100)
    wait_ms(2)
```

TOF

Thermal Camera

Card KB

Units

The M5Stack and M5Stick family share a series of add ons called Units. These units connect to the four pin I2C connector the M5Stack core or M5Stick.

In this chapter I will show you the blocks dedicated to specific units and give basic examples on how to work with them.

AC Socket

AC Socket

Set AC Socket Value

Set acsocket0 value 0

This block is used to turn on or off the AC socket unit.

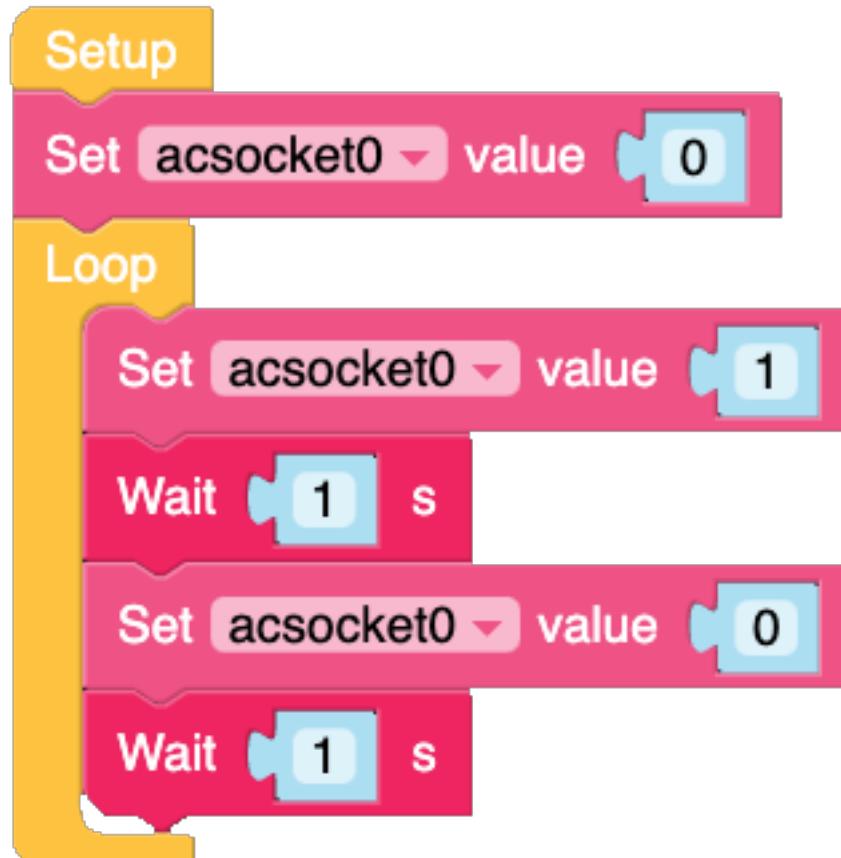
While this is classed as a unit, the AC Socket works over RS485. I am using a Base26 (with the RS485 adapter soldered in place), to test communication here. for the RS485 to work with the AC Socket the port must be set to **PORTC** which is the internal **UART** Port.

The Micropython code for this block is:

```
acsocket0.set_value(0)
```

Example.

In the following example I will just turn the ac socket on and off once a second.



AC Socket

```
from m5stack import *
from m5ui import *
from uiflow import *
import time
import unit

setScreenColor(0x222222)
acsocket0 = unit.get(unit.AC_SOCKET, unit.PORTC)

acsocket0.set_value(0)
while True:
    acsocket0.set_value(1)
    wait(1)
    acsocket0.set_value(0)
    wait(1)
    wait_ms(2)
```

Environmental Unit

**Environmental,
Get Pressure,**

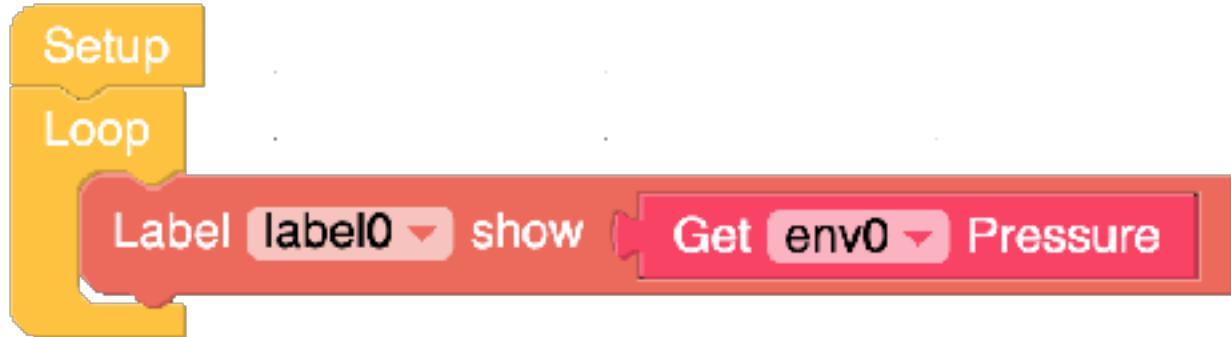
Get env0 ▾ Pressure

The Get Pressure block is a value block that returns the air pressure value read by the sensor. In order to use the value, we need to combine this block with function blocks.

The Micropython code for this block is:

env0.pressure

Example.



In this example I am just using a text block to display the pressure on the M5Stack or M5Sticks screen.

The Micropython code for this example is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x222222)
env0 = unit.get(unit.ENV, unit.PORTA)

label0 = M5TextBox(152, 65, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(env0.pressure))
    wait_ms(2)
```

Environmental Unit

Get Temperature,

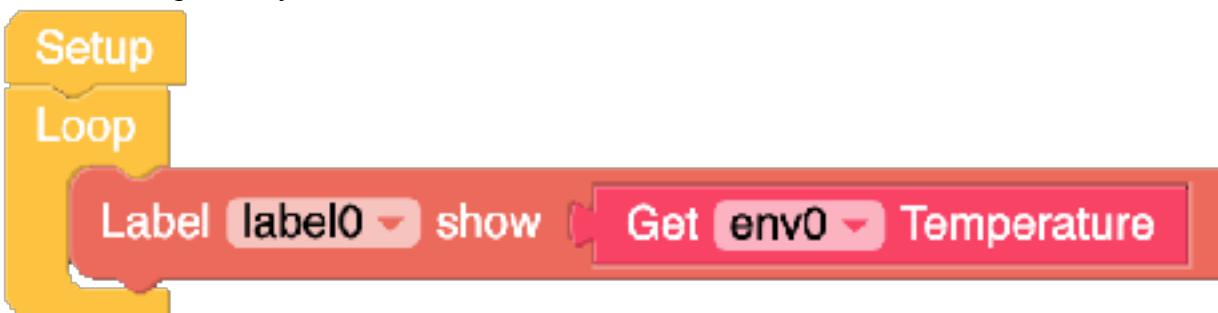
Get env0 ▾ Temperature

The Get Temperature block is a value block that returns the air temperature value read by the sensor. In order to use the value, we need to combine this block with function blocks. The Micropython code for this block is as follows.

env0.temperature

Example

The following example replaces the Pressure block with the temperature block to show the current air temp read by the sensor.



The Micropython code is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x222222)
env0 = unit.get(unit.ENV, unit.PORTA)

label0 = M5TextBox(103, 67, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(env0.temperature))
    wait_ms(2)
```

Environmental Unit

Get Humidity,



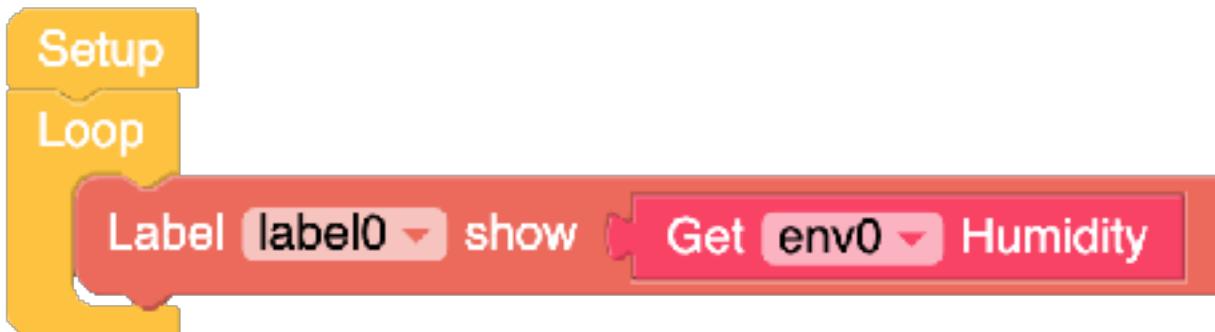
The Get Humidity block is a value block that returns the humidity value read by the sensor. In order to use the value, we need to combine this block with function blocks.

The Micropython code for this block is as follows.

env0.humidity

Example

As with the previous example I have just replace the Pressure block with the humidity block.



The Micropython code is as follows.

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x222222)
env0 = unit.get(unit.ENV, unit.PORTA)

label0 = M5TextBox(103, 67, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(env0.humidity))
    wait_ms(2)
```

Angle

Angle,
Get Angle,

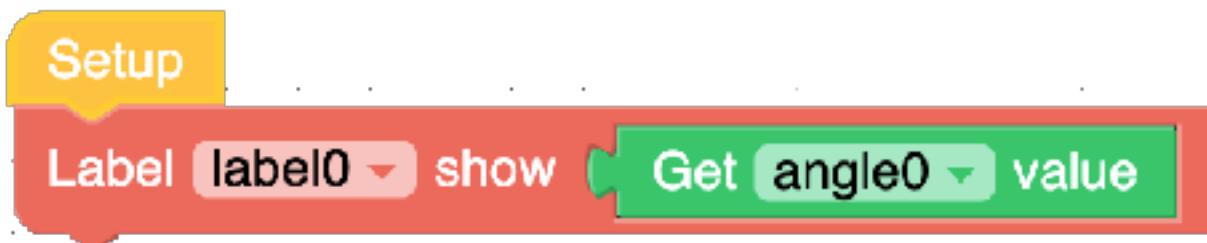
Get angle0 ▾ value

Returns the value generated by the angle sensor. Values are from 0 to 4096.
The Micropython code for this block is.

angle0.read()

Example

In this example I have used a label to show the value read from the sensor.



And the Micropython code for this is:

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x111111)
angle1 = unit.get(unit.ANGLE, unit.PORTA)

label0 = M5TextBox(31, 73, "Text", lcd.FONT_Default,0xFFFFFFF, rotate=0)

while True:
    label0.setText(str(angle0.read()))
    wait_ms(2)
```

PIR Hat

Get PIR Status,

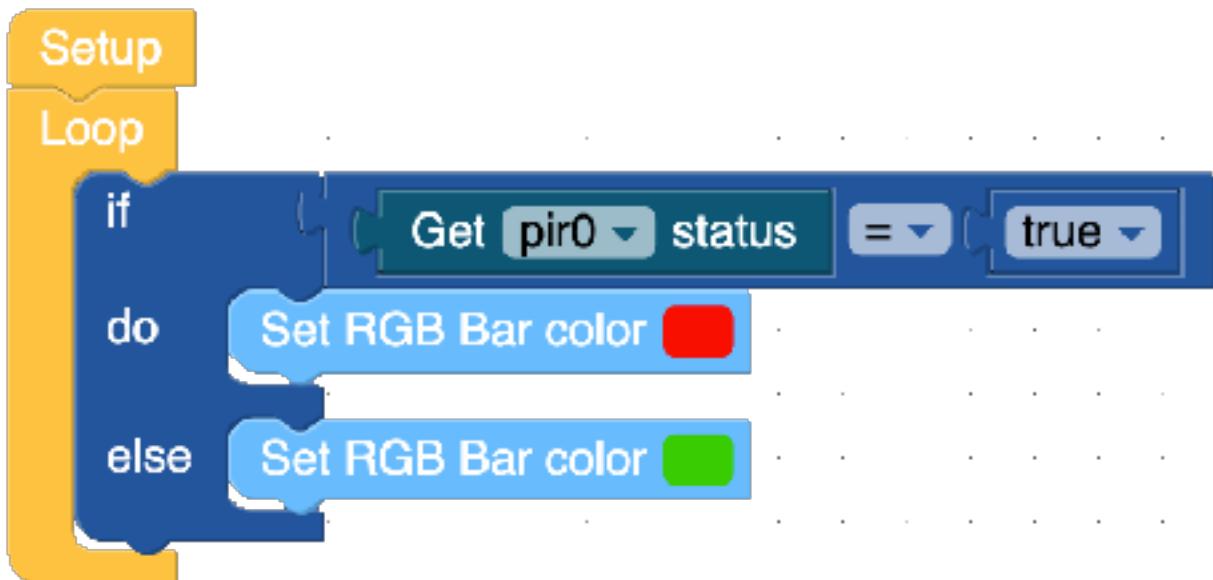
Get pir0 status

Get PIR Status block is a value block that returns true if the sensor detects I.R light given off of people or returns false if nothing gets detected.

The Micropython code for this block is:

pir0.state

Example



In this demo the PIR is connected to Port B of the M5Go. When it detects a human, the lights on the base turn red and only clear when a human is no longer detected.

The Micropython code for this is as follows:

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x222222)
pir0 = unit.get(unit.PIR, unit.PORTB)
while True:
    if (pir0.state) == True:
        rgb.setColorAll(0xff0000)
    else:
        rgb.setColorAll(0x33cc00)
    wait_ms(2)
```

RGB LED

RGB LED,

There are several units produced by M5Stack that fall into the RGB LED unit category, but also any RGB product that uses the WS2812b or the SK6812 RGB LEDS can be controlled with the blocks from this category.

The current available products that use these blocks are as follows

Hex RGB Unit

The HEX RGB unit contains 37 SK6812 LEDs.

RGB LED Unit

The RGB LED Unit contains three SK6812 LEDs.

RGB Flex Strip

The RGB Flex strip is a self adhesive strip of SK6812 LEDs that is available in various lengths from M5Stack and has a grove connector on each end.

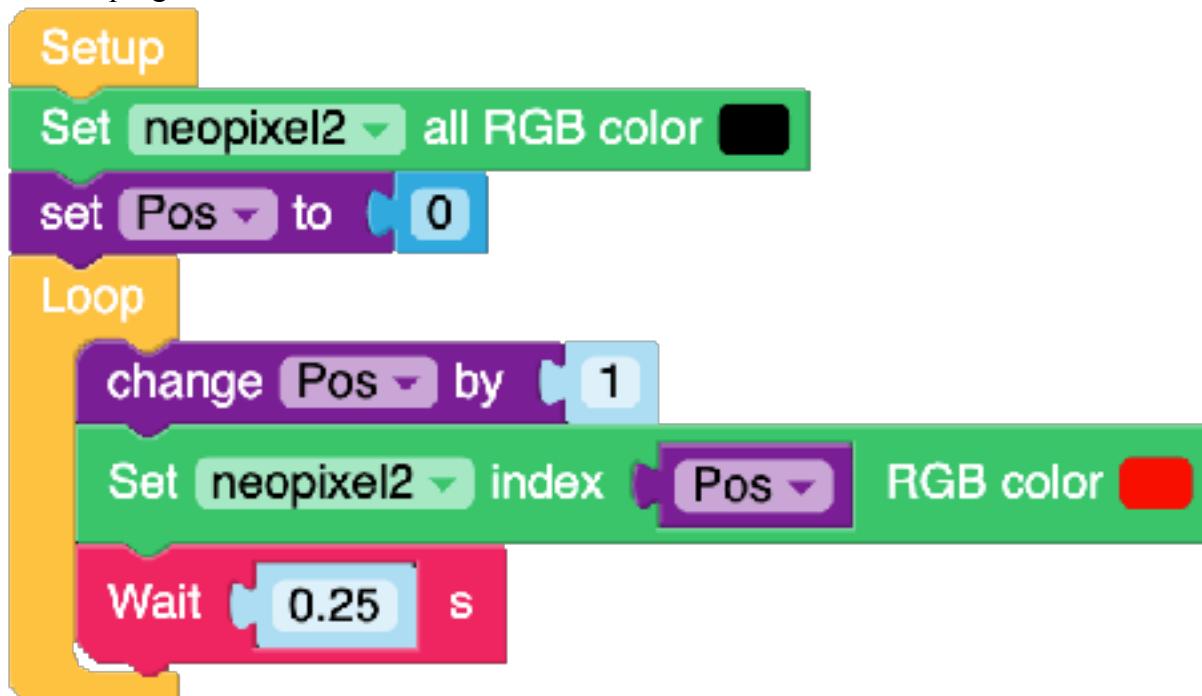
RGB Cat Ear

The RGB Cat Ear is a hand band containing One Hundred and Eighteen SK6812 LEDs.

NeoFlash Light Box.

The Neoflash Light Box contains One Hundred and Ninety two SK6812 LEDs in a grid of eight rows twenty fours LED

In order to understand how the RGB strips and arrays are wired, we created the following UIFlow program.



RGB LED

This program lights an led in a string then moves on to light the next led in sequence showing how they are arranged. This program is useful for testing if arrays have even rows reversed. In the case that they are reversed, the lens will light from left to right then on the next row they will start on the right and move left.

Set Index RGB Colour,



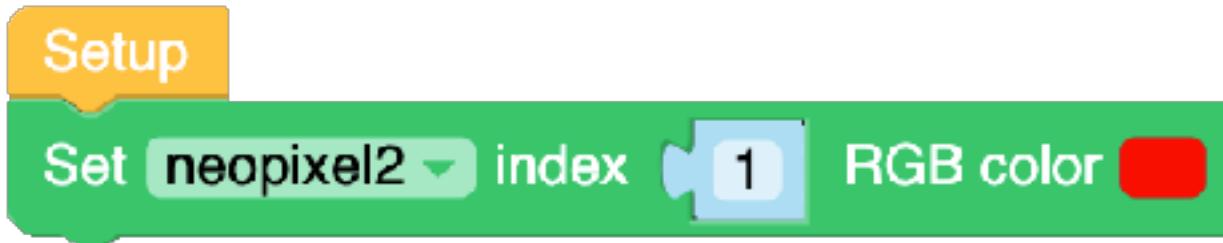
Sets an individual WS2812b LED colour using the Colour picker. Index is a value that can be set with a defined number or a variable and is a reference to the WS2812b led in a position on a array or strip.

The MicroPython code for this block is:

```
neopixel2.setPixelColor(1, 0xff0000)
```

Example

In this example I have set the first WS2812b led to red.



And the MicroPython code for this is as follows:

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x222222)
neopixel2 = unit.get(unit.NEOPIXEL, unit.PORTA, 192)

neopixel2.setPixelColor(1, 0xff0000)
```

RGB LED

Set RGB Range to colour,

Set neopixel2 from 1 to 5 RGB color red

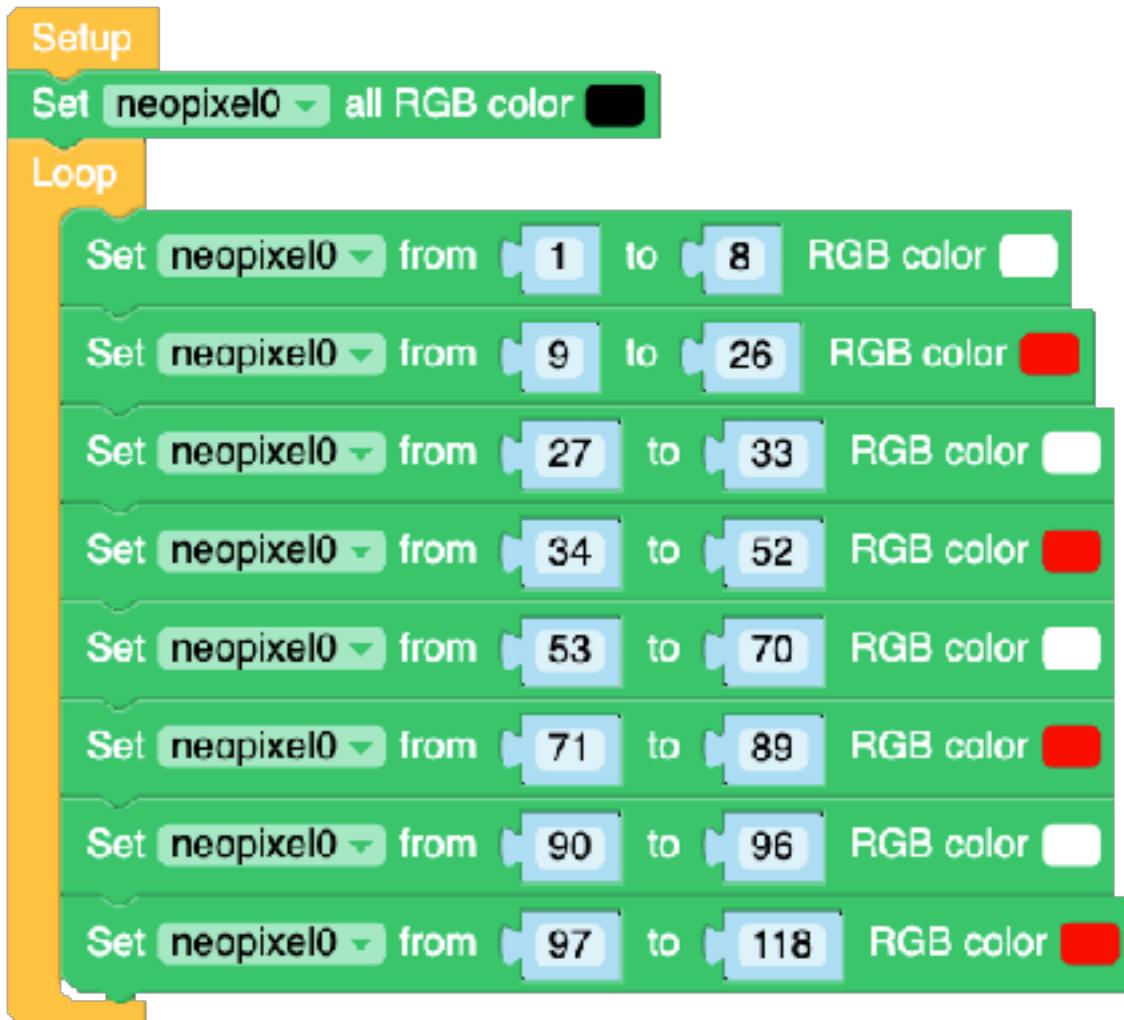
Sets the colour of a range of WS2812b led's using the colour picker. Values can be defined with maths blocks or with variables to specify the led's.

The MicroPython code for this is as follows:

```
neopixel0.setPixelColor(1, 5, 0xff0000)
```

Example

In this example I have used the range blocks to just make the caterpillar glow red.



The MicroPython code for this is as follows:

RGB LED

```
from m5stack import *
from m5ui import *
import units

clear_bg(0x111111)
neopixel0 = units.RGB_Multi(units.PORTA, 118)

neopixel0.setColorAll(0x000000)
while True:
    neopixel0.setColorFrom(1, 8, 0xffffffff)
    neopixel0.setColorFrom(9, 26, 0xff0000)
    neopixel0.setColorFrom(27, 33, 0xffffffff)
    neopixel0.setColorFrom(34, 52, 0xff0000)
    neopixel0.setColorFrom(53, 70, 0xffffffff)
    neopixel0.setColorFrom(71, 89, 0xff0000)
    neopixel0.setColorFrom(90, 96, 0xffffffff)
    neopixel0.setColorFrom(97, 118, 0xff0000)
    wait(0.001)
```

RGB LED

Set WS2812b Range to RGB colour,

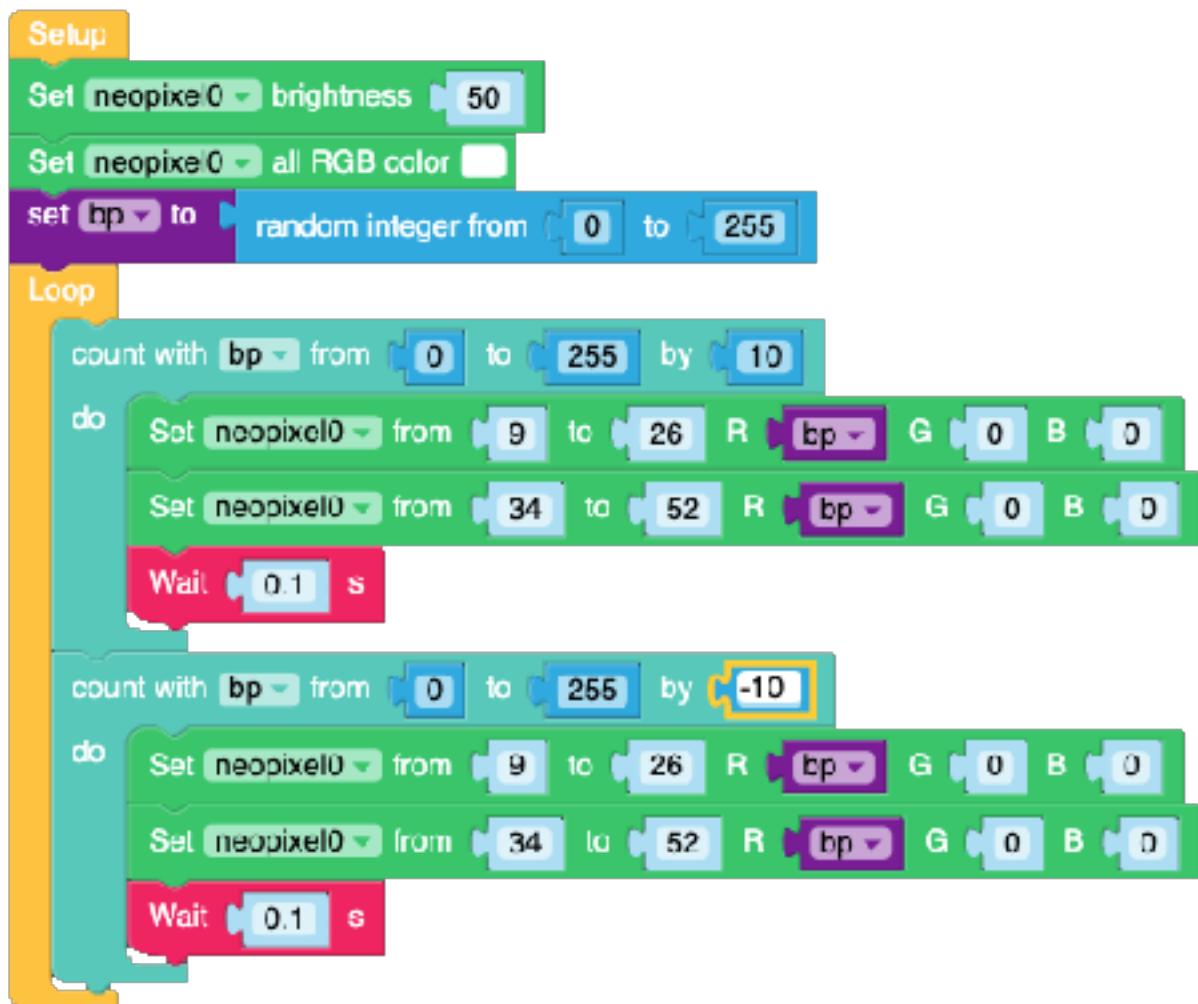
```
Set neopixel2 from 1 to 5 R 0 G 0 B 0
```

Sets the colour of a range of WS2812b led's using individual Red, Green, and Blue values. Values can be defined with maths blocks or with variables to specify the led's.

The MicroPython code for this is as follows.

