Core S3 Development Kit

Written by
Adam Bryant

Introduction.

The CoreS3 Development Kit is the latest generation in the M5Stack Core Development Kit series. The CoreS3 has been designed around the ESP32-S3, dual-core Xtensa LX7 processor, running at a speed of 240MHz and comes with built in ‘WiFi functions, and has onboard 16MFLASH and 8M-PSRAM.

CoreS3 Development Kit Features

The CoreS3 features consist of:

- ESP32-S3
  - Duel Core Xtensa LX7 CPU running @ 240Mhz,
  - 2.4Ghz Wifi,
  - 16MB Flash,
  - 8MB PSRam.

- AXP2101 Power Management chip controlling power from:
  - USB-C 5V,
  - DC 9-24V (though the base connector),
  - 3.7V 500mAH battery located in the base.

- Built in camera featuring:
  - Galaxy Core GC0308 0.3MP camera,
  - LTR-553ALS-WA Light Sensor,
  - 640X480 (VGA) resolution @ 30fps,
  - 30cm to 500CM Focal distance,
  - M12X0.5 thread Lens.

- 2.0” IPS LCD
  - 320X240 pixel resolution,
  - Capacitive Multi Touch,
- Duel Microphones connected to a ES7210 Audio decoder,
- AW88298 Amplifier,
  - 1W Speaker,
- BM1270 6 Axis IMU,
- BMM150 Geomagnetic Sensor
- BM8563 RTC,

Powering the Core

As mentioned earlier, there is three ways in which the CoreS3 can be powered.
On the left hand side you will find the USBC port which allows you to power the CoreS3 from a host PC or a USB battery pack. In UiFlow mode, the CoreS3 draws 4.92V @0.13A.

On the bottom of the CoreS3 we have a power jack which accepts 9 to 24V DC and is run through the power switch next to it to allow the CoreS3 to be switched off when not connected via USB. The ``size of the Jack is 5.5mm OD and 2.1mm pin size.

Exploring the Outside.

Front of the Core S3

On the front of the Core S3 we have the 320X240 pixel (2.0”) IPS LCD. Covering most of the from is the capacitive multitouch screen sensor with three dedicated touch zones available in code as buttons A, B and C. These are not clearly marked as zone A shares the same location and the left microphone, Zone B shares the same location as the camera and zone C shares the same location as the right microphone. Between zones B and C you will find two LEDs that are the LTR-553ALS light sensor.

Bottom of the Core S3

On the bottom of the Core S3 Development kit you will find the. Micro SD card slot, and the reset button which is used to reset the CoreS3 but if held down for more then three seconds, will put the Core S3 into Programming mode. On the base you will find the DC 9-24V power jack and the power switch that provides power from the base to the core.

Left side of the CoreS3

On the left we find the core’s power button which you need to hold down for five seconds to power off, the USB programming/ communications port, and the red Port A I2C connector for connecting to I2C ‘Grove) units

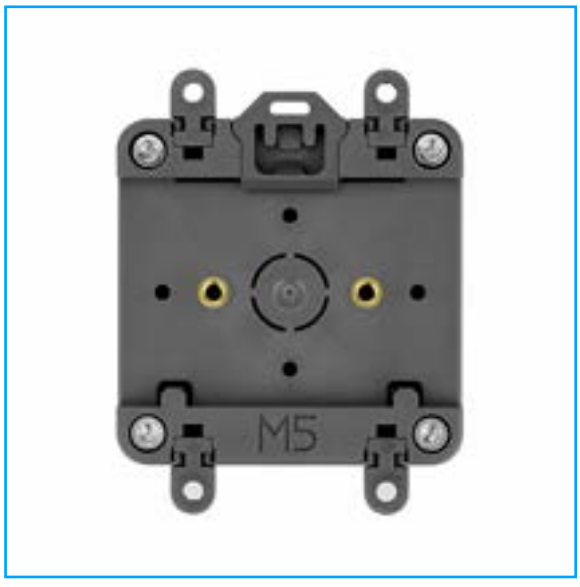
Top of the Core S3

The Top of the Core is clear but on the Base you will find the black Analog “Grove” port and the blue UART “grove port.