```
neopixel0.setPixelColor(9,26,(bp << 16) | (0 << 8) | 0)
```

Example.



In this example I have set all the RGB LEDs to white and am making the ears fade in and out between red and white.

RGB LED

The Micropython code for this is:

```
from m5stack import *
from m5ui import *
import units

clear_bg(0x111111)
neopixel0 = units.RGB_Multi(units.PORTA, 118)
import random
bp = None
neopixel0.setBrightness(50)
neopixel0.setColorAll(0xffffffff)
bp = random.randint(0, 255)
while True:
    for bp in range(0, 256, 10):
        neopixel0.setColorFrom(9,26,(bp << 16) | (0 << 8) | 0)
        neopixel0.setColorFrom(34,52,(bp << 16) | (0 << 8) | 0)
        wait(0.1)
    for bp in range(0, 256, 10):
        neopixel0.setColorFrom(9,26,(bp << 16) | (0 << 8) | 0)
        neopixel0.setColorFrom(34,52,(bp << 16) | (0 << 8) | 0)
        wait(0.1)
    wait(0.001)
```

RGB LED

Set all WS2812b Colour,

Set neopixel2 ▾ all RGB color 

Sets all WS2812b LEDs to the same colour using the colour picker. We can use this block in the setup phase before the loop to set all the pixels in an array to black or off.

The MicroPython code for this is:

```
neopixel0.setColorAll(0xff0000)
```

Example

Setup

Set neopixel2 ▾ all RGB color 

Here I have used the Set All block in the setup section to set all the RGB led's to black or off.

The MicroPython code for this demo code is:

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x111111)
neopixel0 = unit.get(unit.NEOPIXEL, unit.PORTA, 192)

neopixel0.setColorAll(0x000000)
```

RGB LED

Set WS2812b Brightness,

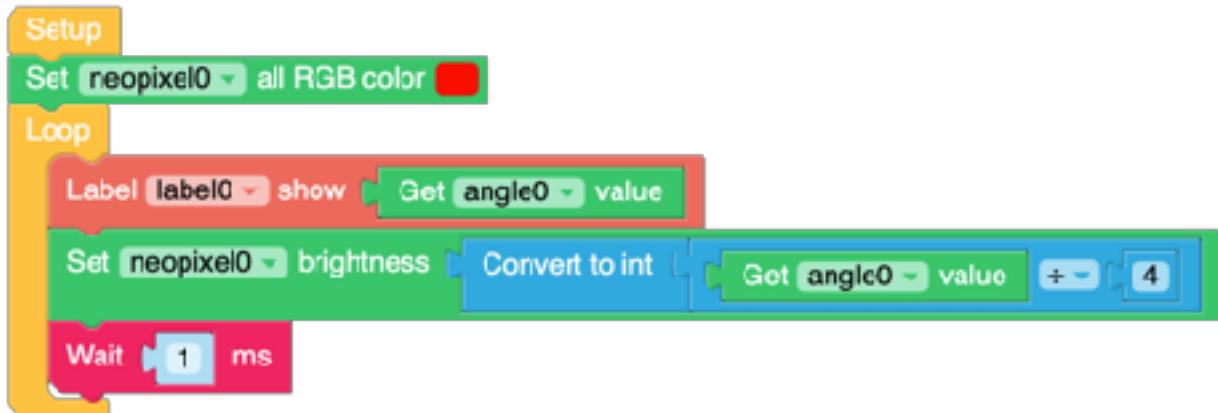
Set neopixel0 brightness 20

Set the brightness levels of all WS2812b LEDs Brightness can only have a value between 0 and 100.

The Micropython code for this block is:

`neopixel0.setBrightness(20)`

Example.



In this example I have use the Angle Unit to control the brightness of the RGB Led's. The Get Angle block returns values as a string with a value from 0 to 1024. In order to use this value as to control the brightness which only has values from 0 to 256 we have to divide the value by four and then use the **convert to int** block to generate the value that the **set brightness** block can use.

RGB LED

And the Micropython code for this is:

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x222222)
neopixel0 = unit.get(unit.NEOPIXEL, unit.PORTA, 256)
angle0 = unit.get(unit.ANGLE, unit.PORTB)

label0 = M5TextBox(79, 63, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

neopixel0.setColorAll(0xff0000)
while True:
    label0.setText(str(angle0.read()))
    neopixel0.setBrightness(int(((angle0.read()) / 4)))
    wait_ms(1)
    wait_ms(2)
```

RGB LED

Set WS2812b Hex Colour,



Used to set the colour of the individual WS2812b LEDs on the Neohex using the colour picker. To select an LED to set to the selected colour, click on a light green pad and a tic will appear to show that the position is now selected.

The MicroPython code for this block is:

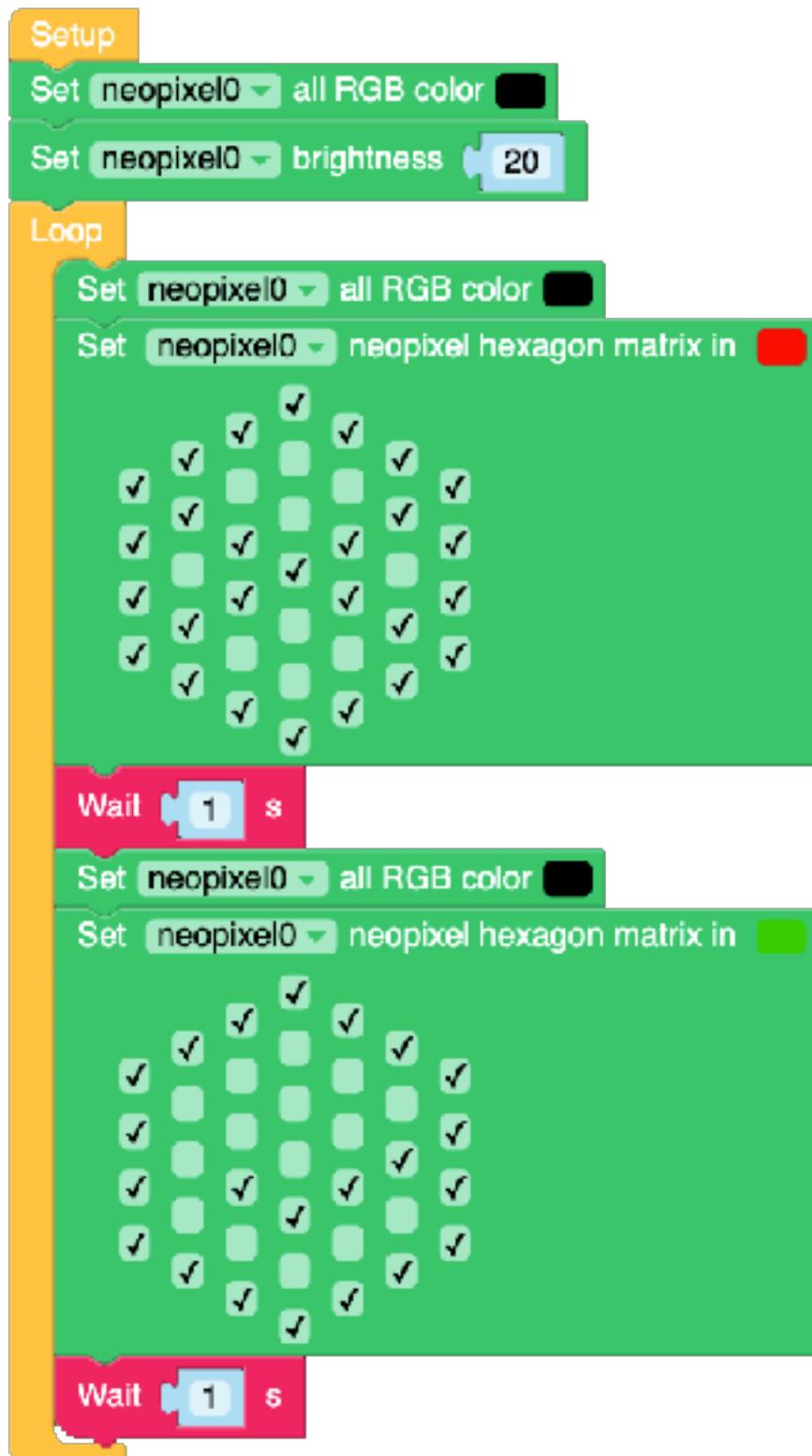
```
neopixel0.setPixelColor(0, 0x000000)
```

The numbers in the brackets are the LED position followed by the hex colour code. The LED positions are shown in the following image:



RGB LED

Example



In this example, when run the Neohex flashes between a red X and a green tick.

RGB LED

And here is the Micropython code for this example:

```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x111111)
neopixel0 = unit.get(unit.NEOPIXEL, unit.PORTA, 38)

neopixel0.setColorAll(0x000000)
neopixel0.setBrightness(20)
while True:
    neopixel0.setColorAll(0x000000)
    neopixel0.setColorFrom(1, 6, 0xFF0000)
    neopixel0.setColorFrom(8, 10, 0xFF0000)
    neopixel0.setColorFrom(12, 13, 0xFF0000)
    neopixel0.setColorFrom(15, 16, 0xFF0000)
    neopixel0.setColor(19, 0xFF0000)
    neopixel0.setColorFrom(22, 23, 0xFF0000)
    neopixel0.setColorFrom(25, 26, 0xFF0000)
    neopixel0.setColorFrom(28, 30, 0xFF0000)
    neopixel0.setColorFrom(32, 37, 0xFF0000)
    wait(1)
    neopixel0.setColorAll(0x000000)
    neopixel0.setColorFrom(1, 5, 0x33cc00)
    neopixel0.setColor(7, 0x33cc00)
    neopixel0.setColorFrom(9, 10, 0x33cc00)
    neopixel0.setColor(13, 0x33cc00)
    neopixel0.setColorFrom(15, 16, 0x33cc00)
    neopixel0.setColor(20, 0x33cc00)
    neopixel0.setColorFrom(22, 23, 0x33cc00)
    neopixel0.setColor(26, 0x33cc00)
    neopixel0.setColorFrom(28, 29, 0x33cc00)
    neopixel0.setColorFrom(33, 37, 0x33cc00)
    wait(1)
    wait_ms(2)
```

RGB LED

Set WS2812b Hex (Variable) Colour,



Used to set the colour of the individual WS2812b LEDs on the Neohex using individual Red, Green and Blue values.

The Micropython code is the same as in the previous example however this block allows us to control the values with numbers which UIFlow then translates into the hexadecimal values.

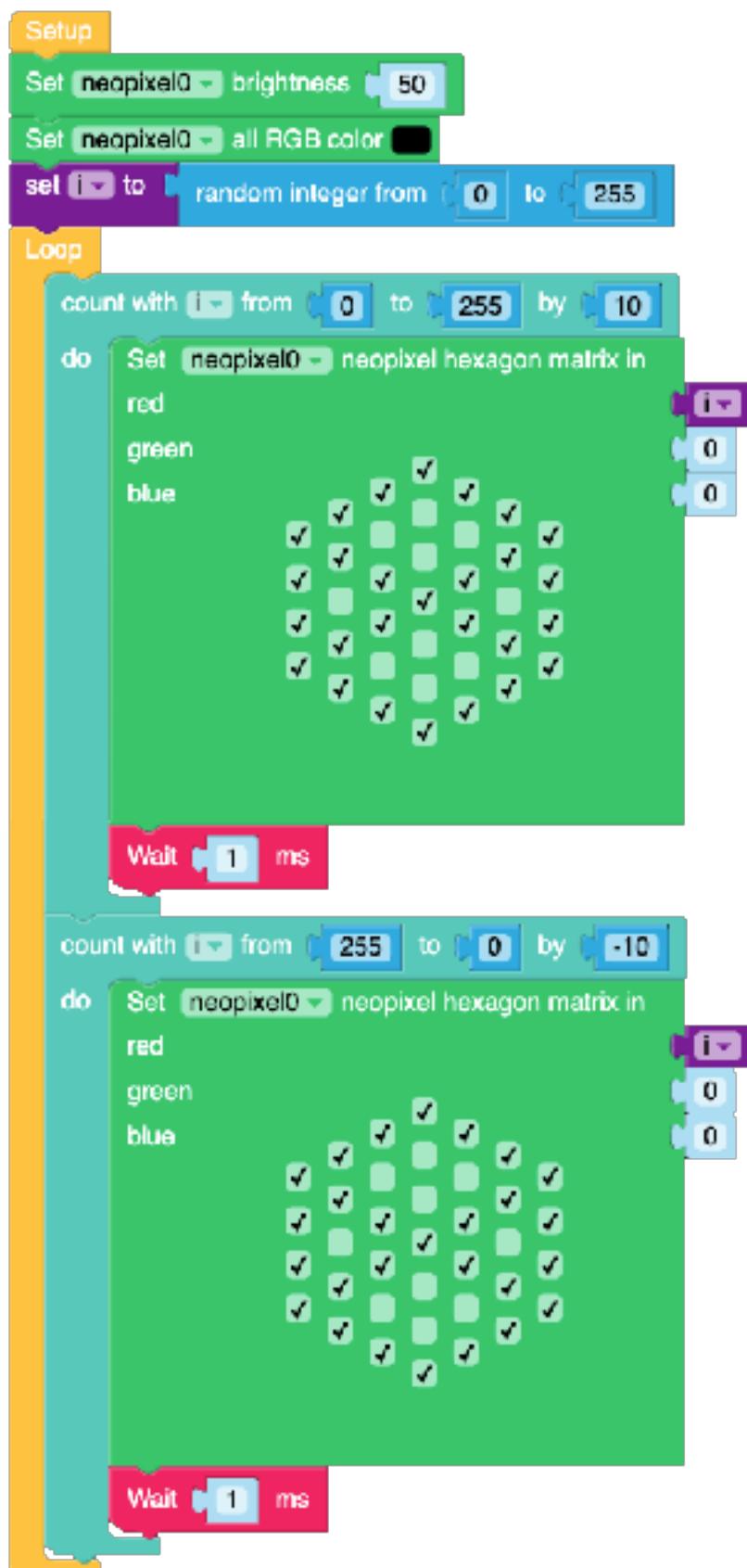
```
neopixel0.setPixelColor(0, 0x000000)
```

Example

In the following example I have used the fading demo from the RGB range block to brighten and darken the red cross.

RGB LED

The Micropython code for this is as follows.



Joystick,

Get X,



Returns the joysticks X value.

Get Y,



Returns the joysticks Y Value.



Is Pressed,

Returns the state of the joysticks button when pressed.

Reverse X



Returns an inverted value of the X position.

Reverse Y



Returns an inverted value of the Y position.

Light,
Get Light Analogue Value,



Returns the analogue value from the light sensor.

Get Light Digital Value,



Returns the digital value from the light sensor.

Earth,

Get Earth Analogue Value,



Returns the analogue value from the earth sensor.

Get Earth Digital Value,

Returns the digital value from the earth sensor.



Makey,

Get Value,



Get all Value,



Servo,



Servo Rotate to Degree,

Tells the servo to move to a specified position.

Servo Speed,



Sets the move speed of the servo.

Weight,



Zero Weight,

Sets the values coming from the sensor to zero.

Get Weight,



Get the weight from the sensor.



Get Raw Data

Gets the raw 24 bit value data from the sensor.

Button, Button Loop,



The button loop continuously watches the button unit and then runs the code inside it when it detects a triggering even.

The four events the button loop waits for are as follows.

Was Pressed - This is the normal mode of operation and waits until a single short press of the button is detected before running the code inside it.

Was Released - This waits until the button is pressed and released before running the code placed inside it.

Long Press - Like the "Was Pressed" block, it waits until the button is pressed however, this block activates if the button is pressed and held down for a few seconds.

Was Double Pressed - This function checks to see if the put-on was pressed and released twice within a few milliseconds before running the code inside it.

Button Action,

The obtain button block works in a similar mode to the Button loop but instead of running code



placed inside it, it is a value block that tells other code if a button event has been triggered.

Duel Button,



Duel_Button Loop,

The duel button loop continuously watches the two buttons on the duel button unit and then runs the code inside it when it detects a triggering event on either of the buttons.

The four events the button loop waits for are as follows.

Was Pressed - This is the normal mode of operation and waits until a single short press of the button is detected before running the code inside it.

Was Released - This waits until the button is pressed and released before running the code placed inside it.

Long Press - Like the "Was Pressed" block, it waits until the button is pressed however, this block activates if the button is pressed and held down for a few seconds.

Was Double Pressed - This function checks to see if the button was pressed and released twice within a few milliseconds before running the code inside it.

Button Action,

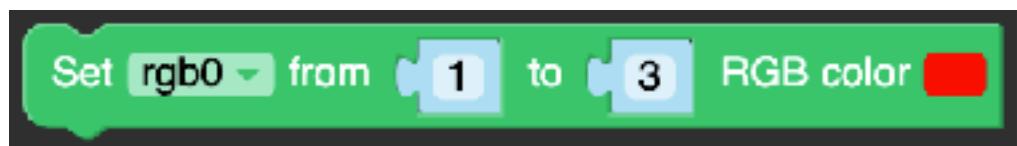


The obtain duel button block works in a similar mode to the Button loop but instead of running code placed inside it, it is a value block that tells other code if a one of the two buttons have been triggered.

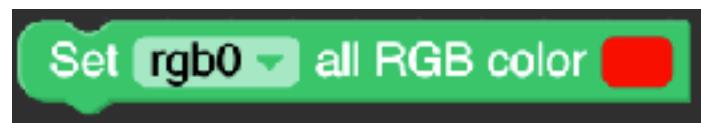
RGB,



Set RGB Bar Colour,
Sets one of the three R.G.B LED colours using the colour picker.



Set RGB Bar Colour R,G,B,
Sets more than one R.G.B LED colours using the colour picker.



Set RGB all Colour,
Sets all the R.G.B LED colours using the colour picker.



Set RGB Brightness,
Sets the Brightness of the R.G.B LEDs.

Relay,

Set Relay On,



Sets the relay unit in to the on position.

Set Relay Off,



Sets the relay unit in to the off position.

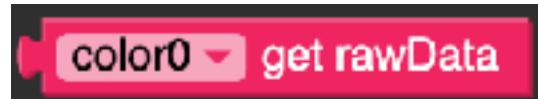
Heart Unavailable,
Not available in UIFlow at present.

ADC,
ADC Read,



Returns an analogue reading from the Analogue to Digital Converter Unit.

Colour,



Get Raw data.

Returns the three raw values for Red, Green and blue from the sensor.



Get Red

Returns the red value from the sensor.

Get Green



Returns therein value from the sensor.



Get Blue

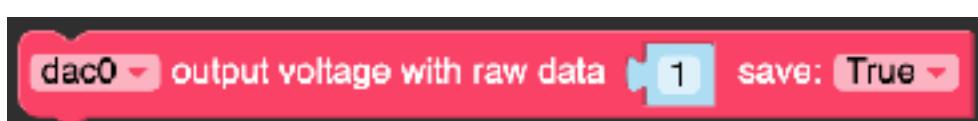
Returns the blue value from the sensor.

DAC,

dac0 Output Voltage



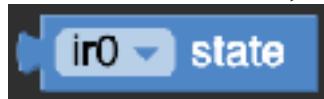
Gets a voltage reading from 0 to 3.3 volts If save it set to true then the data will be written to the M5Sticks E.E.P.R.O.M.



dac0 Output Voltage with Raw Data.

Gets a raw data reading of 0 to 4096. If save it set to true then the data will be written to the M5Sticks E.E.P.R.O.M.

**IR,
Get IR State,**



Returns the state of the IR Unit.



Set IR On,
Turns the IR unit on.
Set IR Off,



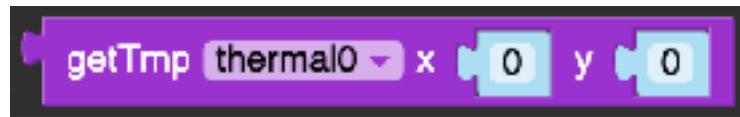
Turns the IR unit off.

NCIR,
NCIR Read,



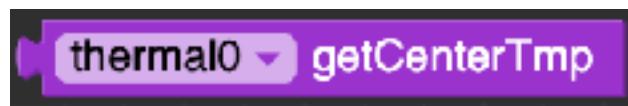
Returns values from the NCIR Unit.

Thermal,



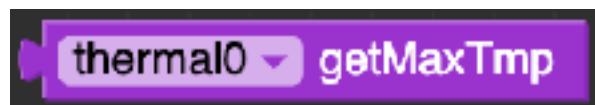
Get Temp X, Y,

Returns the temperature detected at the specified coordinates.



Get Centre Temp

Returns the temperature detected in the middle of the sensor.



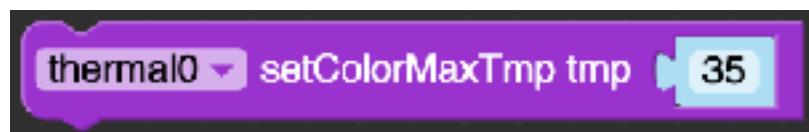
Get Max Temp,

Returns the maximum temperature detected by the unit.



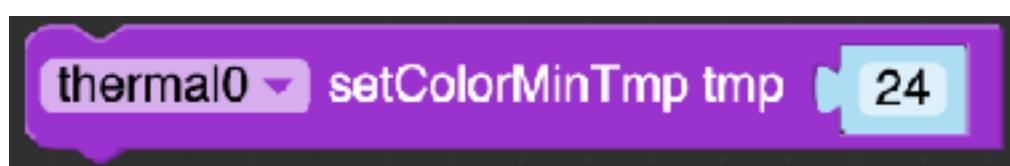
Get Min Temp,

Returns the minimum temperature detected by the unit.



Set Colour Max Temp,

Sets the upper level that the sensor will use.



Set Colour Min Temp,

Sets the lower level that the sensor will use.

Update X, Y, show centre,



Changes the X, Y temperature read position while displaying the centre cross hair.

TOF, Get Distance,



Returns the distance value from the T.O.F Unit.

Example

In the following example I am using two labels to show readings from two separate but identical T.O.F units.

```
from m5stack import *
```



```
from m5ui import *
from uiflow import *
import unit

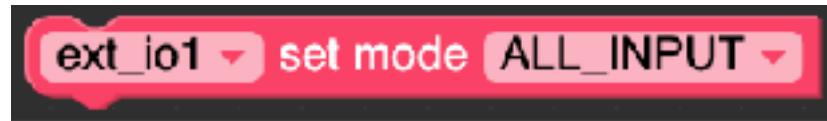
setScreenColor(0x222222)
pahub0 = unit.get(unit.PAHUB,
unit.PORTA)
tof2 = unit.get(unit.TOF, unit.PAHUB0)
tof3 = unit.get(unit.TOF, unit.PAHUB1)

label0 = M5TextBox(29, 90, "Text",
lcd.FONT_Default,0xFFFF, rotate=0)
label1 = M5TextBox(177, 99, "Text",
lcd.FONT_Default,0xFFFF, rotate=0)

while True:
    label0.setText(str(tof2.distance))
    label1.setText(str(tof3.distance))
    wait_ms(2)
```

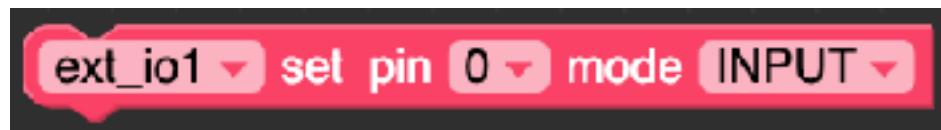
EXT I/O,

Set I/O Port Mode,



Set the mode of all the pins to Input or output.

Set I/O Pin Mode,



Sets the individual pins to input or output.

Digital WritePort,



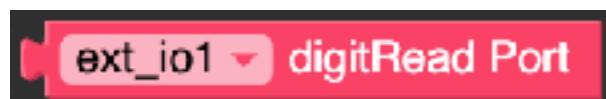
Writes a hex value to the I/O unit.

Digital Write Pin,

Set each individual pin as high or low.

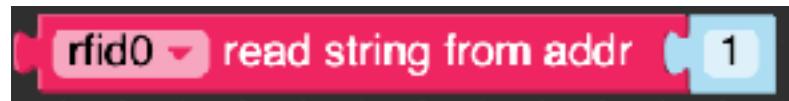


Digital Read Port,



Reads the digital state of the I/O Unit.

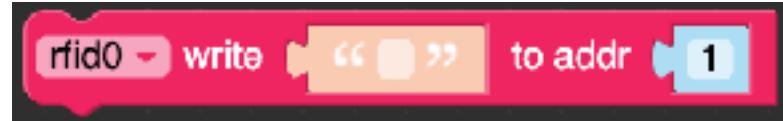
RFID,



Read String,

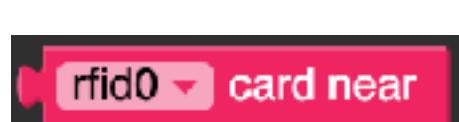
Reads a string of information stored at the specified address on the RFID card or tag.

Write String,



Writes a string of date to the specified address.

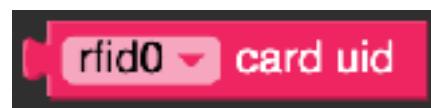
Card Near,



Returns true or false if an RFID tag or card is

in proximity of the unit.

Card UID,



Returns the UID stored on the RFID tag or

card.

PAHub

While the PAHub has some blocks to use, the hub is designed to allow the connection of multiple identical units that have the same I2C address.

The blocks provided are as follows.

Set position state

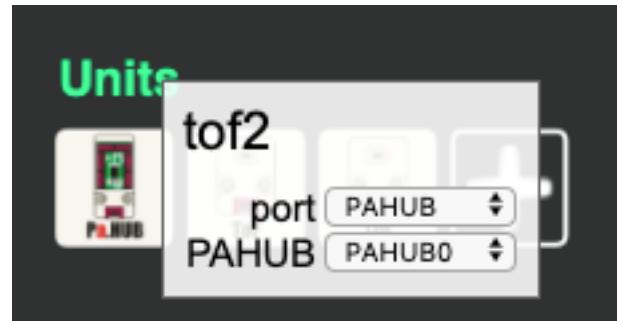
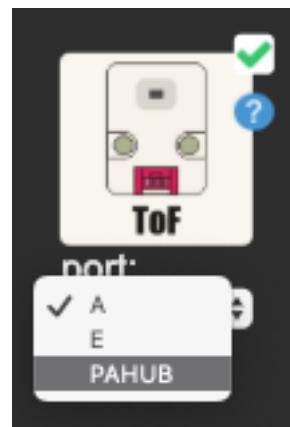
Set position

Set port value

Example

In the following example I am using two labels to show readings from two separate but identical T.O.F units.

To use the Pahub unit we first need to add it by clicking on the "+" sign and then we need to add the T.O.F units. Click on the "+" to add the T.O.F but then click on the "Port" drop down box and select PAHUB and the OK to add it. Repeat the previous steps to add a second T.O.F.



Now click on the T.O.F under the virtual M5Stack and another window will appear.

This is how we configure the PaHub's ports. The Pahub has six ports labeled from 0 to 5 with 0 in the bottom right and 5 on the bottom left. Set the shown window to the locations on the physical pahub where you plugged in the two T.O.F sensors.



```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x222222)
pahub0 = unit.get(unit.PAHUB,
unit.PORTA)
tof2 = unit.get(unit.TOF, unit.PAHUB0)
tof3 = unit.get(unit.TOF, unit.PAHUB1)

label0 = M5TextBox(29, 90, "Text",
lcd.FONT_Default,0xFFFFF, rotate=0)
label1 = M5TextBox(177, 99, "Text",
lcd.FONT_Default,0xFFFFF, rotate=0)

while True:
    label0.setText(str(tof2.distance))
    label1.setText(str(tof3.distance))
    wait_ms(2)
```

To view an introduction video on the PAHub, you can check out Lukes video on the M5Stack youtube channel <https://www.youtube.com/watch?v=nMsCwqCE5c8>

Heart Unit

The Heart unit is based on the MAX 30100 pulse oximetry sensor and is used to read the heart rate when placed on a fingertip.

The heart unit has the following blocks.

Get Heart Rate,

Get SPO2,

Set Mode,

Set LED Current.

Modules

M5Stack cores have additional expansion units that share the sam 50mm X 50mm footprint. These modules can be stacked between the M5Stack cores and M5Stack bases.

Stepper Motor,

Stepper Motor Module Address,



Sets the Hex address to use for the stepper motor driver.

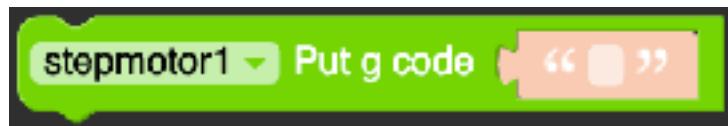
Stepper Motor Position,



Sets the position and speed that the stepper motor is to move to.

Run "G" Code,

For running G-Code in the modules g-code interpreter.



Stepper Motor Movement Mode,

Sets how the stepper motor is to move. Distance is used to make it move a defined distance,



Absolute is use to make the motor move to a set of coordinates.

Lock Stepper Motor,

Locks the stepper motor by powering the cores preventing the motor from being mechanically



turned.

Unlock Stepper Motor,

Unlocks the stepper motor by removing the power to the coils allowing them to be mechanically



turned.

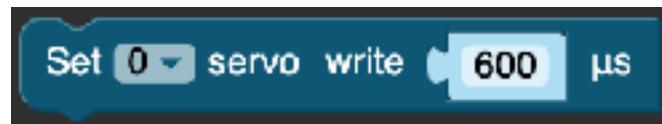
Servo,

Set Servo rotate,



Tells the servo on the defined channel to rotate to a specified position.

Set Servo Speed,



Sets the move speed of the servo on the defined channel.

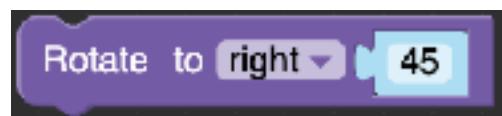
Bala,



Bala Move distance,
Moves the Bala motors forward or backward by a user defined value.



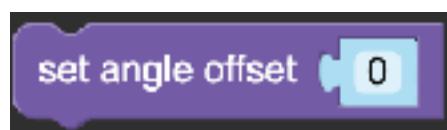
Turn Wheel
Turns the left or the right wheel wheel by a user defined value.
Rotate



Tells the bala to rotate.

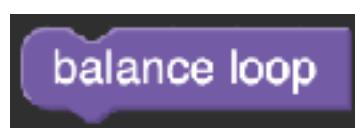


Get Angle.
Returns the angle that the Bala unit has turned.



See Angle Offset.
Used to set an angle offset to handle inaccuracies between the motors.

Balance Loop



Tells the M5Stack to update its internal position in memory.

Bala Motor,



Set Motor direction and speed.

Sets the motors rotation direction and speed.

Run forward distance and speed.



Commands the bala to move in a direction a user defined distance at a user defined speed.

Go to Position



Commands the motor to move to a user defined position at a user defined speed.



Stop Motor

Commands the motor on the selected channel to stop.



Read encoder

Returns the values from the Bala motors built in encoder.

Lego +,



Set Lego Motor Rotate PWM,

Sets the Lego motor on channel 1 to rotate clockwise with a value.



Stop Lego Motor,

Stops the motor o channel 1



Clear encoder,

Clears decoder reading from memory.

Read encoder,



Returns the values from the lego motors built in encoder.

Lidarbot

Lidarbot set front with neopixel colour,

Lidarbot set front with neopixel

Sets the colour of all the RGB LED's using the colour picker. You can use this block to set the front bar, the rear bar or, both of the RGB LED bars.

The MicroPython code for this is.

```
lidarbot0.setRgb(0xff0000,'front')
```

In the following example I turn all the RGB LED's on on the front bar then the rear bar and then both before turning them all off.



Lidar Bot

Lidar bot Set Individual LED Colour,



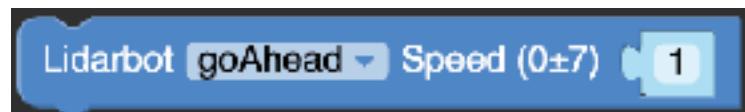
Sets the colour of the individual RGB LED's in the front and rear light bars using the colour picker.

Lidarbot Move,



Moves the Lidar bot in the direction at a user defined speed.

Lidarbot Set Speed,



Sets the speed of each of the individual four wheels.

Lidarbot Set Servo,

Set the angle of a servo connected to the Lidarbot.

Lidarbot Axis Speed,

Set the X and Y axis speed.



Lidarbot Draw Map,

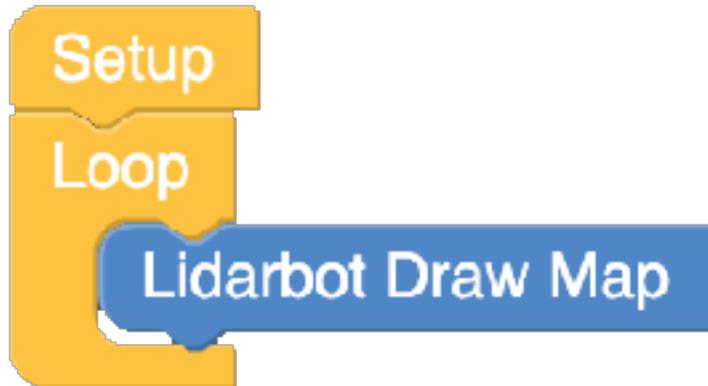
Lidarbot Draw Map

Fetches information from the LIDAR scanner on the top of the robot and displays it on the screen as a map.

The Micropython code for this block is

```
lidarbot0.lidar.draw_map()
```

Example.



This small example is all that is needed to get the LIDAR scanner to show a map of the robots surroundings on the screen. Please note that the loop is required in order for the screen to constantly update or a complete map does not get drawn.

The Micropython code for this example is as follows.

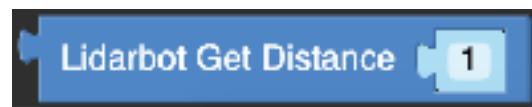
```
from m5stack import *
from m5ui import *
from uiflow import *
import module

setScreenColor(0x222222)

lidarbot0 = module.get(module.LIDARBOT)

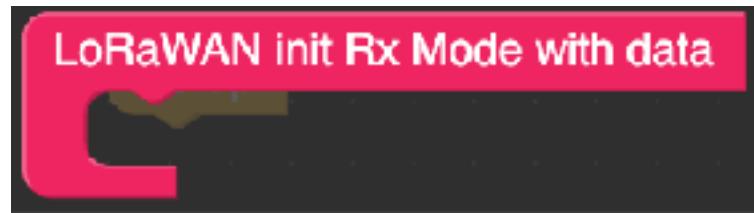
while True:
    lidarbot0.lidar.draw_map()
    wait_ms(2)
```

Lidarbot Get Distance,



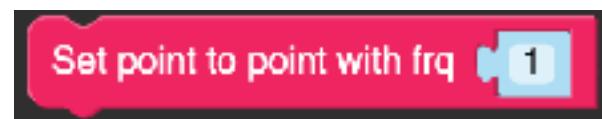
LoRaWan,

Init LoRaWan Rx Mode

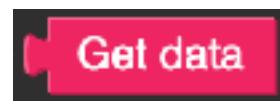
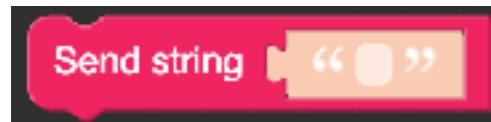


Set Point to Point Frequency.

Send String.



Get Data,



PM2.5



Get PM2.5 Value,

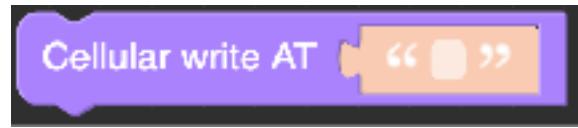
Returns particle count of selected size particles in SPM or APM value.

Get Particle Count,



Returns the amount of particles detected above the selected size.

Cellular,



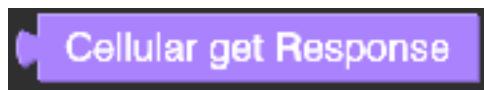
Cellular Write AT,

Used to send AT commands to the Cellular module.

Cellular Wait,



Sets the time that the cellular module waits before responding to AT commands.

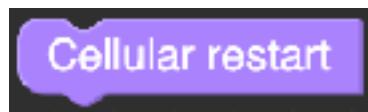


Cellular Get Response,

Immediately returns a response from the cellular module after an AT command is sent.



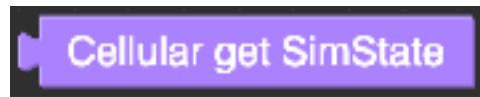
Cellular Test AT



Cellular Restart

Forces the cellular module to restart independent of the M5Stack.

Cellular Get SimState



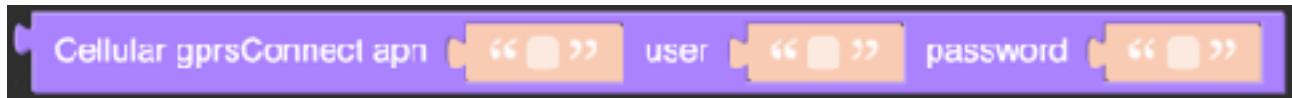
Returns the status of a Sim card plugged into the module.

Cellular Init,



Initialises the Cellular module.

Cellular GPRS Connect,

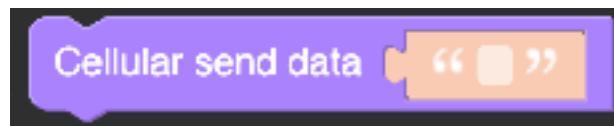


Returns the module status after ordering the module to connect to a GPRS network with the user defined credentials.



Cellular Connect host,

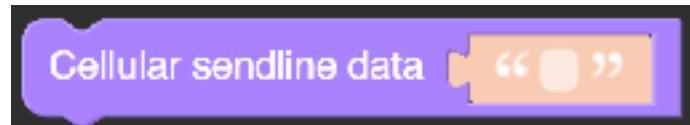
Returns the status of the module after trying to connect to a host with the following user defined credentials.



Cellular Send Data,

Sends data over the connection to the host device.

Cellular Send Line Data,



Cellular Get Available

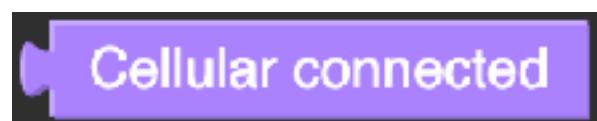
Reports if Cellular networks are available.



Cellular Read

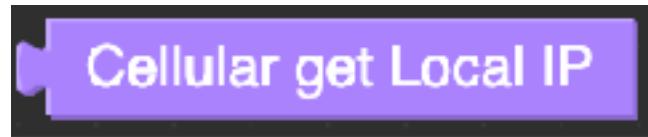
Returns available Cellular networks.

Cellular Connected



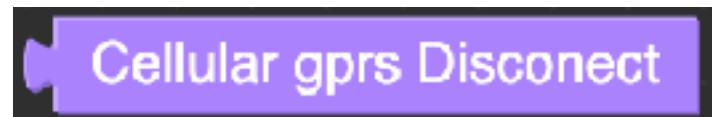
Returns the status if the cellular module has connected to a network or not.

Cellular Get Local IP



Get a local IP from a network that the module has connected to.

Cellular GPRS Disconnect,



Returns the module status after being commanded to disconnect from a network.

Cellular Get Network State,



Reports the status of the Cellular network.

Base X

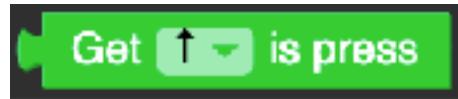
Plus

GoPlus

GPS

Faces Modules

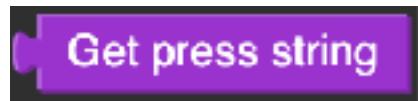
Gameboy Face,



Get Direction Is Pressed,
Get Direction Was Pressed/Released,



Calculator Face,



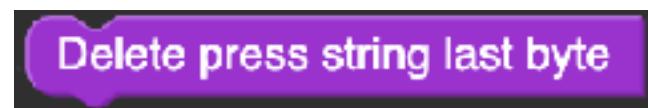
Get Press String,



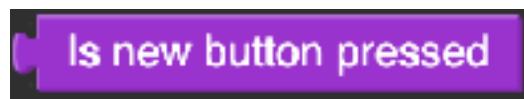
Get Press Button Int Value,
Clear Press String,



Delete Press String,



Is New Button Pressed,



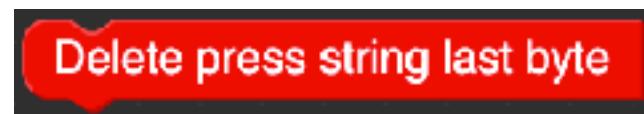
Keyboard Face,



Get Press String,
Clear Press String,



Delete Press String,



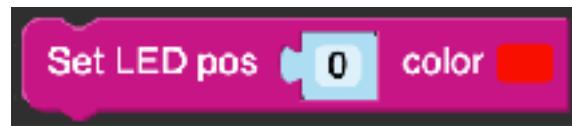
Get Press Button Int Value,



Is New Button Pressed,



Encoder Face,
Set LED Pos Colour,



Clear Encode Value to Zero,
Get Encode Value,



Get Encode Direction,



Is Encode Pressed,



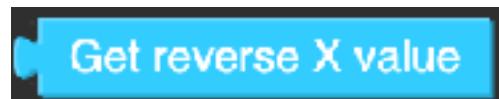
Joystick Face,
Get X Value,



Get Y Value,



Is Pressed,



Get Reverse X Value,
Get Reverse Y Value,

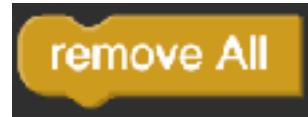


Set LED Pos Colour,

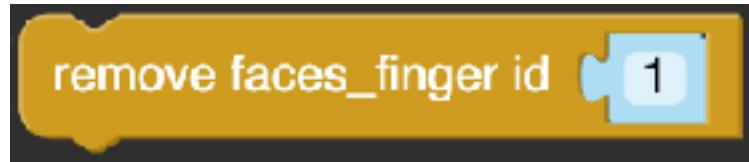
Finger Face,
Get State,



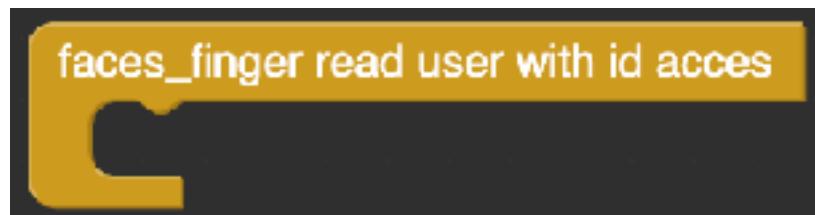
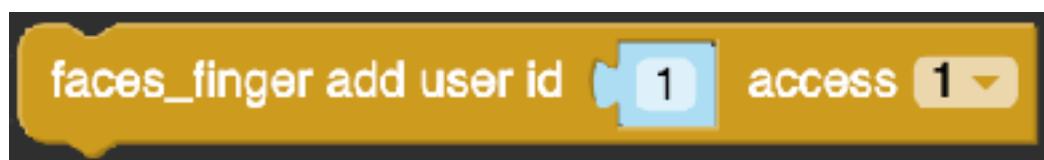
Get access,
Get ID,



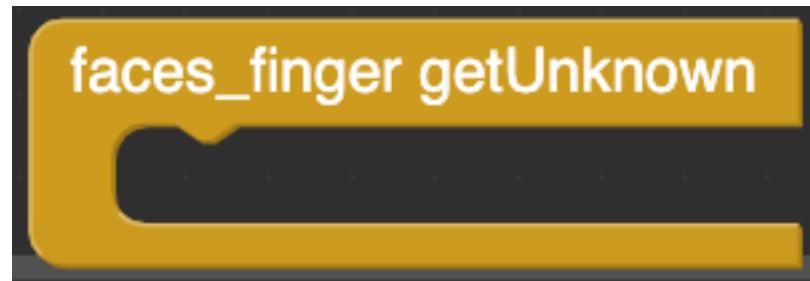
Remove all,
Remover Faces Finger ID (),



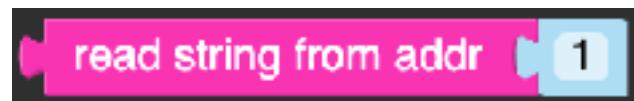
Add User Faces Finger ID (),



Faces Finger read User,
Faces Finger Get Unknown.

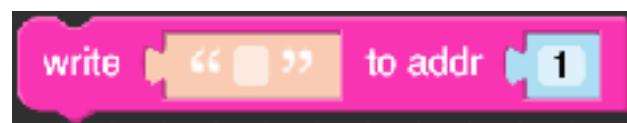


RFID Face,



Read String,

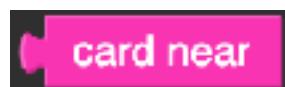
Reads a string from an RFID card or Tag.



Write String,

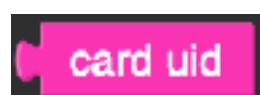
Writes a string to the RFID card or tags storage space.

Card Near,



Returns true if an RFID card or tag is detected.

Card UID,



Gets the Card or Tag UID.

Logic Blocks

This section contains the various blocks that allow us to perform maths and other tasks that tie the blocks in the earlier sections together.

Variables

Variables,

Variables are containers for values that can be changed by our code. We have already seen some variable in use as the hardware units and modules use variable to pass the values from their sensors to be used by our programs in order to set actions based on their values.

Create Variable,



Creates a Variable

By clicking this block to create a variable, we are presented with a message to give our variable a name. Once a name is set and OK is pressed, the following blocks will be created.



Set Variable,

Sets the initial value of the variable to the value of the following connected mathematical block or a sensor block..

The Micropython code for this block is

V1=None



Change Variable,

This function changes the value of the variable to that of a value from a mathematical block or a sensor block.

The Micropython code for this is

V1 = (V1 if isinstance(V1, int) else 0) + 1



Variable,

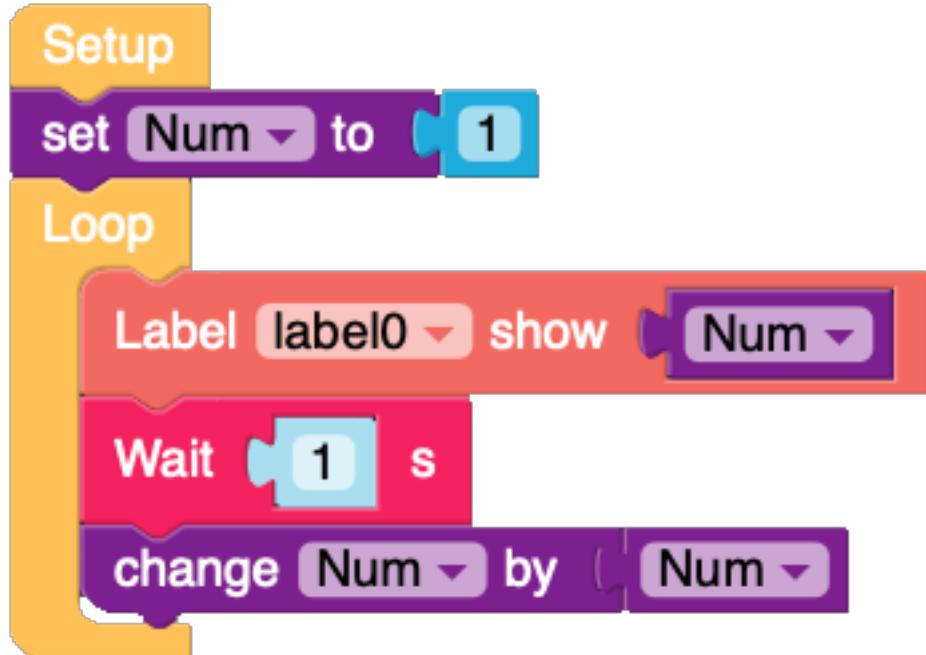
The variable block that other functions can query.

The Micropython code for this block is

(V1)

Example

In this example I have used all three of the variable blocks. The program calculates a value and then multiples that value by the calculated value.



And the Micropython code for the example.

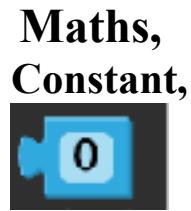
```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(77, 69, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

num = None

num = 1
while True:
    label0.setText(str(num))
    wait(1)
    num = (num if isinstance(num, int) else 0) + num
    wait_ms(2)
```



Used to hold a user settable value

Common Equation,



Forms an equation of values places in the two positions in the block. These values can be constants or values returned from sensors.

Available options are:

- Addition,
- Subtraction,
- Multiplication,
- Devision,
- Power,

Special Values,

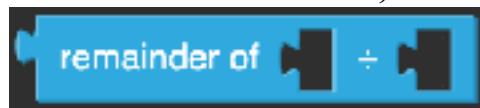


Allows calculations to use special values.

Special values are:

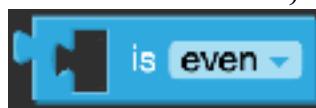
- Pi (3.14),
- Eulars Number (2.718),
- Golden Ration (1.618),
- Square root 2 (1.414)
- Square Root 1/2 (0.707),
- infinity.

Remainder of,



Returns the remainder of one value divided by another value.

Value is even,



Used to decide when a value is Even or Odd dependent on what is set in the block.

Values are

- Even,
- Odd,

Sum of list



Returns the function from a list of values stored in a list block attached to it..
This block has several functions hidden under it which are:

- Sum - The total of all items in the list.



The Micropython code in this mode is

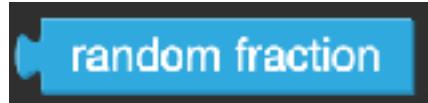
```
sum([None, None, None])
```

where [None] is a place holder for value block added to a position in the list.

- Min - The lowest value in the list.
- Max - The highest value in the list.
- Average - An average value of all items in the list.
- Median - The middle value from a list.
- Mode - The most common value in a list.
- Standard Deviation - The amount of variance in the values in the list.
- Random item - A random item from the list.

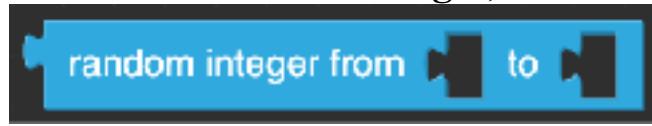
For more information on the list function see the list section.

Random Fraction



Returns a random fraction between 0 and 1.

Random Integer,



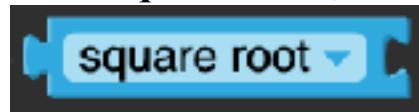
Returns a random integer from a range of user defined values or values from variables and sensors..

Round,



Rounds the number or value that follows it up or down.

Square Root,



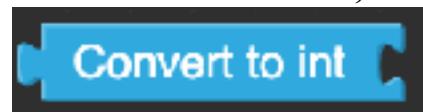
Returns the square root value, absolute value, the negation of a value, the natural logarithm of a number, a base10 logarithm of a number, e to the power of a number, or 10 to the power of a number or value connected to it.

Sin,



Returns the Sine, Cosine, Tangent, Arcsine, Arccosine, or Arctangent of a number.

Convert to int,



Converts a value to an integer.

Convert to Float,



Converts a value to a floating point number.

Reserve Decimal Fraction.



Reserves the user defined number as a decimal fraction.

Get Bit

Set Bit

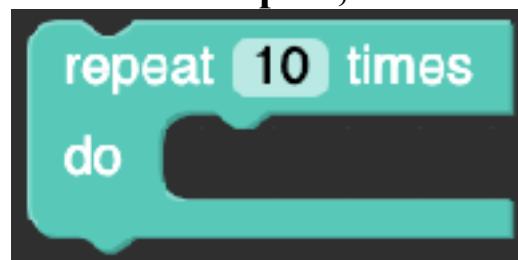
Clear Bit,

Reverse Bit,

Int From Bytes,

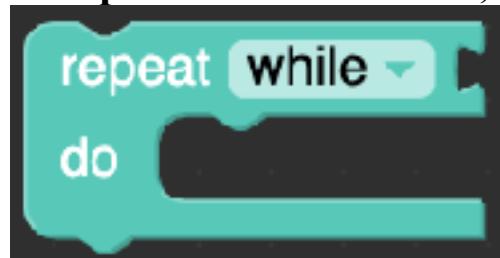
Loops

Repeat,



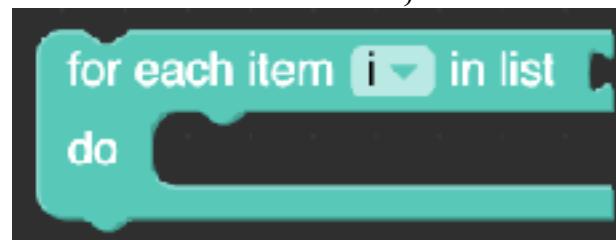
Repeats the code in side it ten times.

Repeat while Condition,



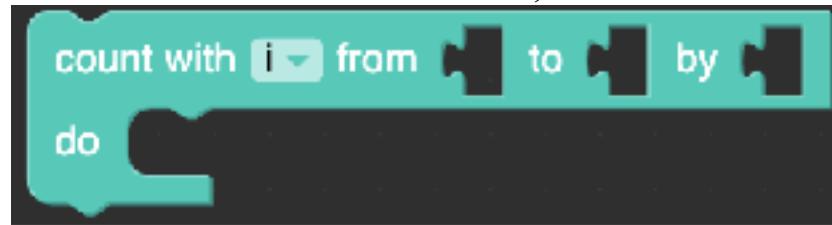
Repeats the code inside while the user defined condition is set.

For Each,



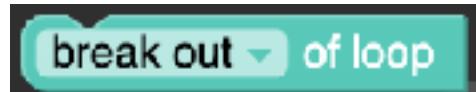
Repeats the code inside for each item in a list.

Count with,



Runs the code inside on selected items in a range using a user defined variable.

Break out,



Used to exit from a loop of code.

Logic,

If - Do,



If a defined condition is met the code inside is run.

If - Else - Do,

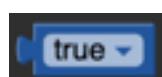


If a defined condition is met the code inside is run otherwise another set of code is run..

Try/Except

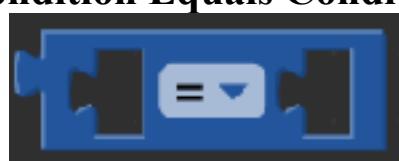


Condition True,



Returns true or false.

Condition Equals Condition,



Returns true or false if a sum two conditions are met.

Not,



Inverts or reverses the command.

Null,

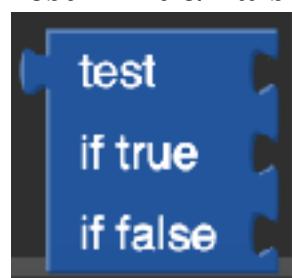


Condition and Condition,

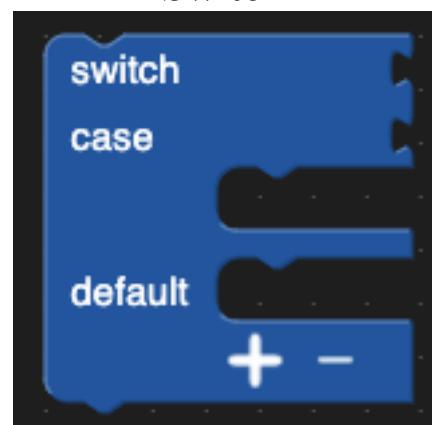


Used to define if two conditions are required.

Test - True/False



Switch



Graphic,

Lcd.clear



Lcd.clear turns all the pixels on the screen to black.

Lcd.fill



Lcd.fill will change all of the pixels on the screen to the same colour. The colour is set in this block by using the colour picker.



lcd.print

Lcd.print will place text on the screen at the specified coordinates and in the specified colour using the colour picker.

Font

Font: FONT_Default ▾

Font specifies a font to be used for the text that will be shown on screen.



Lcd.pixel

Lcd.pixel will colour specific pixel at the specified coordinate and in the specified colour using the colour picker.

Lcd.line,



Lcd.line draws a line starting at the specified coordinates and ending in X1 and Y1 in the specified colour using the colour picker. By specifying X1 and Y1 we can have angled lines.



Lcd.rectangle,

Lcd.rectangle draws a rectangle starting at the specified coordinate with a user defined height and width in the specified colour using the colour picker.

Lcd.triangle,



Lcd.triangle draws a triangle by specifying the coordinates of the three individual points of a triangle in the specified colour using the colour picker.

Lcd.circle,



Lcd.circle draws a circle with the centre point the specified coordinate with a radius defined by the user and in the specified colour using the colour picker.

Lcd.ellipse,



Lcd.ellipse draws a circle with the centre point the specified coordinate with an X radius and separate Y radius defined by the user and in the specified colour using the colour picker.

Lcd.arc,



Lcd.arc draws an arc with the centre at the specified coordinates with a user defined radius, thickness, start point and end point and in the specified colour using the colour picker.

Please Note that there is a bug in the code that keeps deleting the Radius value.



Lcd.polygon

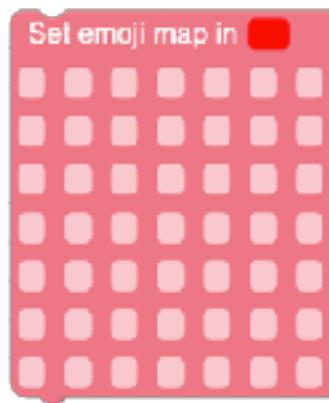
Lcd.polygon draws a polygon with the centre at the specified coordinates with a user definable radius, number of sides, thickness, rotation and in the specified colour using the colour picker. Please note that when Radius and thickness are the same, we can get a snowflake like appearance.

Emoji,

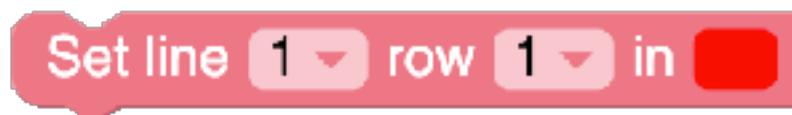
The Emoji menu contains blocks that allow us to draw Emojis and control the background shown behind them.

There are three blocks available here with the biggest being the emoji map

Emoji Map



The emoji map has a 7 X 7 grid which you click on each cell to make it light up in the specified colour using the colour picker on the right of the text. To make a cell active, all you have to do is to click on each of the lighter blocks and a tick will appear to show that you have made that cell active.



Set line 1 row 1 in colour

This block allows you to set the colour of each individual cell.

Change Background Image.

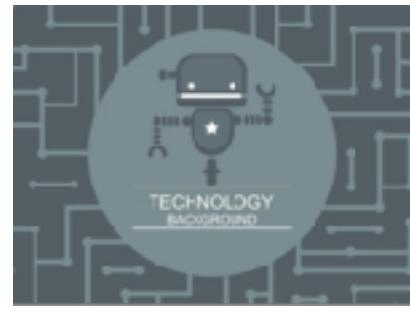
Change backgroundImage 0 ▾

Allows you to chose one of the six built in backgrounds that show behind the emoji. The six background images are shown below.

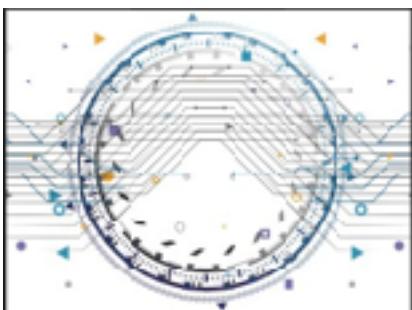
Background 0



Background 3



Background 1



Background 4



Background 2



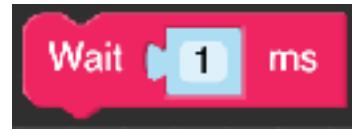
Background 5

Timers (Part 2), Wait Seconds,



Delays a program from moving on to the next block by a user defined pause in seconds.

Wait Milliseconds,



Delays a program from moving on to the next block by a user defined pause in seconds.

Get Ticks ms

Returns current runtime of esp32..



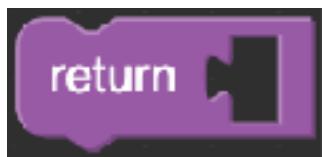
Function,
Do Something,



Creates a function with no output.
Do Something and Return Condition,



Creates a function with an output.



If Condition Return Condition,
If true, returns the condition defined in the space.

Text,



Text block

Used to display a letter, word or sting of text on the screen.



To UPPER Case

Converts the following sting to upper case.

In Text Get Letter

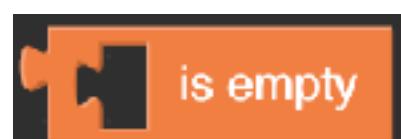


Returns one letter from the string.

Count In.



Returns how many time a string appears in another string.



Is Empty.

Returns true if string is empty.

Length Of,



Returns the length (including spaces) of the string that is connected.

Print



Prints the string or number connected to it on screen.



Replace With In

Finds and replaces all occurrences of a string in a string with another string.
Trim Spaces,



Returns the connected sting of txt with some or all of the spaces removed.

Prompt For Text With Message



Prints a message on screen prompting a user to insert some text.



Prompt For Text With Message ()

Prints a message on screen prompting a user to insert some text or value connected to it.



Convert To String,

Converts the values in the following blocks in to a string.

Message Plus (Concat String.)



Used to add a value to the text for example batch file naming (ABCD + 001/002/003)
Decode



Used to decode a sting of text.

Encode,



Used to encode a string of text.

Examples

The following examples are used to edit the text in labels or to log data.

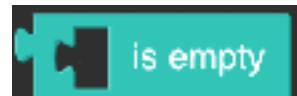
Lists,



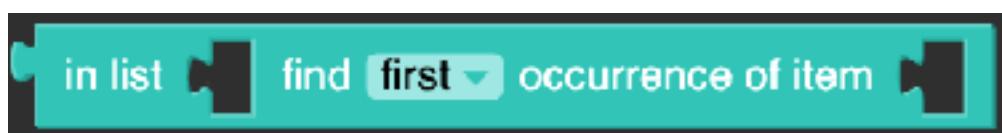
Length Of,

Returns the length of a list.

IsEmpty,



Returns true if list is empty.



In List Find Occurrence

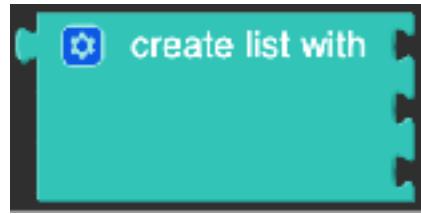
Returns index of first or last occurrence of a specified item or returns zero if an item is not found.



Create Empty List,

Creates an empty list.

Create List With,

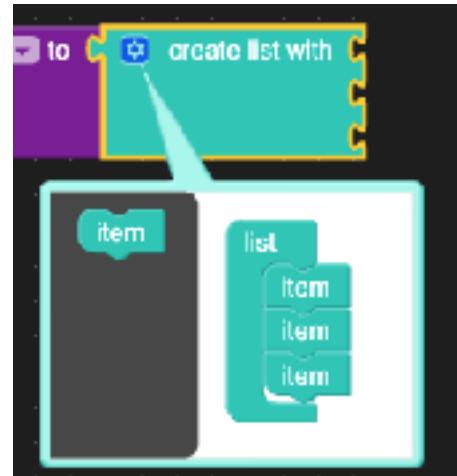


The Create List With block can not be used on its own as it returns values. In order to use this block we need to connect it with other blocks.

The MicroPython code for this block is

= [None, None, None]

In order to add more items to the list you need to click on the gear shape in the top left corner to bring up the dialog shown in the image and then drag Item block to the list to increase the list size.



In the following example I am using the Create List With block to make a list of reading from the StickC's IMU in order to get an average reading that can be passed to other code.



The Micropython code for this example is as follows

```

from m5stack import *
from m5ui import *
from uiflow import *
import imu

setScreenColor(0x111111)

imu0 = imu.IMU()
label0 = M5TextBox(16, 76, "Text", lcd.FONT_Default,0xFFFFFFF, rotate=0)
label1 = M5TextBox(20, 107, "Text", lcd.FONT_Default,0xFFFFFFF, rotate=0)

from numbers import Number

Avg = None

def math_mean(myList):
    localList = [e for e in myList if isinstance(e, Number)]
    if not localList: return
    return float(sum(localList)) / len(localList)

while True:
    label0.setText(str(imu0.ypr[1]))
    Avg = math_mean([imu0.ypr[1], imu0.ypr[1], imu0.ypr[1], imu0.ypr[1], imu0.ypr[1],
                    imu0.ypr[1], imu0.ypr[1], imu0.ypr[1], imu0.ypr[1], imu0.ypr[1]])
    label1.setText(str(Avg))
    wait_ms(2)

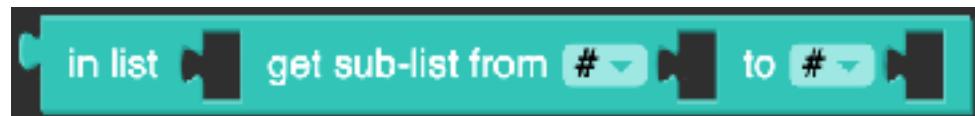
```

In List Get



Returns the item in the specified position of a list.

In List Get Sub List,

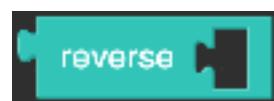


Creates a list from a specified section of another list.



Create List With Item Repeated.

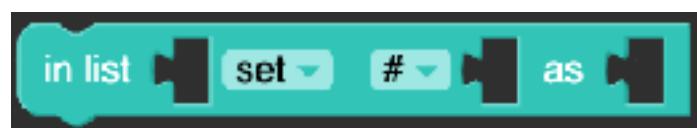
Creates a list with a user defined item repeated a user defined amount of times.



Reverse,

Reverses a copy of a list.

In List Set,



Allows users to define an item at a specific position in a list.

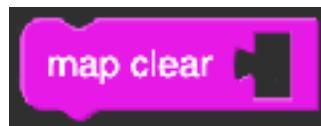


Make List From List With Delimiter.

Allows users to split a list into smaller list using delimiters.

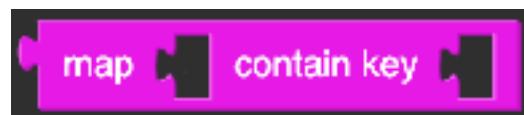
Map

Create Map,

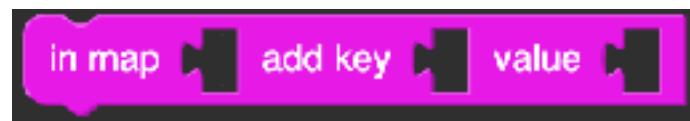


Map Clear,

Map Contain Key,

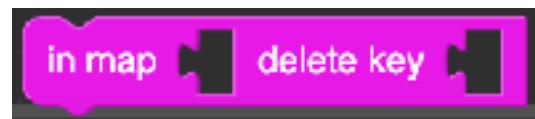
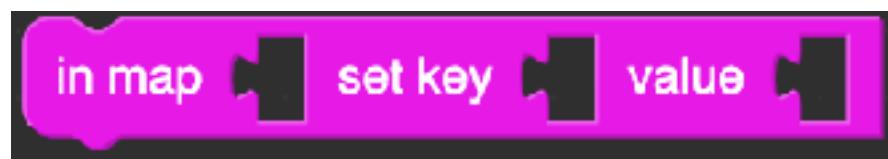


Get Key In Map,



In Map Add Key

In Map Set Key,



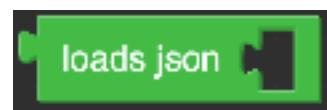
In Map Delete Key,

JSON,



Dump to JSON

Dumps a sting or value to json.



Load JSON

Used to load a JSON value or string.

Advanced Blocks.

The Advanced blocks available in UIFlow provide lower level access to hardware as well as access to hardware that does now have dedicated blocks provided. This section also contains blocks required for accessing the SDCard and files stored on it.

SDCard

Open SDCard File



Open SDCard File block is a loop that opens a file stored on the cards, completes the actions placed inside and then closes the file. By opening and closing the file after each pass, the chance of file corruption is reduced.

The argument mode points to a string beginning with one of the following sequences (Additional characters may follow these sequences.):

``r'' Open text file for reading. The stream is positioned at the beginning of the file.

``r+'' Open for reading and writing. The stream is positioned at the beginning of the file.

``w'' Truncate file to zero length or create text file for writing. The stream is positioned at the beginning of the file.

``w+'' Open for reading and writing. The file is created if it does not exist, otherwise it is truncated. The stream is positioned at the beginning of the file.

``a'' Open for writing. The file is created if it does not exist. The stream is positioned at the end of the file. Subsequent writes to the file will always end up at the then current end of file, irrespective of any intervening fseek(3) or similar.

The Micropython code for the loop is as follows:

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(21, 5, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

with open('/sd/', 'r')
    pass
```

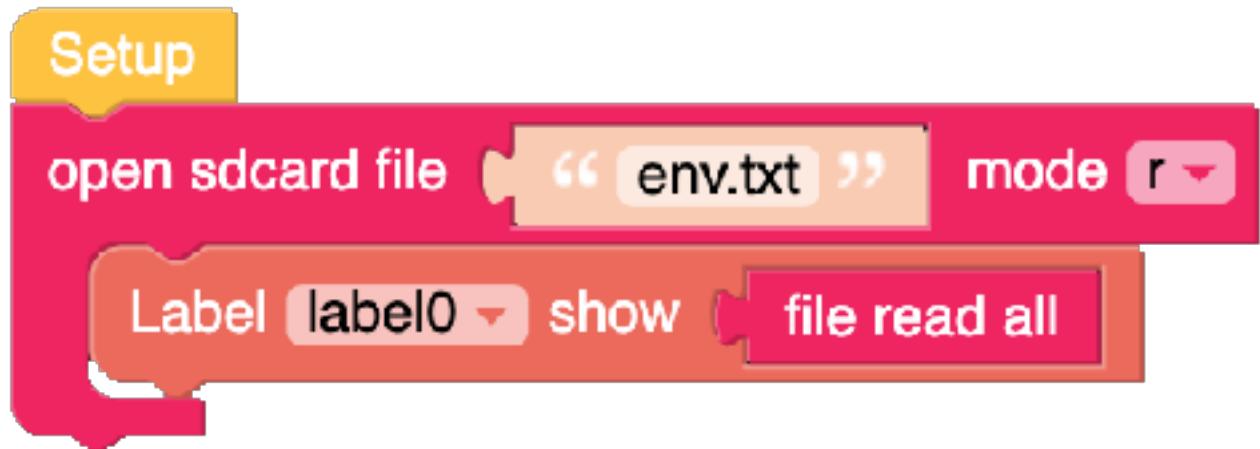

File Read all



Returns the contents of the file opened in the SD File open block.

The Micropython code for this is
`(fs.read())`

In the following example the file env.txt (created for my data logging demo), is opened and by using a Label Show block we can display the contents on the M5Stacks screen. Please note that for this to work the mode has to be set to "r" or "r+"



And the Micropython code for this is:

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(21, 5, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

with open('/sd/env.txt', 'a') as fs:
    label0.setText(str(fs.read()))
```

File Read Bytes

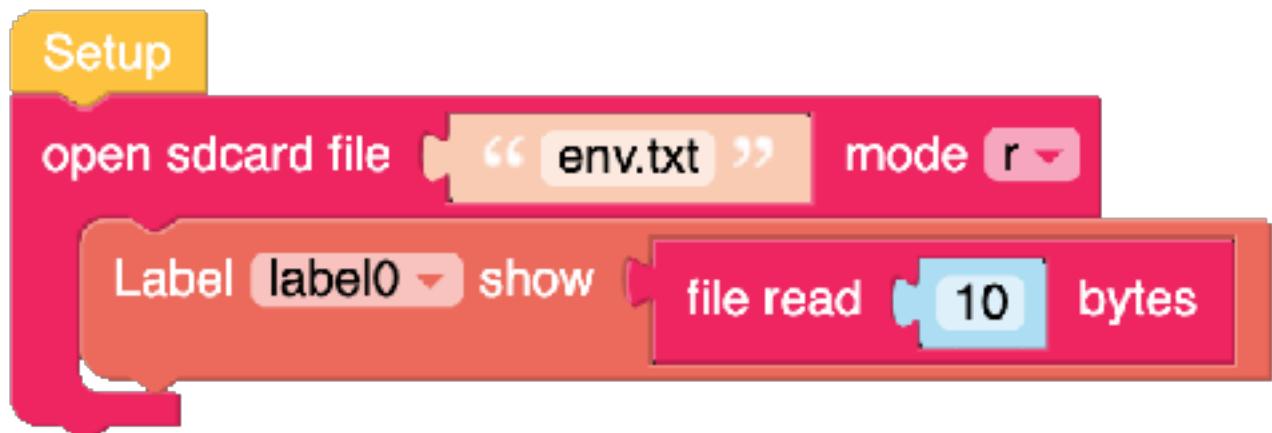


Returns the first number of bytes of a file declared in a maths number block.
The MicroPython code for this is

(fs.read(0))

Example

In the following example the first ten bytes of the file env.txt are display on the M5Stacks screen.



The MicroPython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(21, 5, "Text", lcd.FONT_Default, 0xFFFF, rotate=0)

with open('/sd/env.txt', 'a') as fs:
    label0.setText(str(fs.read(0)))
```

File Read Line

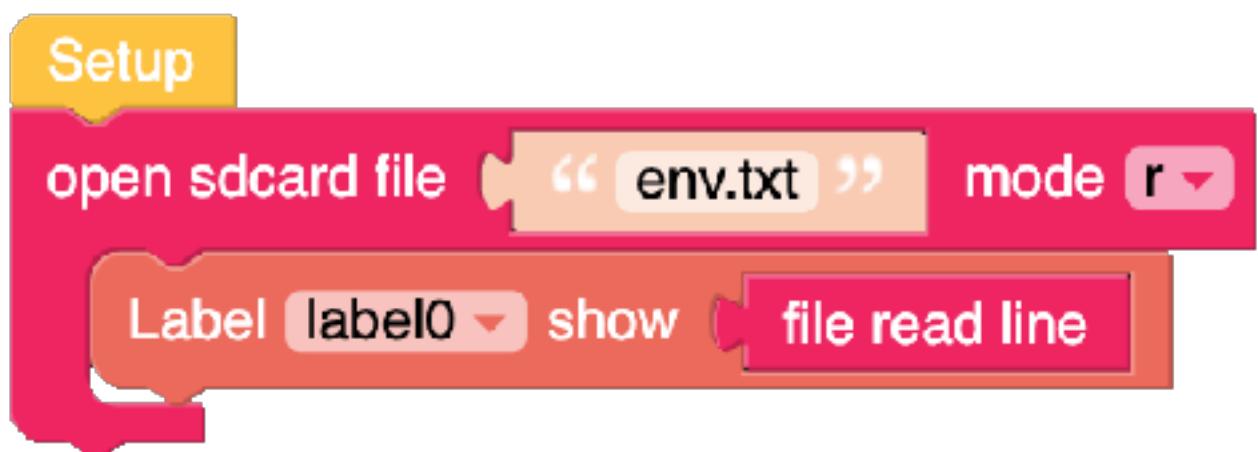


File Read Line is a value block that returns a line of text from a file.

The Micropython code for this is

fs.readline()

Example



In this example the first line of the text file env.txt is displayed on the M5Stacks screen.

The Micropython code for this is

```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(131, 47, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

with open('/sd/env.txt', 'r') as fs:
    label0.setText(str(fs.readline()))
```

File Write

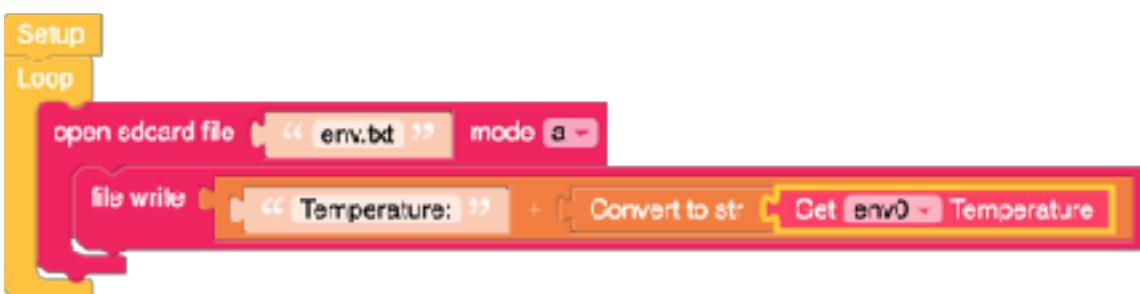


The File Write block is used to write data into a file.

The Micropython code for this is

```
fs.write("")
```

Example



In this example I have created a simple data logging device that records the temperature recorded to the text file env.txt.

The Micropython code for this is as follows.

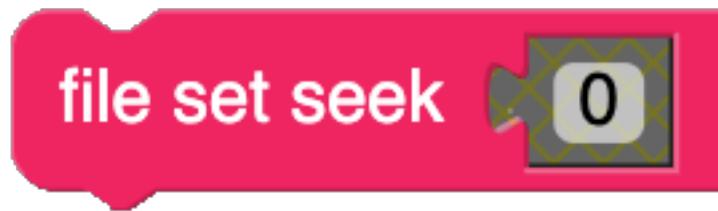
```
from m5stack import *
from m5ui import *
from uiflow import *
import unit

setScreenColor(0x222222)
env0 = unit.get(unit.ENV, unit.PORTA)

label0 = M5TextBox(21, 5, "Text", lcd.FONT_Default, 0xFFFF, rotate=0)

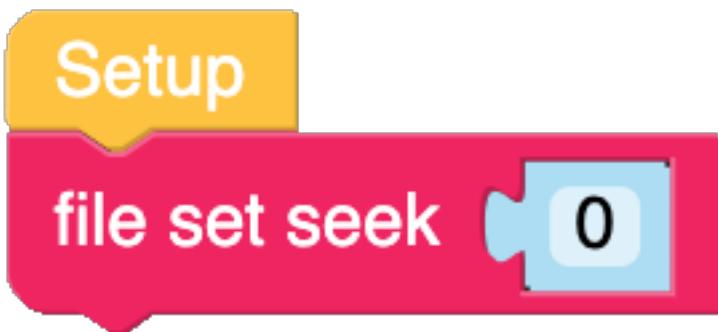
while True:
    with open('/sd/env.txt', 'a') as fs:
        fs.write(str('Temperature:' + str((env0.temperature)))))
    wait_ms(2)
```

File Seek



File Seek looks for the file in the position set with a mathematical value block.
The Micropython code for this is
fs.seek(0)

Example



In this demo the file is position 0 is read from the SD card.

The Micropython code for this is

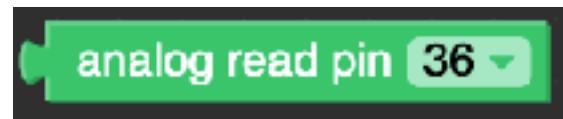
```
from m5stack import *
from m5ui import *
from uiflow import *
```

```
setScreenColor(0x222222)
```

```
label0 = M5TextBox(131, 47, "Text", lcd.FONT_Default,0xFFFF, rotate=0)
```

```
fs.seek(0)
```

**Easy I/O,
Analog Read Pin,**

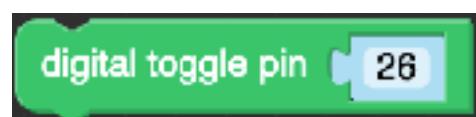


Analogue Read Pin,

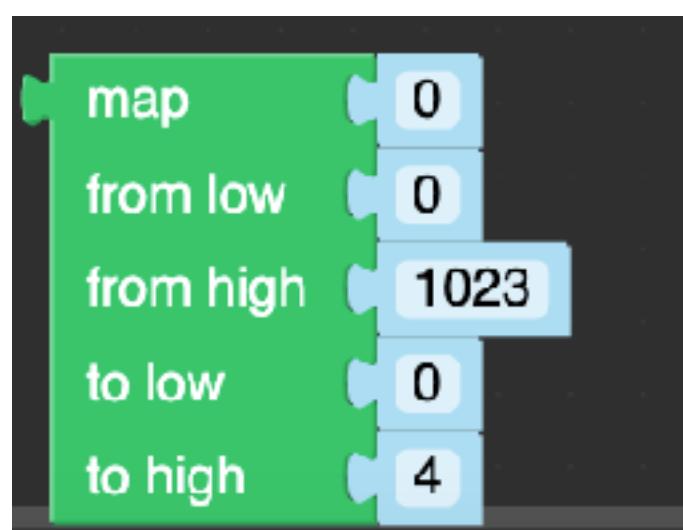


Digital Read,

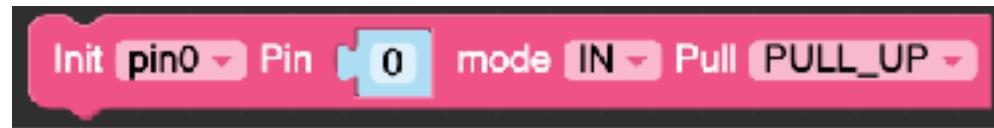
Digital Write Pin,



Digital Toggle Pin,



Map,
GPIO,



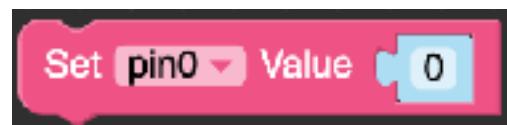
Init Pin Mode,
Set Pin High,



Set Pin Low,
Get Pin Value,

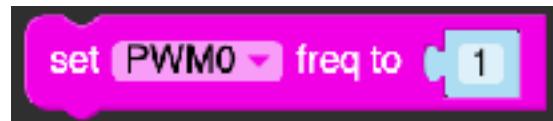


Set Pin Value,



PWM,

Init PWM of Pin,



Set PWM Frequency,

Set PWM Duty Cycle,

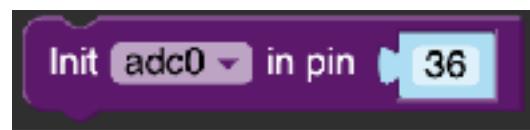


Pause PWM



Resume PWM

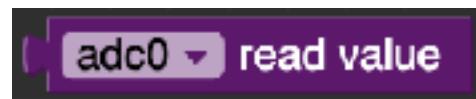
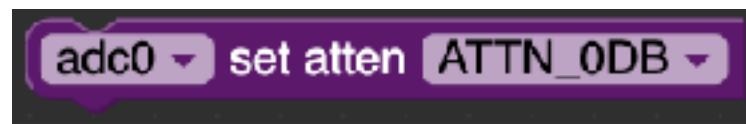
ADC,



Init ADC Pin,



Set ADC Pin Bit Width,
Set ADC Atten,



Read ADC,

DAC,



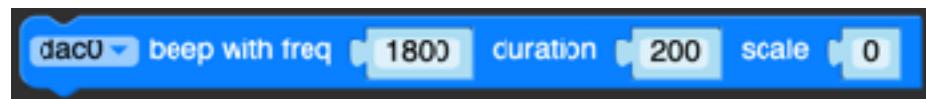
Init DAC



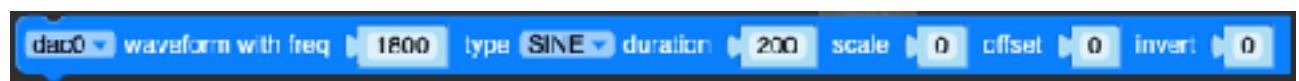
Set DAC Frequency,



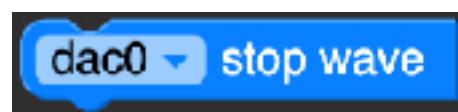
Write Value to DAC,



DAC Beep With Frequency,
DAC Waveform,



DAC Stop Wave



UART,

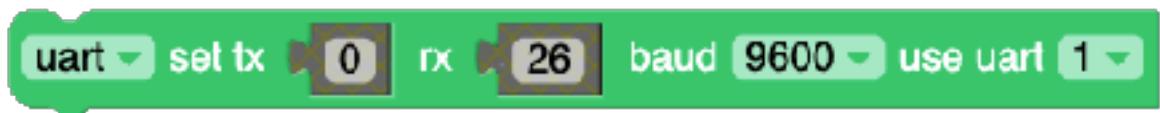
UART stands for Universal Asynchronous Receiver/Transmitter. It's not a communication protocol like SPI and I2C, but a physical circuit in a micro controller, or a stand-alone IC. A UART's main purpose is to transmit and receive serial data. The UART allows us to communicate with devices over RS232 or RS485.

Set Uart,



When using the RS485 Unit the above settings are used however, when using the RS485 Hat, the following setting need to be used.

The Micropython code for this block is



```
from m5stack import *
from m5ui import *
from uiflow import *

setScreenColor(0x222222)

label0 = M5TextBox(21, 5, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

uart = None

uart = machine.UART(1, tx=0, rx=26)
uart.init(9600, bits=8, parity=None, stop=1)
```

Read Uart,

Returns data that is sent from the slave to the master unit.



Read Uart (0) Characters,

Read a Line of Uart,



Get Remain Cache,



Write Value to UART

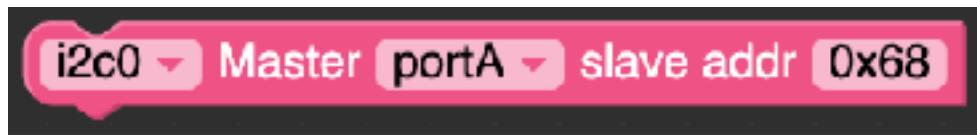


Write a Line to UART

Write String to UART,



I2C, Set I2C Port,



Starts communication with an I2C device with the address.

Set I2c SDA, SCL,



Sets the pins that the I2C port will use for SDA and SCL.

I2C Scan,



Returns a list of I2C addresses detected on the I2C bus. This block returns the addresses as binary instead of hexadecimal.

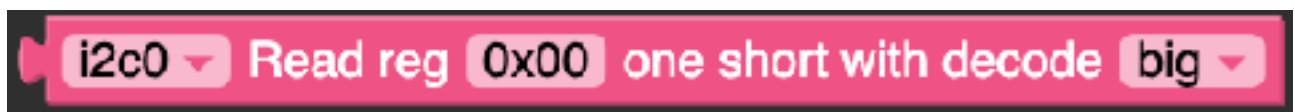
I2C Available Address in List,



I2C Read Reg One Byte,



I2C Read Reg One Short,



I2C Read Reg One Byte



I2C Write Reg One Byte,



I2C Read Byte,



I2C Write Reg One Short,



EXECUTE

Execute,
Execute Code,

Execute code: 

The Execute code block has two functions in UIFlow.

The First Function of this block is to add Micropython code to our programs that we don't have code for, and the second function allows us to add comments to our code for documenting the code.

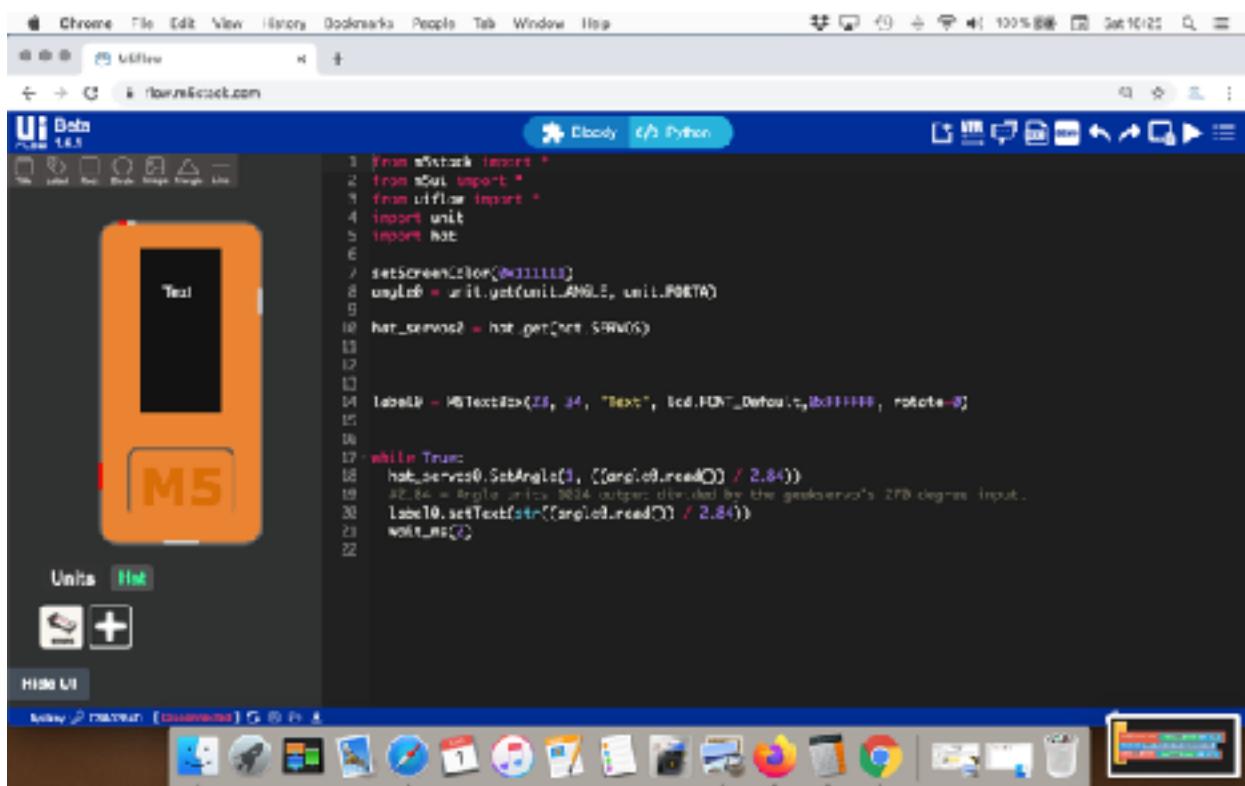
Code Mode.

Documentation Mode.



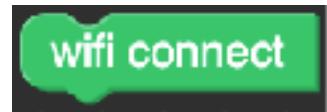
In order to use the **Execute Code** block to add comments to our programs, we need to add the hash symbol to the beginning. The hash symbol looks like this # and you can see it in the screen capture above.

When viewed in Micropython mode, the code look like this:



Because we placed the "#" symbol at the beginning of the line, UIFlow converts this to a comment and the line takes on a dark gray colour to show that this is a comment and that the line is not to be compiled or run by our program.

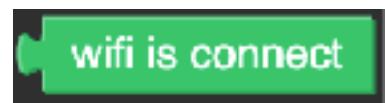
Network,



WIFI Connect,
Wifi Reconnect,



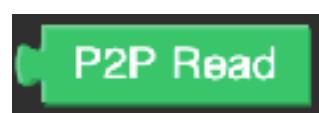
Wifi is Connected,



Connect to Wifi SSID, Password,
P2P Send API Key,



P2P Read,



ESP NOW

ESP NOW

ESP Now is a short-range, low-power communication protocol that enables multiple devices to communicate without or without Wi-Fi. This protocol is similar to the low-power 2.4GHz wireless connection found in wireless mice and keyboards. Before communications can be initiated between devices, they need to be connected in a peer to peer, master to slave configuration using the devices M.A.C addresses. Once the connection is established, these paired devices remain connected together without the use of a handshake protocol.

Get mac Address



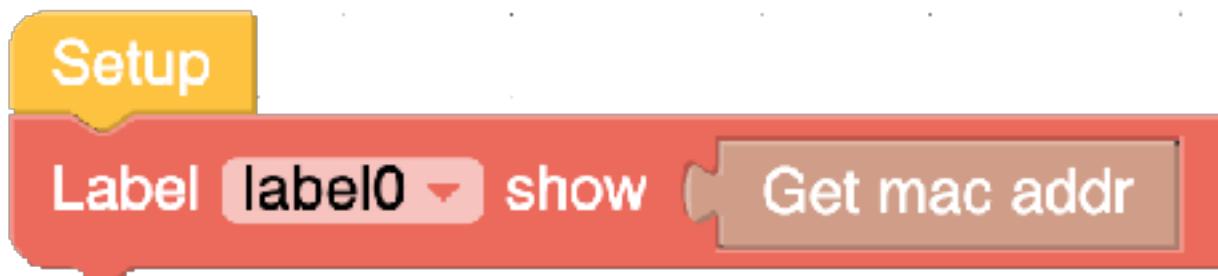
To get the mac addresses of M5Stack and M5Stick devices, we need to use the **Get mac addr** block. This block returns the M5Stack or M5Stick M.A.C address that is stored in the internal memory of the device as a value to be used by function blocks.

The MicroPython code for this block is :

```
espnow.get_mac_addr()
```

Example

As a variable block the **Get mac addr** requires additional blocks to be used. In the following example I have used the **Label** block to get the mac address and display it on the M5Stack or M5Sticks screen.



The Micropython code for this is:

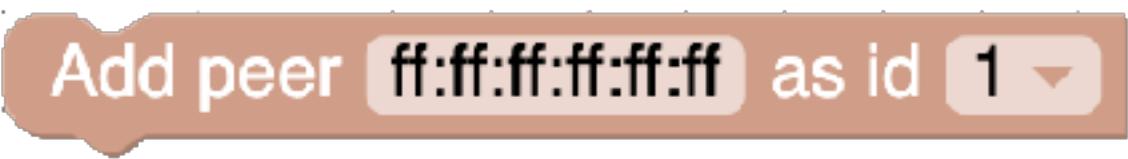
```
from m5stack import *
from m5ui import *
from uiflow import *
import espnow
import wifiCfg

setScreenColor(0x111111)

wifiCfg.wlan_ap.active(True)
espnow.init()
label0 = M5TextBox(21, 80, "Text", lcd.FONT_Default,0xFFFF, rotate=0)

label0.setText(str(espnow.get_mac_addr()))
```

Add Peer

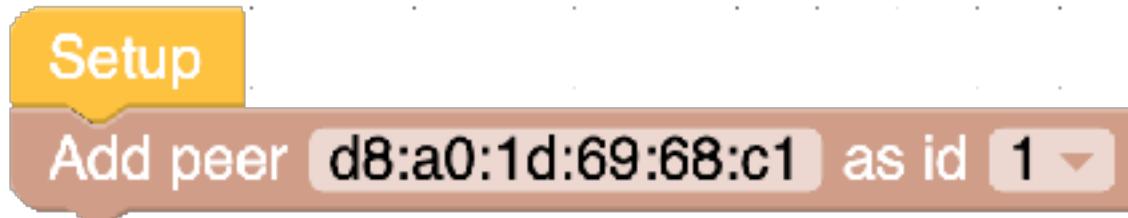


Add peer ff:ff:ff:ff:ff:ff as id 1 ▾

Now that we can see the individual device MAC address' we need to connect them together. To do this we next use the Add Peer block. To set the slave device that a primary device we will control, we replace **ff:ff:ff:ff:ff:ff** with the slave devices MAC address and then add this to master devices code setup. We can add multiple slaves to our network, but for each we need to set a different ID number. The Micropython code for this block is as follows.

```
espnow.add_peer('ff:ff:ff:ff:ff:ff', id=1)
```

Example 1



Setup
Add peer d8:a0:1d:69:68:c1 as id 1 ▾

ESP NOW

In this example I have taken the six hexadecimal value M.A.C address from the slave device and added it to the setup of the master device and set its id value as one.

The Micropython code for this example is:

It is worth noting that not only do these examples call the **espnow** library, they also call the **wifiCfg** library required for handling Wifi actions.

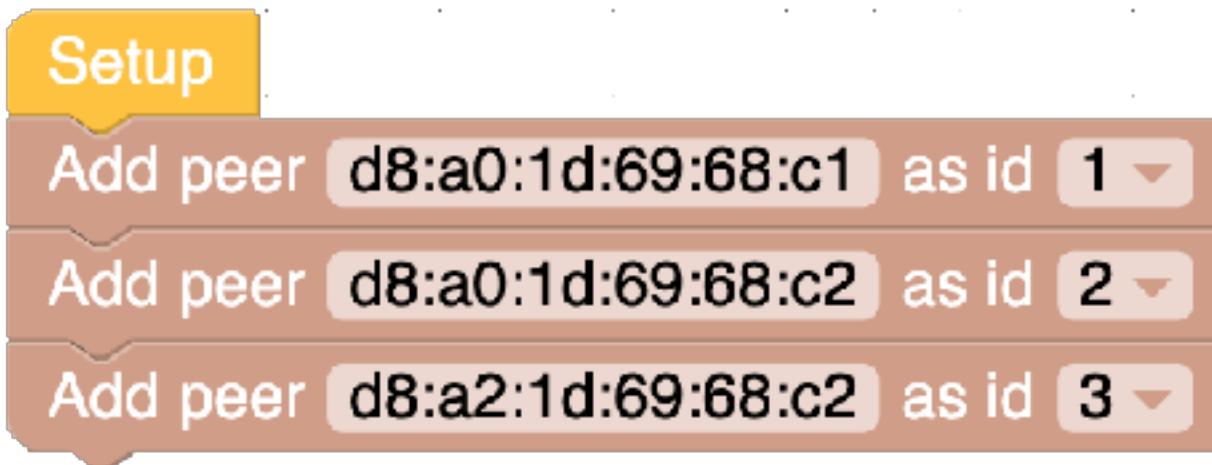
```
from m5stack import *
from m5ui import *
from uiflow import *
import espnow
import wifiCfg

setScreenColor(0x222222)

wifiCfg.wlan_ap.active(True)
espnow.init()

espnow.add_peer('ff:ff:ff:ff:ff:ff', id=1)
```

Example 2



If using more than one slave then the setup will look as follows.

When used this way we can control up to nine slave devices from one master device. The Micropython code shown on the following page show how the different MAC and ID values are changed.

```

from m5stack import *
from m5ui import *
from uiflow import *
import espnow
import wifiCfg

setScreenColor(0x222222)

wifiCfg.wlan_ap.active(True)
espnow.init()

espnow.add_peer('d8:a0:1d:69:68:c1', id=1)
espnow.add_peer('d8:a0:1d:69:68:c2', id=2)
espnow.add_peer('d8:a2:1d:69:68:c1', id=3)

```

Set PMK



Set pmk

The setup I have used for initiating a connection between devices so far is okay for none secure connections but, if we want to secure the connection between devices we need to use the **SET PMK** block. The **SET PMK** or **Set Primary Master Key** block that the M5Stack will use while communicating.

The Micropython code for this is

```
espnow.set_pmk("")
```

This block is used to define a custom, user defined sixteen byte value. If no custom value is defined then the ESP will use a default PMK value. The Set PMK block should be used before adding peers or slave units. For most programmers we wont need to use this block for basic work, but there will be times when this block will have its use.

Broadcast Data



Now that we have established a connection between M5Stack and M5Stick devices it time to send some information. The **Broadcast Data** block is used to send a string across the connection however, this block will send the data to all devices connected to the network.

The Micropython code for this block is:

```
espnow.broadcast(data=str(''))
```

Example

In the following code section, when uploaded to a master device will send the message "Hello M5Stack to any connected slave devices.



The Micropython code for this snippet is as follows.

```
def buttonA_wasPressed():
    # global params
    espnow.broadcast(data=str('Hello M5Stack'))
    pass
btnA.wasPressed(buttonA_wasPressed)
```

Send Message ID with Data



Send message id 1 with data “ ”'

To send data to a master device to a specific slave device in the network, we use the **Send Message ID with Data** block. The Micropython code for this block is

```
espnow.send(id=1, data=str(''))
```

In the following example I have just replaced the **Broadcast Data block** with the **Send Message Id With Data** block in order to send messages to only one unit.

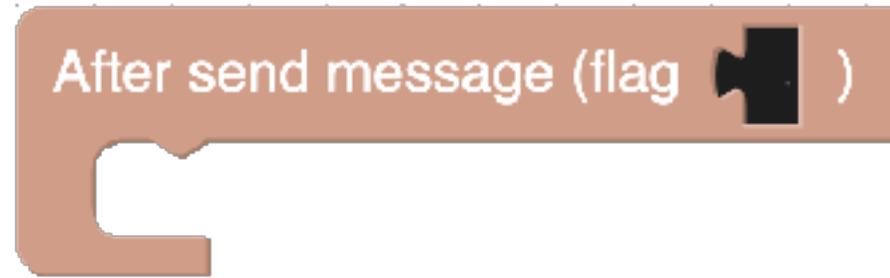


Button A wasPressed
Send message id 1 with data “ ”'

And the code sample for this block is:

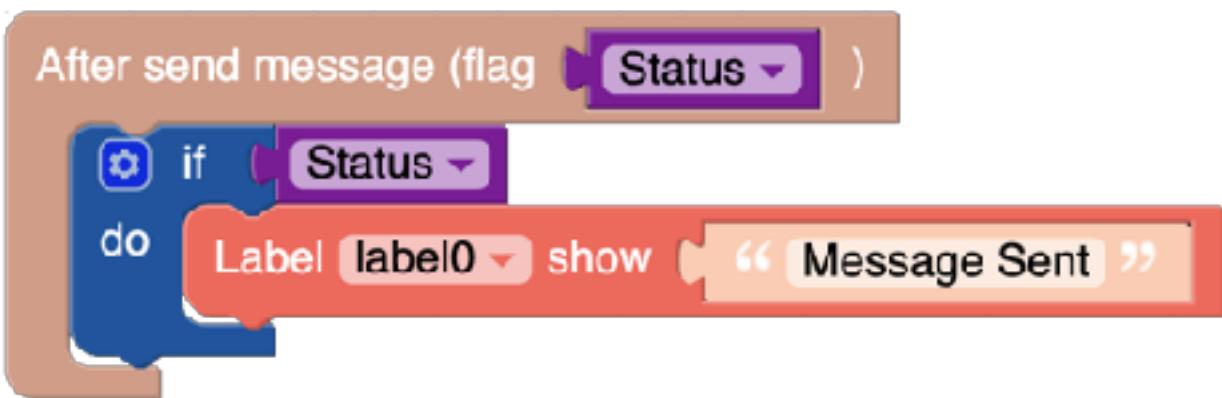
```
def buttonA_wasPressed():
    # global params
    espnow.send(id=1, data=str(''))
    pass
btnA.wasPressed(buttonA_wasPressed)
```

After Send Message Flag



So far the block give no feedback to the sender if the messages and data have been sent. In order to get feedback, we use the Send Message Flag loop block. Behind the scenes this block checks to see if a message has been sent. If the message has then the loop will receive a "True" response and run the code placed inside. If the message hasn't been sent, the block receives a "False" and waits until a true is sent to it.

Example



In this example I have used a Label block to show on the screen if a message has been sent or not. The Micropython code same for this loop is as follows.

```
wifiCfg.wlan_ap.active(True)
espnow.init()
label0 = M5TextBox(21, 54, "Text", lcd.FONT_Default,0xFFFF, rotate=0)
```

Status = None

```
def send_cb(flag):
    global Status
    Status = flag
    if Status:
        label0.setText('Message Sent')

    pass
espnow.send_cb(send_cb)
```

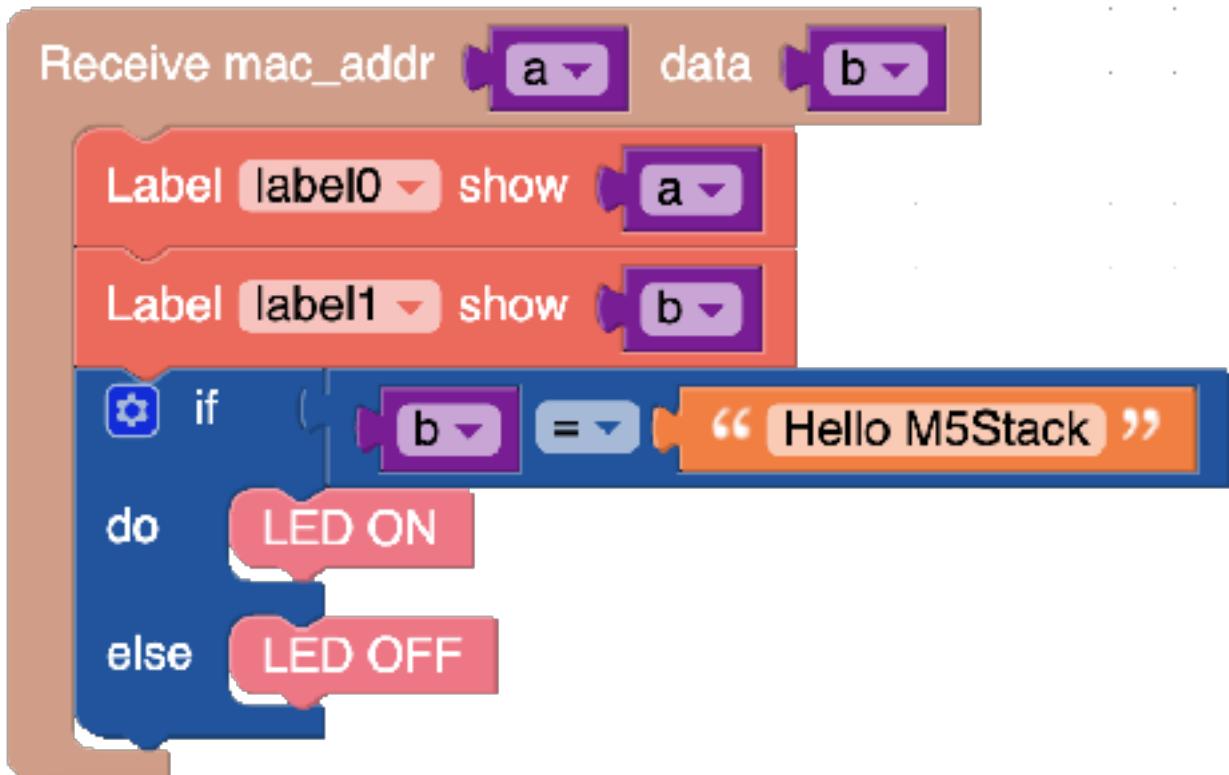
Receive MAC Address Data



So far I have shown the blocks required to send messages and data from a master device to a slave device. The last block is used on the slave device to receive messages and perform an action based on the received message. The **Receive MAC Address Data** loop receives the sent data and also the MAC address sent by the master device.

Variable must be declared for the two value blocks and "addr" and "data" can not be used as variable names.

Example



In the following example which must be run on a slave device, the LED will be lit up if the slave device receives the Hello M5Stack message sent in the earlier examples. The MicroPython code for this example is as follows.

ESP NOW

```
from m5stack import *
from m5ui import *
from uiflow import *
import espnow
import wifiCfg

setScreenColor(0x111111)

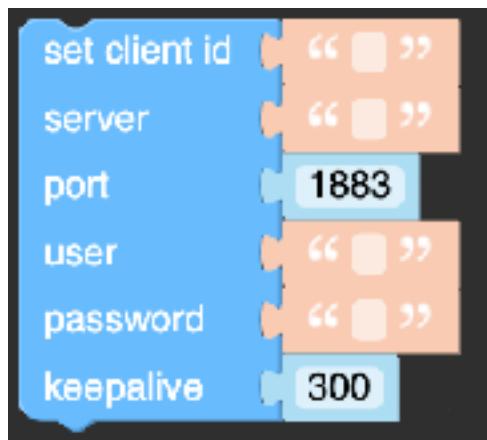
wifiCfg.wlan_ap.active(True)
espnow.init()
label0 = M5TextBox(21, 54, "Text", lcd.FONT_Default,0xFFFFFFF, rotate=0)
label1 = M5TextBox(18, 86, "Text", lcd.FONT_Default,0xFFFFFFF, rotate=0)

a = None
b = None

def recv_cb(_):
    global a,b
    a, _, b = espnow.recv_data(encoder='str')
    label0.setText(str(a))
    label1.setText(str(b))
    if b == 'Hello M5Stack':
        M5Led.on()
    else:
        M5Led.off()

    pass
espnow.recv_cb(recv_cb)
```

MQTT,
MQTT Setup,



This MQTT Setup block contains the credentials required to connect to a MQTT service.



MQTT Subscribe,

This loop subscribes to an MQTT topic and reruns the code inside keeping the M5Stack or Stick subscribed to the topic.



MQTT Start,

Starts the MQTT tasks.

Get Topic Data,



Returns information from the subscribed MQTT topic.

Publish Topic,



Publishes a message to the selected topic.

Remote Control

The remote folders contain blocks dedicate to allowing one M5Stack or Stick to control another or allow none M5Stack products control.

Update: As of version 1.5.0, the Remote functions have been removed as the firmware is now using mainstream Micropython.

Currently there are two folders for remote access as one contains test version of the existing blocks.

Remote,



Remote QR code block display a QR code on the M5Stacks screen pointing to the remote control hosted page on flow.m5stack.com.



Add Remote Switch Button [SwitchName]



Add Remote Switch,

Add Remote Button [ButtonName]



Add Remote Button,
Creates an On/Off button

Add Remote Slider [SliderName]



Add Remote Slider,

Creates a slider which alters the value of a variable "X". This can be used as a brightness control or for controlling separate colour channels.



Add Remote Other,

Custom Blocks.

The custom block menu does not contain blocks by default but can be used to load any blocks created in the custom block editor.

Micropython API's

Micropython API's

In the following pages you will find a list of the Micropython API's decided up by the category's of use. For more information on each of the API's see the related Blockly sections of the book.

Advanced API's

Advanced

Easy IO

```
from easyIO import *
# pin: only >= 32, esp32 GPIO number
analogRead(pin)
# write 38KHz Pwm
# pin: esp32 GPIO number
# duty: 0.0 ~ 100.0
analogWrite(pin, duty)
# pin mode: INOUT, Pull up
# pin: esp32 GPIO number
digitalWrite(pin, value)
# pin mode: INOUT, Pull up
# pin: esp32 GPIO number
# return: 0 or 1
digitalRead(pin)
# pin: esp32 GPIO number, toggle pin digital value
toggleIO(pin)
# Map value from a range to another Scale up or down value
map_value(value, from_low, from_high, from_low, from_high)
```

Pin

```
import machine
# pin: esp32 GPIO number
# mode: machine.Pin.(IN, OUT, OUT_OD, INOUT, OPEN_DRAIN)
# pull: machine.Pin.(PULL_UP, PULL_DOWN, PULL_HOLD, PULL_FLOAT)
# value: 0, 1
pin0 = machine.Pin(pin [, mode, pull, value])
# set pin high level
pin0.on()
# set pin low level
pin0.off()
# get pin state
pin0.value()
# state: 0, 1
# set pin level to state
pin0.value(state)
```

Pwm

```
import machine
# pin: esp32 GPIO number
# freq: 1 - 40000000Hz
# duty: 0 ~ 100.0 (%)
# timer: 0, 3(servo unit use 3), (lcd, ir unit use 1) (speak use 2)
pwm0 = machine.PWM(pin, freq, duty, timer)
# freq: 1 ~ 40000000Hz
pwm0.freq(freq)
# duty: 0 ~ 100.0 (%)
pwm0.duty(duty)
# pause pwm out
pwm0.pause()
# resume pwm out
pwm0.resume()
```

I2C

```
import i2c_bus
# port i2c_bus.PORTA, i2c_bus.PORTB or (sda, scl)
i2c1 = i2c_bus.I2C(port, addr)
i2c1.write_u8(reg, data)
# byteorder: 'big' or 'little'
i2c1.write_u16(reg, data, byteorder='big')
#
data = i2c1.read_u8(reg):
# byteorder: 'big' or 'little'
data = i2c1.read_u16(reg, byteorder='big')
# data is bytes
data = i2c1.read(num)
# state: True or False
state = i2c1.available()
# addrList is a available addr list
addrList = i2c1.scan()
```

Advanced API's

ADC

```
import machine
# pin number must > 36
adc=machine.ADC(pin)
# value: ATTN_0DB (range 0 ~ 1.1V)
#     ATTN_2_5DB (range 0 ~ 1.5V)
#     ATTN_6DB (range 0 ~ 2.5V)
#     ATTN_11DB (range 0 ~ 3.9V)
adc.attenuation(value)
# value: WIDTH_9BIT - capture width is 9Bit
#     WIDTH_10BIT - capture width is 10Bit
#     WIDTH_11BIT - capture width is 11Bit
#     WIDTH_12BIT - capture width is 12Bit
adc.width(value)
# Read the ADC value as voltage (in mV)
data = adc.read()
```

DAC

```
import machine
dac = machine.DAC(25)
# set the dac value: 0 ~ 255
dac.write(value)
# freq: 16-32000 Hz
# duration: the duration of the beep in ms
# scale: range: 0-3; scale the output voltage by 2^scale
dac.beep(freq, duration [, scale=0])
# freq: 16-32000 Hz for sine wave
#     500-32000 Hz for noise
#     170 - 3600 Hz for triangle
#     170 - 7200 Hz for ramp and sawtooth
# type: SINE, TRIANGLE, RAMP, SAWTOOTH and NOISE
# scale: 0~ 3, scale the output voltage by 2^scale
# offset: optional, only valid for sine wave; range: 0-255; offset the output voltage by offset value
# invert: optional, only valid for sine wave; range: 0-3; invert the half-cycle of the sine wave
dac.waveform(freq, type [,duration=0] [,scale=0] [,offset=0] [,invert=2])
# Stops the background process started by dac.waveform()
dac.stopwave()
# Changes the frequency of the background waveform process started by dac.waveform()
# function.
dac.freq(freq)
```

MQTT

Mqtt lib support auto reconnect Wi-Fi, MQTT server and restart subscribe

Mqtt lib will auto ping broker **every 10000ms**

Mqtt lib will auto connect Wi-Fi

```
from m5mqtt import M5mqtt
```

```
m5mqtt = M5mqtt(id, server, port, user, password, keepalive)
```

```
def callback(topic_data):
```

```
    # global params
```

```
    pass
```

```
m5mqtt.subscribe(topic, callback)
```

```
# Must be called after subscription
```

```
# after start can't subscribe again
```

```
m5mqtt.start()
```

```
m5mqtt.publish(topic, msg)
```

ESP-NOW

```
import espnow
```

```
# init espnow
```

```
# mac ff:ff:ff:ff:ff add peer in channel 1, IF_WIFI_STA
```

```
espnow.init()
```

```
# deinit espnow
```

```
espnow.deinit()
```

```
# return mac addr,type: bytes
```

```
espnow.get_mac_addr()
```

```
# peer: must be bytes, len: 6, slave mac addr
```

```
# channel: 1 ~ 13, default 1
```

```
# ifidx: espnow.IF_WIFI_STA or espnow.IF_WIFI_AP or espnow.IF_MAX, default:
```

```
espnow.IF_WIFI_STA
```

```
# encrypt: True or False, default: False
```

```
# lmk: len must be 16, type string or bytes, if encrypt is False, ignore, default: None
```

```
# id: save peer, 0 ~ 9, default: -1
```

```
espnow.add_peer(peer, [channel, ifidx, encrypt, lmk, id])
```

```
# pmk: type should be bytes or str, len must 16
```

```
espnow.set_pmk(pmk)
```

```
# recv msg will callback
```

```
# if msg is broadcast, broadcast = True
```

```
def recv_cb(broadcast):
```

Advanced API's

ESP-NOW (continued)

```
print(broadcast)
mac_addr, cmd_id, data = espnow.recv_data()

espnow.recv_cb(recv_cb)

# turn tuple (mac_addr, cmd id, data)
# if no data: mac_addr = b", cmd_id = 0, msg = b"
mac_addr, cmd_id, msg = espnow.recv_data()

def send_cb(send_state):
    print(send_state)

espnow.send_cb(send_cb)

# cmd id: 0~65535, default 0
# data: bytes or str, default: None
espnow.broadcast([cmd_id, data])
# id: 0 ~ 9, (by espnow.add_peer set)
# mac_addr: bytes, len 6 (if mac_addr was give, ignore id)
# cmd_id: 0~65535
# data: bytes or string
espnow.send([id, mac_addr, cmd_id, data])
```

UART

```
import machine
# Set uart port and tx rx
uart = machine.UART(channel, tx_pin, rx_pin)
# Set part baud and data bits, Parity bits, stop bits
uart.init(baud, bits, parity, stop)
# Read all data from uart
uart.read()
# Read 10 characters from uart
uart.read(10)
# Read a line data from uart
uart.readline()
# Read the rest of the cache
uart.any()
# Write characters to uart
uart.write(ch)
```

WiFi

```
import wifiCfg
# Display wifi option "reconnect" and "set wifi" on bottom
wifiCfg.screenShow()

# auto connect wifi
wifiCfg.autoConnect(lcdShow=True)

# Return true when wifi is connected
wifiCfg.wlan_sta.isconnected()

# Connect with ssid and pswd
wifiCfg.doConnect(ssid, pswd)
```

P2P

```
# Initialize P2P settings
remoteInit()

# Connect remotely APIkey and send data
sendP2PData(APIkey, data)

# Return the data of remotely host
getP2PData()
```

Display API's

Display Module

NOTE: base https://github.com/lororis/MicroPython_ESP32_psRAM_LoBo/wiki/display

Class TFT

Some [TFT examples](#) are available.

Create the TFT class instance

Before using the display, the **display** module must be imported and the instance of the TFT class has to be created:

```
from m5stack import lcd  
tft = lcd
```

Colors

Color values are given as 24 bit integer numbers, 8-bit per color.

For example: **0xFF0000** represents the RED color. Only upper 6 bits of the color component value is used.

The following color constants are defined and can be used as color arguments:

BLACK, NAVY, DARKGREEN, DARKCYAN, MAROON, PURPLE
OLIVE, LIGHTGREY, DARKGREY, BLUE, GREEN, CYAN, RED
MAGENTA, YELLOW, WHITE, ORANGE, GREENYELLOW, PINK

Drawing

All **drawings** coordinates are **relative** to the **display window**.

Initialy, the display window is set to full screen, and there are methods to set the window to the part of the full screen.

Fonts

9 bit-mapped fornts and one vector 7-segment font are included. Unlimited number of fonts from file can also be used.

The following font constants are defined and can be used as font arguments:

FONT_Default, FONT_DefaultSmall, FONT_DejaVu18, FONT_Devau24
FONT_Ubuntu, FONT_Comic, FONT_Minya, FONT_Tooney, FONT_Small
FONT_7seg

Display API's

Touch panel

Touch panels based on **XPT2066** and **STMPE610** controllers are supported at the moment.

Methods

`tft.init(type, mosi=pinnum, miso=pinnum, clk=pinnum, cs=pinnum [, opt_args])`

Initialize the SPI interface and set the various operational modes.

All arguments, except for type are KW arguments and must be entered as arg=value.

Pins have to be given as pin numbers, not the machine. Pin objects

opt_args are optional, see the table bellow.

Argument	Description
type	required, sets the display controller type, use one of the constants: ST7789, ILI9341, ILI9488, ST7735, ST7735B, ST7735R, M5STACK, GENERIC If GENERIC type is set, the display will not be initialized Python initialization function must be provided and used to initialize the display.
mosi	required , SPI MOSI pin number
miso	required , SPI MISO pin number
clk	required , SPI CLK pin number
cs	required , SPI CS pin number
spihost	optional, default=HSPI_HOST, select ESP32 spi host, use constants: HSPI_HOST or VSPI_HOST
width	<i>optional</i> , default=240, display physical width in pixels (display's smaller dimension).
height	<i>optional</i> , default=320, display physical height in pixels (display's larger dimension).
speed	<i>optional</i> , default=10000000, SPI speed for display communication in Hz. Maximal usable speed depends on display type and the wiring length
tc	optional, Touch panel CS pin number if used
rst_pin	<i>optional</i> , default=not used, pin to drive the RESET input on Display module, if not set the software reset will be used

backl_pin	<i>optional</i> , default=not used, pin to drive the backlight input on Display module, do not use if the display module does not have some kind of backlight driver, the display's backlight usually needs more current than gpio can provide.
backl_on	<i>optional</i> , default=0, polarity of <i>backl_pin</i> for backlight ON, 0 or 1
hastouch	<i>optional</i> , default=TOUCH_NONE, set to TOUCH_XPT or TOUCH_STMPE if touch panel is used
invrot	<i>optional</i> , default=auto, configure special display rotation options If not set default value for display type is used. If you get some kind of mirrored display or you can't set correct orientation mode, try to use different values for invrot : 0, 1, 2 or 3
bgr	<i>optional</i> , default=False, set to True if the display panel has BGR matrix. If you get inverted <i>RED</i> and <i>BLUE</i> colors, try to change this argument
color_bits	<i>optional</i> , default=COLOR_BITS24; Set color mode to 24 or 16 bits. Use constants COLOR_BITS16 or COLOR_BITS24 Some display controllers, like ILI9488, does not support 16-bit colors.
rot	<i>optional</i> , default=PORTRAIT; Set the initial display orientation
splash	<i>optional</i> , default=True; If set to False do not display "MicroPython" string in RGB colors after initialization

`tft.deinit()`

De initialize the used spi device(s), free all used resources.

`tft.pixel(x, y [,color])`

Draw the pixel at position (x,y).

If *color* is not given, current foreground color is used.

`tft.readPixel(x, y)`

Get the pixel color value at position (x,y).

`tft.readScreen(x, y, width, height [, buff])`

Read the content of the rectangular screen area into buffer.

If the buffer object **buff** is not given, the new string object with the screen data will be returned.

3 bytes per pixel are returned (R, G, B).

`tft.line(x, y, x1, y1 [,color])`

Draw the line from point (x,y) to point (x1,y1)

If *color* is not given, current foreground color is used.

Display API's

`tft.lineByAngle(x, y, start, length, angle [,color])`

Draw the line from point (x,y) with length *length* starting at distance *start* from center.

If *color* is not given, current foreground color is used.

The angle is given in degrees (0~359).

`tft.triangle(x, y, x1, y1, x2, y2 [,color, fillcolor])`

Draw the triangle between points (x,y), (x1,y1) and (x2,y2).

If *color* is not given, current foreground color is used.

If *fillcolor* is given, filled triangle will be drawn.

`tft.circle(x, y, r [,color, fillcolor])`

Draw the circle with center at (x,y) and radius *r*.

If *color* is not given, current foreground color is used.

If *fillcolor* is given, filled circle will be drawn.

`tft.ellipse(x, y, rx, ry [opt, color, fillcolor])`

Draw the circle with center at (x,y) and radius *r*.

If *color* is not given, current foreground color is used.

**opt* argument defines the ellipse segment to be drawn, default id 15, all ellipse segments.

Multiple segments can be drawn, combine (logical or) the values.

- 1 - upper left segment
- 2 - upper right segment
- 4 - lower left segment
- 8 - lower right segment

If *fillcolor* is given, filled ellipse will be drawn.

`tft.arc(x, y, r, thick, start, end [color, fillcolor])`

Draw the arc with center at (x,y) and radius *r*, starting at angle *start* and ending at angle *end*

The thickness of the arc outline is set by the *thick* argument

If *fillcolor* is given, filled arc will be drawn.

`tft.poly(x, y, r, sides, thick, [color, fillcolor, rotate])`

Draw the polygon with center at (x,y) and radius *r*, with number of sides *sides*

The thickness of the polygon outline is set by the *thick* argument

If *fillcolor* is given, filled polygon will be drawn.

If *rotate* is given, the polygon is rotated by the given angle (0~359)

`tft.rect(x, y, width, height, [color, fillcolor])`

Draw the rectangle from the upper left point at (x,y) and width *width* and height *height*

If *fillcolor* is given, filled rectangle will be drawn.

`tft.roundrect(x, y, width, height, r [color, fillcolor])`

Draw the rectangle with rounded corners from the upper left point at (x,y) and width **width** and

height **height**

Corner radius is given by **r** argument.

Display API's

If **fillcolor** is given, filled rectangle will be drawn.

`tft.clear([color])`

Clear the screen with default background color or specific color if given.

`tft.clearWin([color])`

Clear the current display window with default background color or specific color if given.

`tft.orient(orient)`

Set the display orientation.

Use one of predefined constants:

`tft.PORTRAIT`, `tft.LANDSCAPE`, `tft.PORTRAIT_FLIP`, `tft.LANDSCAPE_FLIP`

`tft.font(font [,rotate, transparent, fixedwidth, dist, width, outline, color])`

Set the active font and its characteristics.

Argument	Description
<code>font</code>	required, use font name constant or font file name
<code>rotate</code>	optional, set font rotation angle (0~360)
<code>transparent</code>	only draw font's foreground pixels
<code>fixedwidth</code>	draw proportional font with fixed character width, max character width from the font is used
<code>dist</code>	only for 7-seg font, the distance between bars
<code>width</code>	only for 7-seg font, the width of the bar
<code>outline</code>	only for 7-seg font, draw the outline
<code>color</code>	font color, if not given the current foreground color is used

`tft.attrib7seg(dist, width, outline, color)`

Set characteristics of the 7-segment font

Argument	Description
<code>dist</code>	the distance between bars
<code>width</code>	the width of the bar
<code>outline</code>	outline color
<code>color</code>	fill color

Display API's

`tft.fontSize()`

Return width and height of the active font

`tft.text(x, y, text [, color])`

Display the string `text` at position `(x,y)`.

If `color` is not given, current foreground color is used.

- `x`: horizontal position of the upper left point in pixels, special values can be given:
 - `CENTER`, centers the text
 - `RIGHT`, right justifies the text
 - `LASTX`, continues from last X position; offset can be used: `LASTX+n`
- `y`: vertical position of the upper left point in pixels, special values can be given:
 - `CENTER`, centers the text
 - `BOTTOM`, bottom justifies the text
 - `LASTY`, continues from last Y position; offset can be used: `LASTY+n`
- `text`: string to be displayed. Two special characters are allowed in strings:
 - `\r` CR (0x0D), clears the display to EOL
 - `\n` LF (0x0A), continues to the new line, `x=0`

`tft.textWidth(text)`

Return the width of the string `text` using the active font `fontSize`

`tft.textClear(x, y, text [, color])`

Clear the screen area used by string `text` at position `(x,y)` using the background color `color`.

If `color` is not given, current background color is used.

`tft.image(x, y, file [,scale, type])`

Display the image from the file `file` on position `(x,y)`

- **JPG** images are supported.
- Baseline only. Progressive and Lossless JPEG format are not supported.
Image size: Up to 65520 x 65520 pixels
Color space: YCbCr three components only. Gray scale image is not supported.
Sampling factor: 4:4:4, 4:2:2 or 4:2:0.
- **BMP** images are supported.
- Only uncompressed RGB 24-bit with no color space information BMP images can be displayed.
- Constants `tft.CENTER`, `tft.BOTTOM`, `tft.RIGHT` can be used for x&y
- x and y values can be negative

`scale (jpg)`: image scale factor: 0 to 3; if `scale>0`, image is scaled by factor $1/(2^{\text{scale}})$ (1/2, 1/4 or 1/8)

`scale (bmp)`: image scale factor: 0 to 7; if `scale>0`, image is scaled by factor $1/(\text{scale}+1)$

`type`: optional, set the image type, constants `tft.JPG` or `tft.BMP` can be used. If not set, file extension and/or file content will be used to determine the image type.

Display API's

WARNING: Displaying images from SDCard connected in **SPI mode** will be very slow. In such a case, it is recommended to copy the image files to the internal file system.

`tft.setwin(x, y, x1, y1)`

Set active display window to screen rectangle (x,y) - (x1,y1)

`tft.resetwin()`

Reset active display window to full screen size.

`tft.savewin()`

Save active display window dimensions.

`tft.restorewin()`

Restore active display window dimensions previously saved with `savewin()`.

`tft.screensize()`

Return the display size, (width, height)

`tft.winsize()`

Return the active display window size, (width, height)

`tft.hsb2rgb(hue, saturation, brightness)`

Converts the components of a color, as specified by the HSB model, to an equivalent set of values for the default RGB model.

Returns 24-bit integer value suitable to be used as color argument

Arguments

- **hue**: float: any number, the floor of this number is subtracted from it to create a fraction between 0 and 1. This fractional number is then multiplied by 360 to produce the hue angle in the HSB color model.
- **saturation**: float; 0 ~ 1.0
- **brightness**: float; 0 ~ 1.0

`tft.get_bg()`

Get the default background color

`tft.get_fg()`

Get the default foreground color

`tft.set_bg(color)`

Set the default background color

`tft.set_fg(color)`

Set the default foreground color

Low level display functions

Display API's

`tft.tft_setspeed(speed)`

Set display SPI speed

Returns tuple with the actual SPI speed set for writting and reading (write_speed, read_speed)

`tft.tft_select()`

Activate display CS

`tft.tft_deselect()`

Deactivate display CS

`tft.tft_writecmd(cmd)`

Write cmd byte to the display controller

`tft.tft_writecmddata(cmd, data)`

Write cmd byte followed by data (bytarray) to the display controller

`tft.tft_readcmd(cmd, len)`

Write cmd byte to the display controller and read len bytes of the command response bytarray
of read bytes is returned

Event API's

Event API's Button

```
from m5stack import *
# Determine if the button is pressed,buttonA_wasPressed is callback function.
btnA.wasPressed(buttonA_wasPressed)
# Define button callback function
def buttonA_wasPressed():
    # global params
    pass
# Determine if the button is released, buttonA_wasReleased is callback function.
btnA.wasReleased(buttonA_wasReleased)
# Define button callback function
def buttonA_wasReleased():
    # global params
    pass
# Decide button long press, time is pressed interval, buttonA_pressFor is callback function.
btnA.pressFor(time, buttonA_pressFor)
# Define button callback function
def buttonA_pressFor():
    # global params
    pass
# Determine if the button is double clicked,buttonA_wasDoublePress is callback function.
btnA.wasDoublePress(buttonA_wasDoublePress)
# Define button callback function
def buttonA_wasDoublePress():
    # global params
    pass
# Obtain the button was pressed, it's happen at once
btnA.wasPressed()
# Obtain the button was released, it's happen at once
btnA.wasReleased()
# Obtain the button was longpressed, it's happen at once, interval default value is 0.8
btnA.pressFor(interval = 0.8)
# Decide both buttons are pressed at the same time, btnA and btnB are button name, callback
multiBtnCb_AB.
btn.multiBtnCb(btnA, btnB, multiBtnCb_AB)
# Define button callback function
def multiBtnCb_AB():
    # global params
    pass
# Decide the button is pressed, Continuous capture of state
btnA.isPressed()
# Decide the button is released, Continuous capture of state
btnA.isReleased()
```

Event API's

Time

```
from m5stack import *
# Define a timer as name "timer1"
@timerSch.event(timer)
# Callback function which you want timer1 to run
def timer1():
    # global params
    pass

# Start the timer
Timer is timer name, interval is Running time interval, There are two mode "0X00" as Loop
execution,or "0x01" as One time.
timerSch.run(timer, interval, mode)

# If you want to change your timer setting in your programming, you can use this.
timerSch.setTimer(timer, interval, mode)

# Stop timer
timerSch.stop(timer)
```

Faces API's

Faces API's

Calculator

```
I2C Device ( detail )
Chip: ATmega328P
Addr: 0x08
import face
calc = face.get(face.CALC)
# press key will be save to str, return str
calc.readStr()
# return press key ascii int value
calc.readKey()
# clear read str to "
calc.clearStr()
# delete str last byte
calc.deleteStrLast()
# if key press turn True
calc.isNewKeyPress()
```

Encode

```
I2C Device ( detail )
Chip: ATmega328P
Addr: 0x5e
import face
encode = face.get(face.ENCODE)
# pos: 0 ~ 11
# color: 0x000000 ~ 0xffffffff (rgb888)
encode.setLed(pos, color)
# return encode value
encode.getValue()
# set encode value to zero
encode.clearValue()
# get encoder dir
# return: 0->Clockwise, 1->Counterclockwise
encode.getDir()
# get encode press status
# return: 1->not press. 0->press
encode.getPress()
```

Faces API's

Finger

```
Uart Device ( detail )
Chip: FPC1020A
finger = face.get(face.FINGER)
# Method
# will change if finger state change
finger.state

# user_id: 0 ~ 255
# access: 1, 2, 3
# note: if call this fun, finger.state -> 'Wait add finger'
# finish: finger.state -> 'Wait add finger'
# fail: finger.state -> 'Add user fail'
finger.addUser(user_id, access)

# if read know finger, will callback fingerCb with 2 formal parameter
def fingerCb(user_id, access):
    if user_id == 1 and access == 2:
        pass
finger.readFingerCb(callback=fingerCb)

# if get unknown finger, callback
def fingerCb1():
    pass
finger.getUnknownCb(callback=fingerCb1)

# user_id: 0~255
# finish: finger.state -> 'Delete user finish'
# fail: finger.state -> 'Delete user fail'
finger.removeUser(user_id)

# remove all user data
finger.removeAllUser()
```

Faces API's

Gameboy

I2C Device ([detail](#))
Chip: ATmega328P
Addr: **0x08**
import face
boy = face.get(face.GAMEBOY)
note: ↑->0, ↓->1, ←->2, →->3, A->4, B->5, START->6, SELECT->7
num: 0~7
if press return False else return True
boy.getStatus(num)
if pressed return True else return False
boy.getPressed(num)
if Released return True else return False
boy.getReleased(num)

Joystick

I2C Device ([detail](#))
Chip: ATmega328P
Addr: **0x5e**
import face
joystick = face.get(face.JOYSTICK)
X direction value, 0 ~ 1024
x = joystick.X
X invert direction value, 0 ~ 1024
x = joystick.InvertX
Y direction value, 0 ~ 1024
y = joystick.Y
Y invert direction value, 0 ~ 1024
y = joystick.InvertY
if press, return 1, else 0
press = joystick.Press
num: 0 ~ 3, color: 0x000000 ~ 0xffffffff(rgb888)
joystick.setLed(num, color)

Faces API's

Keyboard

I2C Device ([detail](#))
Chip: ATmega328P
Addr: **0x08**
import face
keyboard = face.get(face.KEYBOARD)
press key will be save to str, return str
keyboard.readStr()
return press key ascii int value
keyboard.readKey()
clear read str to "
keyboard.clearStr()
delete str last byte
keyboard.deleteStrLast()
if key press turn True
keyboard.isNewKeyPress()

RFID

I2C Device ([detail](#))
Chip: MFRC522
Addr: **0x28**
rfid = unit.get(face.RFID)
if card near return True, else return False
state = rfid.isCardOn()
return card Uid (string), like '73f66c1bf2'
if read fail, return "
uid = rfid.readUid()
block: must (block + 1) % 4 != 0
data: string, int, float, will convert to string
rfid.writeBlock(block, data)
return block data (string)
dataStr = rfid.readBlockStr(block)

Graphics/LCD API's

Graphics/LCD API's LCD

```
from m5stack import *
# Clear the screen with default background color or specific color if given
lcd.clear([color])

# Same as clear screen
lcd.fill([color])

# Print text on screen, x and y is beginning cursor , use current foreground color if not given
lcd.print(text, x, y, [color])

# Load font file
# 9 bit-mapped fonts and one vector 7-segment font are included. Unlimited number of fonts
# from file can also
# be used.
# The following font constants are defined and can be used as font arguments:
FONT_Default, FONT_DefaultSmall, FONT_DejaVu18, FONT_Dejavu24
FONT_Ubuntu, FONT_Comic, FONT_Minya, FONT_Tooney, FONT_Small
FONT_7seg

# font : required, use font name constant or font file name
# rotate : optional, set font rotation angle (0~360)
# transparent : only draw font's foreground pixels
# fixedwidth : draw proportional font with fixed character width, max character width from the
# font is used
# dist : only for 7-seg font, the distance between bars
# width : only for 7-seg font, the width of the bar
# outline : only for 7-seg font, draw the outline
# color : font color, if not given the current foreground color is used
lcd.font(font [,rotate, transparent, fixedwidth, dist, width, outline, color])

# Draw a pixel on Specified location , if color is not given, current foreground color is used.
lcd.pixel(x, y, [color])

# Draw a line on (x1, y1) and end on(x2, y2),if color is not given ,use current foreground color.
lcd.line(x1, y1, x2, y2, [color])

# Draw a rectangle rom the upper left point at (x,y) and width width and height height, color or
# fillcolor is # option, if it is not given ,use current setting
lcd.rect(x, y, width, height [,color, fillcolor])
```

Graphics/LCD API's

```
# Draw a triangle with three point at(x1, y1), (x2, y2), (x3, y3), if color is not given ,it use  
current setting  
lcd.triangle(x1, y1, x2, y2, x3, y3, [,color, fillcolor])  
  
# Draw the circle with center at (x,y) and radius r.  
lcd.circle(x, y, r, [,color, fillcolor])  
  
# Draw the circle with center at (x,y), the Center of mind is (rx, ry). If color is not given,  
current  
# foreground color is used.  
# *opt argument defines the ellipse segment to be drawn, default id 15, all ellipse segments  
# Multiple segments can drawn, combine (logical or) the values.  
  
# * 1 - upper left segment  
# * 2 - upper right segment  
# * 4 - lower left segment  
# * 8 - lower right segment  
  
lcd.ellipse(x, y, rx, ry [opt,color, fillcolor])  
  
# Draw the arc with center at (x,y) and radius r, starting at angle start and ending at angle end  
# The thicknes of the arc outline is set by the thick argument  
# If fillcolor is given, filled arc will be drawn.  
lcd.arc(x, y, r, thick, strat, end [,color, fillcolor])  
  
# Draw the polygon with center at (x,y) and radius r, with number of sides .The thicknes of the  
polygon outline # is set by the thick argument.  
# If fillcolor is given, filled polygon will be drawn.  
# If rotate is given, the polygon is rotated by the given angle (0~359)  
lcd.polygon(x, y, r, sides, thick [,color, fillcolor, rotate])  
  
# Return font size with width and height  
lcd.fontSize()  
  
# Distance the line from point (x,y) start pixels with offset angle to draw then move length  
distance pixel.  
# If color is not given, current foreground color is used.  
# The angle is given in degrees (0~359).  
lcd.lineByAngle(x, y, start, length, angle [,color])  
  
# Draw the rectangle with rounded corners from the upper left point at (x,y) and width width and  
height height  
# Corner radius is given by r argument.  
# If fillcolor is given, filled rectangle will be drawn.  
lcd.roundrect(x, y, width, height, r [color, fillcolor])
```

Graphics/LCD API's

```
# Return the width of the string text using the active font fontSize
lcd.textWidth(text)

# Clear the screen area used by string width of text at position (x,y) using the background color color.
# If color is not given, current background color is used.
lcd.textClear(x, y, text [, color])

# Display the image from the file file on position (x,y)
lcd.image(x, y, file [,scale, type])
# JPG images are supported.
# Baseline only. Progressive and Lossless JPEG format are not supported.
# Image size: Up to 65520 x 65520 pixels
# Color space: YCbCr three components only. Gray scale image is not supported.
# Sampling factor: 4:4:4, 4:2:2 or 4:2:0.
# BMP images are supported.
# Only uncompressed RGB 24-bit with no color space information BMP images can be displayed.
# Constants tft.CENTER, tft.BOTTOM, tft.RIGHT can be used for x&y
# x and y values can be negative
# scale (jpg): image scale factor: 0 to 3; if scale>0, image is scaled by factor 1/(2^scale) (1/2, 1/4 or 1/8)
# scale (bmp): image scale factor: 0 to 7; if scale>0, image is scaled by factor 1/(scale+1)
# type: optional, set the image type, constants tft.JPG or tft.BMP can be used. If not set, file extension
# and/or #file content will be used to determine the image type.image(0, 0, "res/default.jpg" )

# Set active display window to screen rectangle (x,y) - (x1,y1)
lcd.setwin(x, y, x1, y1)

# Clear the current display window with default background color or specific color if given
lcd.clearWin([color])

# Set the display rotation by angle.
lcd.setRotation(angle)

# Set screen brightness (0 - 100)
lcd.setBrightness(brightness)

# Return screensize of width and height
lcd.screensize()

# Save active display window dimensions.
lcd.savewin()
```

Graphics/LCD API's

```
# Restore active display window dimensions previously saved wint savewin()
lcd.restorewin()

# Set the default background color
lcd.set_bg(color)

# Get the default background color
lcd.get_bg()

# Set the default foreground color
lcd.set_fg(color)

# Get the default foreground color
lcd.get_fg()
```

Internal Hardware API's

Internal Hardware API's MPU6050

```
import mpu6050

# default: accel_fs = 2G, gyro_fs = 500DPS
imu = mpu6050.MPU6050()

# 3-tuple of X, Y, Z axis acceleration values in m/s^2 as floats
acc = imu.acceleration

# 3-tuple of X, Y, Z radians per second as floats.
gyro = imu.gyro

# 3-tuple of yaw, pitch, roll as floats.
ypr = imu.ypr
```

SD

```
import uos

uos.sdconfig(uos.SDMODE_SPI,clk=18,mosi=23,miso=19,cs=4)
uos.mountsd()
```

Speaker

```
from m5stack import speak

# freq: 20 ~ 20000(hz)
# duration: 0 ~ 2000(ms)
# volume: 0 ~ 100
speaker.tone(freq,duration,volume)

# freq: 20 ~ 20000(hz)
# beat: number of beat, 1 beat is 500ms
speaker.sing(freq, beat)

# volume: 0 ~ 100
speaker.setVolume(volume)
```

Internal Hardware API's

RGB Bar

```
from m5stack import rgb

# pos: 1 ~ 10
# color: 0x000000 ~ 0xffffffff rgb888
rgb.setColor(pos, color)

# 0 < posBegin < posEnd < 10
# color: 0x000000 ~ 0xffffffff rgb888
rgb.setColor(posBegin, posEnd, color)

# color: 0x000000 ~ 0xffffffff rgb888
rgb.setColorAll(color)

# brightness: 0~255
rgb.setBrightness(brightness)

Power
from m5stack import *
# Return true if is battery charge full
power.isChargeFull()

# Return true if is charging
power.isCharging()

# Return battery level
power.getBatteryLevel()
```

M5UI

Colors

Color value are given as 24 bit integer numbers, 8-bit per color.

For example: **0xFF0000** represents the RED color. Only upper 6 bits of the color component value is used.

The following color constants are defined and can be used as color arguments:

BLACK, NAVY, DARKGREEN, DARKCYAN, MAROON, PURPLE, OLIVE, LIGHTGREY, DARKGREY, BLUE, GREEN, CYAN, RED, MAGENTA, YELLOW, WHITE, ORANGE, GREENYELLOW, PINK

```
from m5stack import *
from m5ui import *
# color example
color = lcd.RED
color = lcd.YELLOW
# color: 0x000000 ~ 0xfffffff
# note: used in text, circle, rect, img bg color
setScreenColor(color)
```

Title

Init

```
# text: string, len < 50
# color1: 0x000000 ~ 0xfffffff
# color2: 0x000000 ~ 0xfffffff
title = M5Title(title=text, fgcolor=color1, bgcolor=color2)
```

Method

```
# hide and show title
title.hide()
title.show()
# Text is string, len < 50
title.setTitle(text)
# color: 0x000000 ~ 0xfffffff
title.setFgColor(color)
# color: 0x000000 ~ 0xfffffff
title.setBgColor(color)
```

M5UI API's

Circle

Init

```
from m5ui import M5Circle  
# if borderColor not give: borderColor = fillColor  
circle = M5Circle(x, y, r, fillColor, [borderColor])
```

Method

```
circle.setSize(r)  
circle.setBgColor(color)  
circle.setBorderColor(color)  
# change circle x, y pos  
circle.setPosition(x=0, y=0)  
# change circle x pos, y unchanging  
circle.setPosition(x=0)  
# change circle y pos, x unchanging  
circle.setPosition(y=0)  
circle.hide()  
circle.show()
```

Img

Init

```
# init, img support jpg and bmp  
# path: string, like "res/default.jpg"  
# visibility: True or False  
img = M5Img(x, y, path, visibility)
```

Method

```
img.hide()  
img.show()  
# change img x, y pos  
img.setPosition(x=0, y=0)  
# change img x pos, y unchanging  
img.setPosition(y=0)  
# change img y pos, x unchanging  
img.setPosition(x=0)  
# path: img path. like 'res/default.jpg'  
img.changeImg(path)
```

M5UI API's

Rect

Init

init

```
rect = M5Rect(x, y, width, height, fillColor, borderColor)
```

Method

change rect width and height

```
rect.setSize(width=10, height=10)
```

only change rect height

```
rect.setSize(height=10)
```

only change rect width

```
rect.setSize(width=10)
```

change rect x pos and y pos

```
rect.setPosition(x=10, y=10)
```

only change rect x pos

```
rect.setPosition(x=10)
```

only change rect y pos

```
rect.setPosition(y=10)
```

```
rect.setBgColor()
```

```
rect.setBorderColor()
```

```
rect.hide()
```

```
rect.show()
```

M5UI API's

TextBox

Init

```
# font: lcd.FONT_DejaVu18, lcd.FONT_DejaVu24, lcd.FONT_DejaVu40,  
lcd.FONT_DejaVu56  
#    lcd.FONT_DejaVu72, lcd.FONT_Ubuntu, lcd.FONT_Comic    lcd.FONT_Minya  
#    lcd.FONT_Tooney  
# rotate: 0 ~ 360  
label = M5TextBox(x, y, text, font, color, [rotate=0])
```

Method

```
label.setTextColor(color)  
# change rect x pos and y pos  
label.setPosition(x=10, y=10)  
# only change rect x pos  
label.setPosition(x=10)  
# only change rect y pos  
label.setPosition(y=10)  
label.setFont(font)  
label.setRotate(rotate)  
label.setText(text)  
label.hide()  
label.show()
```

Remote API's

Remote API's Remote

```
import machine
remoteInit()

# Generate QR code ,(x, y)is left top cursor, the QR image is size width and size height
lcd.qrcode('http://flow-remote.m5stack.com/?remote=undefined', x, y, size)

# Define a switch button.
def _remote_SwitchName():
    pass

# Define a button.
def _remote_ButtonName():
    pass

# Define a sliderbar.
def _remote_SliderName():
    pass

# Define a label and return a value.
def _remote_LabelName():
    pass
```

HAT API's

Hat API's

The following API's defined in the Hat section are located in the HAT Library. To use the API's we use the following Micropython code. (please note that a bug in UIFlow UI sometimes calls this twice as you have seen in some of the demo's in previous sections.

Import Hat

ENV HAT

I2C Device

Chip: DHT12, BMP280, BMM150

Addr: *0x5c, 0x76*

```
hat_env0 = hat.get(hat.ENV)
# temperature: -20 ~ 60, Celsius
tmp = hat_env0.temperature
# humidity: 20 ~ 95, percentage
hum = hat_env0.humidity
# pressure: 30000 ~ 110000Pa, pa
pre = hat_env0.pressure
```

PIR HAT

Digital Device

```
hat_pir0 = hat.get(hat.PIR)
# return 0 or 1
state = hat_pir0.state
```

Speaker

```
hat_spk0 = hat.get(hat.SPEAKER)
# frequency: 20 ~ 20000(hz)
# duration: 0 ~ 2000(ms)
# beat: number of beat, 1 beat is 500ms
# volume: 0 ~ 100
hat_spk0.tone(freq, duration)
hat_spk0.setVolume(volume)
hat_spk0.sing(frequency, beat)
```

NCIR

I2C Device

Chip: MLX90614

Addr: *0x5a*

```
hat_ncir0 = hat.get(hat.NCIR)
# tmp: -70°C ~ 382.2°C
hat_ncir0.temperature
```

HAT API's

DAC

```
I2C Device  
Chip: MCP4725  
Addr: 0x60  
hat_dac0 = hat.get(hat.DAC)  
# vol: 0~3.3  
# save: True or False, whether to save to eeprom  
hat_dac0.setVoltage(1,save=True)  
# data: 0~4096, maybe 2640 output is 3.3V, max is 3.7V  
# save: True or False, whether to save to eeprom  
hat_dac0.writeData(1,save=True)
```

ADC

```
I2C Device  
Chip: ADS1100  
Addr: 0x48  
hat_adc0 = hat.get(hat.ADC)  
# get voltage, value range 0~12, measurement: V  
hat_adc0.voltage
```

Servo

```
hat_servo0 = hat.get(hat.SERVO)  
# degrees: 0 ~ 180, degree  
hat_servo0.write_angle(degrees)  
# us: 500 ~ 2500, 50HZ, High level duration  
hat_servo0.write_us(600)
```

Servo8

```
I2C Device  
Chip: STM32F030F4  
Addr: 0x38  
hat_servos0 = hat.get(hat.SERVOS)  
#channel 0, 1, 2, 3, 4, 5, 6, 7  
# degrees: 0 ~ 180, degree  
# us: 500 ~ 2500, 50HZ, High level duration  
hat_servos0.SetAngle(channel, degree)  
hat_servos0.SetPulse(channel, us)  
# color: 0x000000 ~ 0xffffffff rgb888  
hat_servos0.SetRGB(color)
```

PuppyC

```
hat_puppy0 = hat.get(hat.PUPPY)  
#channel 0, 1, 2, 3.  
# degrees: 0 ~ 180, degree  
hat_puppy0.SetAngle(channel, degrees)  
hat_puppy0.SetAllAngle(degrees, degrees, degrees, degrees)
```

HAT API's

Joystick

I2C Device

Chip: ATmega328P

Addr: **0x52**

```
hat_Joystick0 = hat.get(hat.JOYSTICK)
# X direction value, 0 ~ 255
x = hat_Joystick0.X
# Y direction value, 0 ~ 255
y = hat_Joystick0.Y
# if press, return 1, else 0
press = hat_Joystick0.Press
# X invert direction value, 0 ~ 255
x = hat_Joystick0.InvertX
# Y invert direction value, 0 ~ 255
y = hat_Joystick0.InvertY
```

BugC

```
hat_bugc0 = hat.get(hat.BUGC)
```

```
hat_bugc0.SetPulse(channel, us)
# us: 500 ~ 2500, 50HZ, High level duration
hat_bugc0.SetAllPulse(us, us, us, us)
# color: 0x000000 ~ 0xffffffff rgb888
hat_bugc0.SetRGB(0, color
hat_bugc0.SetRGB(0, color)
hat_bugc0.SetAllRGB(color)
```

HAT API's

Finger

Uart Device

```
Chip: FPC1020A hat_Finger0 = hat.get(hat.FINGER)
# Method
# will change if finger state change
state = hat_Finger0.state
hat_Finger0_unknownCb()
# user_id: 0 ~ 255
# access: 1, 2, 3
hat_Finger0_cb(user_id, access)
# remove all user data
hat_Finger0.removeAllUser()
#removes user stored in location.
hat_Finger0.removeUser(user_id)
# note: if call this function, unit_finger.state -> 'Wait add finger'
# finish: unit_finger.state -> 'Wait add finger'
# fail: unit_finger.state -> 'Add user fai
hat_Finger0.addUser(user_id,access)
access = access
user_id = user_id
```

BeetleC

Chip: STM32F030

```
hat_BeetleC0 = hat.get(hat.BEETLEC)
#wheel: 0 = left wheel, 1 = right wheel
#pulse = 0 to 255 (?)
#led = 0 to 6
# color: 0x000000 ~ 0xffffffff rgb888
hat_BeetleC0.SetPulse(wheel, pulse)
hat_BeetleC0.SetRGB(led, color)
hat_BeetleC0.SetAllRGB(color)
```

Yun

Chip: STM32F030, SHT20, BMP280

```
hat_yun0 = hat.get(hat.YUN)
hat_yun0.temperature
hat_yun0.humidity
hat_yun0.pressure
at_yun0.getLight(light)
#led = 0 to 13
# color: 0x000000 ~ 0xffffffff rgb888
hat_yun0.SetRGB(led, color)
hat_yun0.SetRGBAll(color)
```

HAT API's

JoyC

```
hat_joyc0 = hat.get(hat.JOYC)
#led = 0 to 6
# color: 0x000000 ~ 0xffffffff rgb888
hat_joyc0.SetLedColor(color)
#joyx = 0 left, 1 right.
hat_joyc0.GetX(joyx)
#joyy = 0 left, 1 right.
hat_joyc0.GetY(joyy)
#joyangle = 0 left, 1 right.
hat_joyc0.GetAngle(joyangle)
#joyd = 0 left, 1 right.
hat_joyc0.GetDistance(joyd)
# if press, return 1, else 0
#joyp = 0 left, 1 right.
hat_joyc0.GetPress(Joyp)
```

RoverC

I2C Device
Chip: STM32F030
Addr = 0x38
hat_roverc0 = hat.get(hat.ROVERC)
#motor = 0 - 3
us: 500 ~ 2500, 50HZ, High level duration
hat_roverc0.SetPulse(motor, us)
hat_roverc0.SetAllPulse(us, us, us, us)
#speed = 0 to 255 in each direction.
hat_roverc0.SetSpeed(speed, speed, speed)

TOF

I2C Device
Chip: VL53L0X
Addr: **0x29**
hat_tof0 = hat.get(hat.TOF)
distance: 30 ~ 800(mm), other is invalid
hat_tof0.GetDistance()

HAT API's

Thermal

I2C Device

Chip: MLX90640

Addr: 0x33

```
hat_thermal0 = hat.get(hat.MLX90640)
hat_thermal0.getTmp(Xtemp, YTemp)
hat_thermal0.getCenterTmp()
hat_thermal0.getMaxTmp()
hat_thermal0.getMinTmp()
hat_thermal0.setColorMaxTmp(35)
hat_thermal0.setColorMinTmp(24)
hat_thermal0.update(x=0, y=0, show=True, showCenter=True)
```

CardKBC

I2C Device

Addr: 0x5F

```
hat_cardkb0 = hat.get(hat.CARDKB)
hat_cardkb0.keyData
hat_cardkb0.keyString
hat_cardkb0.isNewKeyPress()
```

Unit API's

units

Note: If you want use I2C device at PORTC, please remove the **PORTC resistor in M5GO**

M5GO PORT IO

PORTA = (21,22) # I2C

PORTB = (26,36) # Analog read

PORTC = (17,16) # UART2

ENV

I2C Device

Chip: dht12, bmp280

Addr: **0x5c, 0x76**

init --- Effective IO: PORTA | PORTC

unit_env = unit.get(unit.ENV, unit.PORTA)

temperature: -20 ~ 60, Celsius

tmp = unit_env.temperature

humidity: 20 ~ 95, percentage

hum = unit_env.humidity

pressure: 30000 ~ 110000Pa, pa

pre = unit_env.pressure

PIR

Digital Device

init --- Effective IO: PORTA | PORTB | PORTC

unit_pir = unit.get(unit.PIR, unit.PORTB)

return 0 or 1

state = pir.state

Neopixel

Digital Device

Chip: **SK6812**

init --- Effective IO: PORTA | PORTB | PORTC, number --> 0 ~ 1023

unit_Neopixel = unit.get(unit.NEOPIXEL, unit.PORTB, number)

pos: 1 ~ number

color: 0x000000 ~ 0xffffffff rgb888

unit_Neopixel.setColor(pos, color)

0 < posBegin < posEnd < number

color: 0x000000 ~ 0xffffffff rgb888

unit_Neopixel.setColor(posBegin, posEnd, color)

color: 0x000000 ~ 0xffffffff rgb888

unit_Neopixel.setColorAll(color)

brightness: 0~255

unit_Neopixel.setBrightness(brightness)

Joystick

I2C Device

Chip: ATmega328P

Addr: **0x52**

```
# init --- Effective IO: PORTA | PORTC
unit_joystick = unit.get(unit.JOYSTICK, unit.PORTA)
# X direction value, 0 ~ 255
x = unit_joystick.X
# X invert direction value, 0 ~ 255
x = unit_joystick.InvertX
# Y direction value, 0 ~ 255
y = unit_joystick.Y
# Y invert direction value, 0 ~ 255
y = unit_joystick.InvertY
# if press, return 1, else 0
press = unit_joystick.Press
```

Makey

I2C Device)

Chip: ATmega328P

Addr: **0x51**

```
# init --- Effective IO: PORTA | PORTC
unit_makey = unit.get(unit.MAKEY, unit.PORTA)
# data:-1, 0 ~ 15
data = unit_makey.value
```

Servo

Analog Device

```
# init --- Effective IO: PORTA | PORTB | PORTC
unit_servo = unit.get(unit.SERVO, unit.PORTA)
# us: 500 ~ 2500, 50HZ, High level duration
unit_servo.write_us(us)
# degrees: 0 ~ 180, degree
unit_servo.write_angle(degrees)
```

Unit API's

Weight

Digital Device

Chip: **HX711**

```
# init --- Effective IO: PORTA | PORTB | PORTC
unit_weight = unit.get(unit.WEIGHT, unit.PORTA)
# 0 ~ 16777216 (24bit)
rawData = unit_weight.rawData
# get weight
weight = unit_weight.weight
# set weight to zero
unit_weight.zero()
```

Tracker

I2C Device

Chip: ATmega328P

Addr: **0x5a**

```
# init --- Effective IO: PORTA | PORTC
unit_track = unit.get(unit.TRACK, unit.PORTA)
# pos: 1, 2, 3, 4
# value: 0 ~ 1024
value = unit_track.getAnalogValue(pos)
# pos: 1, 2, 3, 4
# value: 0 ~ 1024
unit_track.setAnalogValue(pos, value)
# pos: 1, 2, 3, 4
# value: 0, 1
value = unit_track.getDigitalValue(pos)
```

Button

Digital Device

```
# init --- Effective IO: PORTA | PORTB | PORTC  
unit_btn = unit.get(unit.BUTTON, unit.PORTB)
```

Method

```
# if set the callback param, it will interrupt callback function  
# or if not set param it will return result(True or False) at once  
unit_btn.wasPressed(callback=None)  
unit_btn.wasReleased(callback=None)  
unit_btn.wasDoublePress(callback=None)  
# holdTime: sec  
unit_btn.pressFor(holdTime=1.0, callback=None):
```

#example

```
def on_wasPressed():  
    lcd.print('Button was Pressed\n')
```

```
def on_wasReleased():  
    lcd.print('Button was Released\n')
```

```
def on_pressFor():  
    lcd.print('Button press for 1.2s press hold\n')
```

```
def on_doublePress():  
    lcd.print('Button was double press\n')
```

```
unit_btn.wasPressed(on_wasPressed)  
unit_btn.wasReleased(on_wasReleased)  
unit_btn.wasDoublePress(on_doublePress)  
unit_btn.pressFor(1.2, on_pressFor)
```

Dual Button

Digital Device

```
# init --- Effective IO: PORTA | PORTB | PORTC  
unit_dualButton = unit.get(unit.DUAL_BUTTON, unit.PORTB)  
# btnRed, btnRed method like button unit  
btnRed = unit_dualButton.btnRed  
btnRed = unit_dualButton.btnBlue
```

Unit API's

RGB

Digital Device

Chip: **SK6812**

```
# init --- Effective IO: PORTA | PORTB | PORTC
unit_Rgb = unit.get(unit.RGB, unit.PORTB)
# pos: 1 ~ number
# color: 0x000000 ~ 0xffffffff rgb888
unit_Rgb.setColor(pos, color)
# 0 < posBegin < posEnd < number
# color: 0x000000 ~ 0xffffffff rgb888
unit_Rgb.setColor(posBegin, posEnd, color)
# color: 0x000000 ~ 0xffffffff rgb888
unit_Rgb.setColorAll(color)
# brightness: 0~255
unit_Rgb.setBrightness(brightness)
```

Relay

Digital Device

```
# init --- Effective IO: PORTA | PORTB | PORTC
unit_relay = unit.get(unit.RELAY, unit.PORTB)
# select com connect on
unit_relay.on()
# select com connect off
unit_relay.off()
```

ADC

I2C Device

Chip: ADS1100

Addr: **0x48**

import unit

```
# init --- Effective IO: PORTA | PORTC
unit_adc = unit.get(unit.ADC, unit.PORTA)
```

```
# get voltage, value range 0~12, measurement: V
vol = unit_adc.voltage
```

DAC

```
I2C Device
Chip: ADS1100
Addr: 0x48
# init --- Effective IO: PORTA
unit_dac = unit.get(unit.DAC, unit.PORTA)

# data: 0~4096, maybe 2640 output is 3.3V, max is 3.7V
# save: True or False, whether to save to eeprom
unit_dac.writeData(data, save=False)
# vol: 0~3.3
# save: True or False, whether to save to eeprom
unit_dac.setVoltage(vol, save=False)
```

Ncir

```
I2C Device
Chip: MLX90614
Addr: 0x5a
# init --- Effective IO: PORTA | PORTC
unit_ncir = unit.get(unit.NCIR, unit.PORTA)
# tmp: -70°C ~ 382.2°C
tmp = unit_ncir.temperature
```

IR

```
Digital Device
# init --- Effective IO: PORTA | PORTB | PORTC
unit_ir = unit.get(unit.IR, unit.PORTB)
# ir tx 25hz signal with 38khz signal carrier
unit_ir.txOn()
# off tx off
unit_ir.txOff()
# if receive ir signal return 1 else return 0
value = unit.rxStatus()
```

Unit API's

Extend IO

I2C Device
Chip: PCA9554PW
Addr: *0x27*
init --- Effective IO: PORTA | PORTC
`unit_extIO = unit.get(unit.EXT_IO, unit.PORTA)`
Method
mode: unit_extIO.ALL_OUTPUT or unit_extIO.ALL_INPUT
`unit_extIO.setPortMode(mode)`
pin: 0~7, mode: 1 or 0, 1:output, 0:INPUT
`unit_extIO.setPinMode(pin, mode)`
state --> 0x00 ~ 0xff,
0 pin in 0 bit 7 pin in 7 bit
singState = (state >> pin) & 0x01
`state = unit_extIO.digitReadPort()`
pin: 0~7
state --> 0: LOW, 1:HIGH
`state = unit_extIO.digitRead(pin)`
state: 0x00 ~ 0xff
`unit_extIO.digitWritePort(state)`
pin: 0 ~ 7, value: 0:LOW, 1:HIGH
`unit_extIO.digitWrite(pin, value)`

Angle

Analog Device
init --- Effective IO: PORTB
`unit_angle = unit.get(unit.ANGLE, unit.PORTB)`

get raw value, range 0 ~ 4095
`adValue = unit_angle.readraw()`

get filter value, range 0 ~ 1023
`filterValue = unit_angle.read()`

Light

Analog/Digital Device
init --- Effective IO: PORTB
`unit_light = unit.get(unit.LIGHT, unit.PORTB)`
adc value, range 0 ~ 1024
`analogValue = unit_light.analogValue`
digital value, only 0 or 1, base rotate the knob
`digitalValue = unit_light.digitalValue`

Earth

```
Analog/Digital Device  
# init --- Effective IO: PORTB  
unit_earth = unit.get(unit.LIGHT, unit.PORTB)  
# adc value, range 0 ~ 1024  
analogValue = unit_earth.analogValue  
# digital value, only 0 or 1, base rotate the knob  
digitalValue = unit_earth.digitalValue
```

Tof

```
I2C Device  
Chip: VL53L0X  
Addr: 0x29  
# init --- Effective IO: PORTA | PORTC
```

```
unit_tof = unit.get(unit.TOF, unit.PORTA)  
# distance: 30 ~ 800(mm), other is invalid  
distance = unit_tof.distance
```

Color

```
I2C Device  
Chip: TCS3472  
Addr: 0x29  
# init --- Effective IO: PORTB  
unit_color = unit.get(unit.COLOR, unit.PORTA)
```

```
# return: tuple, like (3521, 1515, 1147, 865), (clear, red, green, blue)  
raw = unit_color.rawData
```

```
# return red value, range 0~255  
red = unit_color.red  
# return green value, range 0~255  
unit_color.green  
# return blue value, range 0~255  
unit_color.blue
```

Unit API's

RFID

I2C Device

Chip: MFRC522

Addr: *0x28*

```
# init --- Effective IO: PORTA | PORTC
unit_rfid = unit.get(unit.RFID, unit.PORTA)
# if card near return True, else return False
state = unit_rfid.isCardOn()
# return card Uid (string), like '73f66c1bf2'
# if read fail, return ""
uid = unit_rfid.readUid()
# block: must (block + 1) % 4 != 0
# data: string, int, float, will convert to string
unit_rfid.writeBlock(block, data)
# return block data (string)
dataStr = unit_rfid.readBlockStr(block)
```

Finger

Uart Device

Chip: FPC1020A

```
# init --- Effective IO: PORTA | PORTC
unit_finger = unit.get(unit.FINGER, unit.PORTC)
# Method
# will change if finger state change
unit_finger.state
# user_id: 0 ~ 255
# access: 1, 2, 3
# note: if call this function, unit_finger.state -> 'Wait add finger'
# finish: unit_finger.state -> 'Wait add finger'
# fail: unit_finger.state -> 'Add user fail'
unit_finger.addUser(user_id, access)
# if read know finger, will callback fingerCb with 2 formal parameter
def fingerCb(user_id, access):
    if user_id == 1 and access == 2:
        pass
unit_finger.readFingerCb(callback=fingerCb)
# user_id: 0~255
# finish: unit_finger.state -> 'Delete user finish'
# fail: unit_finger.state -> 'Delete user fail'
unit_finger.removeUser(user_id)
# remove all user data
unit_finger.removeAllUser()
```

CardKB

I2C Device

Chip: ATmega328P

Addr: **0x5F**

init --- Effective IO: PORTB

```
unit_cardKB = unit.get(unit.CARDKB, unit.PORTA)
```

isNewKeyPress() --> if press key return True else False

```
unit_cardKB.isNewKeyPress():
```

return last press key value at once, type: int

```
key = unit_cardKB.keyData
```

return press string, len < 50, press ESC clear, press [X] delete last one

```
keyString = unit_cardKB.keyString
```

Heart

I2C Device,

Chip: MAX 30100

Addr: 0x57

init -- Effective IO: PORTA

```
heart0 = unit.get(unit.HEART, unit.PORTA)
```

```
heart0.setMode(0x02)
```

#0x02 = Heart rate only,

#0x03 = Heart rate with SPO2

```
heart0.setLedCurrent(0x00, 0x00)
```

Data-sheet Catalogue

Appendix 1 **Data-sheet Catalogue.**

This page is a catalogue of the known data sheets pertaining to modules used throughout the M5Stack Product line. Please note: while every attempt is made to keep this list up to date, due to the ever changing electronics environment, the list may become inaccurate as data sheets are posted online or are removed.

ADS1100 self calibrating analogue to digital converter
<http://www.ti.com/lit/ds/symlink/ads1100.pdf>

AK8983C
<https://www.akm.com/akm/en/file/datasheet/AK8963C.pdf>

ATMEGA328P
http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

AS312 PIR Sensor
<https://forum.mysensors.org/assets/uploads/files/1494013712469-pir-as312.pdf>

AT6558 GPS Receiver
<http://www.icochina.com/d/file/xiazai/2016-12-05/b1be6f481cdf9d773b963ab30a2d11d8.pdf>

AXP192 Single Cell Li Battery and Power Management IC
<http://www.x-powers.com/en.php/Info/down/id/50>

BMM150 Geomagnetic Sensor
https://m5stack.oss-cn-shenzhen.aliyuncs.com/resource/docs/datasheet/core/BMM150_datasheet_en.pdf

BMP280 Pressure Sensor
<https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf>

Cadence/Tensilica LX6 Processor
https://mirrobo.ru/wp-content/uploads/2016/11/Cadence_Tensillica_Xtensa_LX6_ds.pdf

CP2102 USB to UART communications chip
<https://www.silabs.com/documents/public/data-sheets/CP2102-9.pdf>

DHT12 Digital Temperature and Humidity Sensor.
<http://www.robototehnika.ru/file/DHT12.pdf>

ESP32

https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

ESP32 Wrover Module

https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf

Data-sheet Catalogue

FPC1020A fingerprint Module

http://www.shenzhen2u.com/doc/Module/Fingerprint/710-FPC1020_PB3_Product-Specification.pdf

HK4100 F 5V DC relay

https://img.ozdisan.com/ETicaret_Dosya/445413_4369639.pdf

Holtek HT75XX LDO Voltage Regulator.

<http://www.e-ele.net/DataSheet/HT75XX-1.pdf>

L293D H-Bridge Driver

<http://www.ti.com/lit/ds/symlink/l293.pdf>

LM393DR2G

<https://www.onsemi.com/pub/Collateral/LM393-D.PDF>

MAX2359 Amplifier

<https://datasheets.maximintegrated.com/en/ds/MAX2359.pdf>

MAX30100 Pulse-Oximeter.

<https://datasheets.maximintegrated.com/en/ds/MAX30100.pdf>

MCP4725 12-Bit Digital-to-Analog Converter

<http://ww1.microchip.com/downloads/en/devicedoc/22039d.pdf>

MLX90614 N.C.I.R Sensor

https://www.sparkfun.com/datasheets/Sensors/Temperature/MLX90614_rev001.pdf

MPU6050

https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf

MPU9250

<https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf>

PCA9554A

https://www.nxp.com/docs/en/data-sheet/PCA9554_9554A.pdf

SK6812 LEDs

<https://cdn-shop.adafruit.com/product-files/1138/SK6812+LED+datasheet+.pdf>

SPX3819 LDO Voltage Regulator.

http://www.mouser.com/ds/2/146/SPX3819_DS_R200_082312-17072.pdf

TCA5948 A 8 way I2C Switch.

<http://www.ti.com/lit/ds/symlink/tca9548a.pdf>

Data-sheet Catalogue

TCS3472 Colour Light to Digital Converter.

https://ams.com/documents/20143/36005/TCS3472_DS000390_2-00.pdf

VL53L0X Time Of Flight Sensor.

<https://www.st.com/resource/en/datasheet/vl53l0x.pdf>

WS2812b (Neopixel) LED

<https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf>

I2C Address

Appendix 2 I2C Address.

The following tables contain a list of known I2C addresses used by M5Stack devices.

CORE	ADDRESS	CHIP
M5CORE	0x75	IP5306
M5CORE	0x68	MPU6886
M5CORE	0x6C	SH200Q
M5CORE	0x10	BMM150
M5STICK	0x75	IP5306
M5STICK	0x68	MPU9250
M5STICK-C	0x34	AXP192
M5STICK-C	0x6C	SH200Q
M5STICK-C	0x51	BMM8563

Table 1 - M5Stack and M5Stick internal additional hardware I2C addresses. Please note that the additional devices were optional on some versions.

Device	Address	Chip
ACCEL	0x53	ADXL345
ADC	0x48	ADS1100
Card KB	0x5F	MEGA328P
Colour Sensor	0x29	TCS3472
DAC	0x60	MCP4725
ENV	0x5C	DH12
ENV	0x76	BMP280
EXT I/O	0x27	PCA9554PW
Heart	0x57	MAX30110
Joystick	0x52	MEGA328P
Makey Makey	0x51	MEGA328P
N.C.I.R.	0x5A	MLX90614
PaHub	0x70	TCA9548A

Device	Address	Chip
PbHub	0x40	MEGA328P
R.F.I.D	0x28	MFRC522
Thermal	0x33	MLX90640
Trace (See 3.3.38)	0x5A	MEGA328P
TOF	0x29	VL53L0X
HAT-ENV	0x5C	DH12
HAT-ENV	0c77	BMP280
HAT-ENV	0x10	BMM150
HAT-THERMAL	0x33	MLX90640
HAT-NCIR	0x5A	MLX90614
HAT ADC	0x48	ADS1100
HAT-TOF	0x29	VL53L0X & VCSEL
HAT-DAC	0x60	MCP4725
HAT-JOYSTICK	0x38	STM32F030

Table 2 - Unit I2C addresses.

Camera	Address	Chips
ESP32	0x68, 0x76	MPU6050, BME280
M5CAMERA	0x68, 0x76	MPU6050, BME280
M5CAMERA - F	0x68, 0x76	MPU6050, BME280
M5CAMERA - X	0x68, 0x76	MPU6050, BME280

Table 3 - Camera hardware devices. Please note that the additional devices were optional on some versions.

MODULE	ADDRESS	IC
PLUS	0x62	MEGA328P
GO - PLUS	0x61	MEGA328P
STEP - MOTOR	0x70	MEGA328P
SERVO	0x53	MEGA328P
LEGO +	0x56	MEGA328P
FACES - ENCODER	0x5E	MEGA328P
FACES - JOYSTICK	0x5E	MEGA328P

MODULE	ADDRESS	IC
FACES - KEYBOARD	0x78	MEGA328P
FACES - CALCULATOR	0x78	MEGA328P
FACES - GAMEBOY	0x78	MEGA328P
FACES - RFID	0x28	MFRC522

Table 4 - Module and Faces I2C address.

CardKB Symbols

Appendix 3

Table of symbols available on the CardKB and the CardKB Hat.

The following table lists the symbols available through the various Key combinations along with the raw hexadecimal values.

Key	Value	Sym+Key	Value	Shift+Key	Fn+key
Esc	0x1B	Esc	0x1B	Esc	Esc
1	0x31	!	0x21	1	1
2	0x32	@	0x40	2	2
3	0x33	#	0x23	3	3
4	0x34	\$	0x24	4	4
5	0x35	%	0x25	5	5
6	0x36	^	0x5E	6	6
7	0x37	&	0x26	7	7
8	0x38	*	0x2A	8	8
9	0x39	(0x28	9	9
0	0x30)	0x29	0	0
Del	0x08	Del	0x08	Del	Del
Tab	0x09	Tab	0x09	Tab	Tab
q	0x71	{	0x7B	Q	Q
w	0x77	}	0x7D	W	W
e	0x65	[0x5B	E	E
r	0x72]	0x5D	R	R
t	0x74	/	0x2F	T	T
y	0x79	\	0x5C	Y	Y
u	0x75		0x7C	U	U
i	0x69	-	0x7E	I	I

Key	Value	Sym+Key	Value	Shift+Key	Fn+key
o	0x6F	'	0x27	O	O
p	0x70	"	0x22	P	P
Fn	NULL	NULL	NULL	NULL	NULL
Shift	NULL	NULL	NULL	NULL	NULL
a	0x61	;	0x3B	A	A
s	0x73	:	0x3A	S	S
d	0x64	`	0x60	D	D
f	0x66	+	0x2B	F	F
g	0x67	-	0x2D	G	G
h	0x68	_	0x5F	H	H
j	0x6A	=	0x3D	J	J
k	0x6B	?	0x3F	K	K
l	0x6C	NULL	NULL	L	L
Enter	0x0D	Enter	0x0D	Enter	Enter
Sym	NULL	NULL	NULL	NULL	NULL
z	0x7A	NULL	NULL	Z	Z
x	0x7B	NULL	NULL	X	X
c	0x63	NULL	NULL	C	C
v	0x76	NULL	NULL	V	V
b	0x62	NULL	NULL	B	B
n	0x6E	NULL	NULL	N	N
m	0x6D	NULL	NULL	M	M
,	0x2C	<	0x3C	,	,
.	0x2E	>	0x3E	.	.
Space	0x20	Space	0x20	Space	0x20
Up	0xB5	Up	0xB5	Up	Up
Down	0xB6	Down	0xB6	Down	Down
Left	0xB4	Left	0xB4	Left	Left
Right	0xB7	Right	0xB7	Right	Right

Index

Index

A
acc = imu.acceleration
ACCEL
Acceleration
Accelerometer
ADC
ADC Hat
ADC Unit
adc.attenuation()
adc=machine.ADC(pin)
adc.read()
adc.width(value)
Add
Addition
add_peer
Address
addrList = i2c1.scan()
ADS1100
Advanced
AK8933C
Amp
Analogue
Analogue Read
Analogue Write
Angle
Angle Unit
API
APIKey
AS321
AT6885
ATMEGA 328P
AXP192

B
BeetleC
BLACK
BLUE
BMP280
boy = face.get(face.GAMEBOY)
boy.getPressed(num)
boy.getReleased(num)
boy.getStatus(num)

btn.multiBtnCb(btnA, btnB,
multiBtnCb_AB)
btnA.isPressed()
btnA.isReleased()
btnA.pressFor(interval = 0.8)
btnA.pressFor(time, buttonA_pressFor)
btnA.wasDoublePress(buttonA_wasDoublePress)
btnA.wasPressed()
btnA.wasPressed(buttonA_wasPressed)
btnA.wasReleased()
btnA.wasReleased(buttonA_wasReleased)
BugC
Button
Button Loop

C
Calculator
calc.clearStr()
calc.deleteStrLast()
calc.isNewKeyPress()
calc.readKey()
calc.readStr()
CardKB
CardKB C
Circle
circle = M5Circle(x, y, r, fillColor,
[borderColor])
circle.hide()
circle.show()
circle.setBgColor(color)
circle.setBorderColor(color)
circle.setPosition(x=0, y=0)
circle.setPosition(x=0)
circle.setPosition(y=0)
circle.setSize(r)
COLOR Unit
color = lcd.RED
CYAN

D
DAC
DAC Hat

DAC Unit	Finger Face
data = adc.read()	FONT
data = i2c1.read_u8(reg):	FONT_7seg
data = i2c1.read_u16(reg, byteorder='big')	FONT_Comic
data = i2c1.read(num)	FONT_Default
dataStr = rfid.readBlockStr(block)	FONT_DefaultSmall
dac = machine.DAC()	FONT_DejaVu18
dac.beep(freq, duration [, scale=0])	FONT_DejaVu24
dac.freq(freq)	FONT_Minya
dac.stopwave()	FONT_Small
dac.waveform(freq, type [, duration=0][, scale=0] [, offset=0] [, invert=2])	FONT_Tooney
dac.write(value)	FONT_Ubuntu
DARKCYAN	FPC1020A
DARKGREEN	G
DARKGREY	Gameboy
DHT12	getP2PData()
digitalRead(pin)	GPS Unit
digitalWrite(pin)	GREEN
Dual Button	GREENYELLOW
E	gyro = imu.gyro
Earth Unit	H
ENV	Heart Unit
ENV Unit	hum = unit_env.humidity
ESP NOW	I
espnow.add_peer(peer, [channel, ifidx, encrypt, lmk, id])	I2C
espnow.broadcast([cmd_id, data])	i2c1 = i2c_bus.easyI2C(port, addr)
espnow.deinit()	i2c1.write_u8(reg, data)
espnow.get_mac_address()	i2c.write_u16(reg, data, byteorder='big')
espnow.init()	Image
espnow.recv_cb(recv_cb)	img.changeImg(path)
espnow.send_cb(send_cb)	img.hide()
espnow.send([id, mac_addr, cmd_id, data])	img = M5Img(x, y, path, visibility)
espnow.set_pmk(pmkk)	img.setPosition(x=0, y=0)
EXT.IO	img.setPosition(y=0)
F	img.setPosition(x=0)
finger.addUser(user_id, access)	img.show()
finger = face.get(face.FINGER)	import espnow
finger.getUnknownCb(callback=fingerCb1)	import face
finger.readFingerCb(callback=fingerCb)	import machine
finger.removeAllUser()	import i2c_bus
finger.removeUser(user_ID)	import mpu6050
finger.state	import M5mqtt
Finger Hat	import speak
Finger Unit	import uos
	import wifiCfg

imu = mpu6050.MPU6050()

IR Unit

J

joystick = face.get(face.JOYSTICK)

Joystick Hat

Joystick Unit

JoyC

K

keyboard = face.get(face.KEYBOARD)

keyboard.clearStr()

keyboard.deleteStrLast()

keyboard.isNewKeyPress()

keyboard.readKey()

keyboard.readStr()

L

Label

label = M5TextBox(x, y, text, font, color,
[rotate=0])

label.hide()

label.setColor(color)

label.setFont(font)

label.setPosition(x=10, y=10)

label.setPosition(x=10)

label.setPosition(y=10)

label.setRotate(rotate)

label.setText(text)

label.show()

LCD

lcd.arc(x, y, r, thick, strat, end [,color,
fillcolor])

lcd.circle(x, y, r, [,color, fillcolor])

lcd.ellipse(x, y, rx, ry [opt,color, fillcolor])

lcd.clear([color])

lcd.clearWin([color])

lcd.fill([color])

lcd.fontSize()

lcd.font(font [,rotate, transparent, fixedwidth,
dist, width, outline, color])

lcd.get_bg()lcd.set_fg(color)

lcd.get_fg()

lcd.image(x, y, file [,scale, type])

lcd.line(x1, y1, x2, y2, [,color])

lcd.lineByAngle(x, y, start, length, angle
[,color])

lcd.pixel(x, y, [,color])

lcd.polygon(x, y, r, sides, thick [,color,
fillcolor, rotate])

lcd.print(text, x, y, [,color])

lcd.rect(x, y, width, height [,color, fillcolor])

lcd.restorewin()

lcd.roundrect(x, y, width, height, r [color,
fillcolor])

lcd.savewin()

lcd.screensize()

lcd.set_bg(color)

lcd.setBrightness(brightness)

lcd.setRotation(angle)

lcd.setwin(x, y, x1, y1)

lcd.textClear(x, y, text [, color])

lcd.triangle(x1, y1, x2, y2, x3, y3, [,color,
fillcolor])

LIGHTGREY

Light Unit

Loop

M

M5MQTT

M5UI

MCP4725

MFRC522

m5mqtt = M5mqtt(id, server, port, user,
password, keepalive)

m5mqtt.publish(topic, msg)

m5mqtt.start()

m5mqtt.subscribe(topic, callback)

map_value(value, from_low, from_high,
from_low, from_high)

mac_addr

Makey Unit

MAROON

MAGENTA

MISO

MOSI

MPU6050

N

NAVY

NCIR Hat

NCIR Unit

O

OLIVE

ORANGE

P	
Pa.Hub	RED
Pb.Hub	Remote
P2P	remoteInit()
Pin	rfid = unit.get(face.RFID)
pin0 = machine.Pin(pin [, mode, pull, value])	rfid.writeBlock(block, data)
pin0.on()	RFID Face
pin0.off()	RFID Unit
pin0.value()	RGB BAR
pin0.value(state)	rgb.setColor(pos, color)
PINK	rgb.setColor(posBegin, posEnd, color)
PIR Hat	rgb.setColorAll(color)
PIR Unit	rgb.setBrightness(brightness)
PORTA	RGB LED
PORTB	RGB Unit
PORTC	Relay Unit
Power	RoverC
power.getBatteryLevel()	S
power.isChargeFull()	SD
power.isCharging()	SendP2PData(APIKey, data)
PuppyC	setScreenColor(color)
pre = unit_env.pressure	state = i2c1.available()
PURPLE	Servo
PWM	Servo Hat
Pulse Width Modulation	Servo8 Hat
pwm0 = machine.PWM(pin, freq, duty, timer)	Servo Module
pwm0.freq(freq)	SPK
pwm0.duty(duty)	Speaker
pwm0.pause()	speaker.tone(freq,duration,volume)
pwm0.resume()	speaker.sing(freq, beat)
	speaker.setVolume(volume)
Q	state = per.state
	state = rfid.isCardOn()
R	T
Rectangle	Textbook
rect = M5Rect(x, y, width, height, fillColor,	TFT
borderColor)	tft.arc(x, y, r, thick, start, end [color,
rect.hide()	fillcolor])
rect.setBgColor()	tft.attrib7seg(dist, width, outline, colour)
rect.setBorderColor()	tft = lcd
rect.setPosition(x=10, y=10)	tft.circle(x, y, r [,color, fillcolor])
rect.setPosition(x=10)	tft.clear([color])
rect.setPosition(y=10)	tft.clearWin([color])
rect.setSize(width=10, height=10)	tft.deinit()
rect.setSize(height=10)	tft.ellipse(x, y, rx, ry [opt, color, fillcolor])
rect.setSize(width=10)	tft.font(font [,rotate, transparent, fixedwidth,
rect.show()	dist, width, outline, color])

tft.setFontSize()	timerSch.stop(timer)
tft.get_bg()	Title
tft.get_fg()	title = M5Title(title=text, fgcolor=color1, bgcolor=color2)
tft.hsb2rgb(hue, saturation, brightness)	title.hide()
tft.image(x, y, file [, scale, type])	title.show()
tft.init(type, mosi=pinnum, miso=pinnum, clk=pinnum, cs=pinnum [, opt_args])	title.setBgColor(color)
toggleIO(pin)	title.setFgColor(color)
tft.LANDSCAPE	title.setTitle(text)
tft.LANDSCAPE_FLIP	tmp = unit_env.temperature
tft.line(x, y, x1, y1 [,color])	Touch Panel
tft.lineByAngle(x, y, start, length, angle [,color])	TOF Hat
tft.orient(orient)	TOF Unit
tft.poly(x, y, r, sides, thick, [color, fillcolor, rotate])	Trace Bar
tft.PORTRAIT	 U
tft.PORTRAIT_FLIP	UART
tft.triangle(x, y, x1, y1, x2, y2 [,color, fillcolor])	uart.any()
tft.pixel(x, y [,color])	uart.init(baud, bits, parity, stop)
tft.readPixel(x, y)	uart = machine.UART(channel, tx_pin, rx_pin)
tft.readScreen(x, y, width, height [, buff])	uart.read()
tft.rect(x, y, width, height, [color, fillcolor])	uart.read(10)
tft.resetwin()	uart.readline()
tft.restorewin()	uart.write(ch)
tft.roundrect(x, y, width, height, r [color, fillcolor])	uid = rfid.readUid()
tft.screensize()	unit_env = unit.get(unit.ENV, unit.PORTA)
tft.savewin()	unit_Neopixel = unit.get(unit.NEOPIXEL, unit.PORTB, number)
tft.set_bg(color)	unit_Neopixel.setBrightness(brightness)
tft.set_fg(color)	unit_Neopixel.setColor(posBegin, posEnd, color)
tft.setwin(x, y, x1, y1)	unit_Neopixel.setColor(pos, color)
tft.text(x, y, text [, color])	unit_Neopixel.setColorAll(color)
tft.textClear(x, y, text [, color])	unit_pir = unit.get(unit.PIR, unit.PORTB
tft.textWidth(text)	uos.mountsd()
tft.tft_deselect()	uos.sdconfig(uos.SDMODE_SPI,clk=18,mosi =23,miso=19,cs =4)
tft.tft_readcmd(cmd, len)	
tft.tft_setspeed(speed)	
tft.tft_select()	
tft.tft_writecmd(cmd)	
tft.tft_writemddata(cmd, data)	
tft.winsize()	V
Thermal Hat	
Thermal Unit	
TIME	
timerSch.run(timer, interval, mode)	W
timerSch.setTimer(timer, interval, mode)	Weight Unit
	WHITE
	WIFI
	WIFICFG
	wifiCfg.autoConnect(lcdShow=True)
	wifiCfg.doConnect(ssid, pswd)

```
wifiCfg.screenShow()  
wifiCfg.wlan_sta.isconnected()
```

X

Y
YELLOW
YUN
ypr = imu.ypr

Z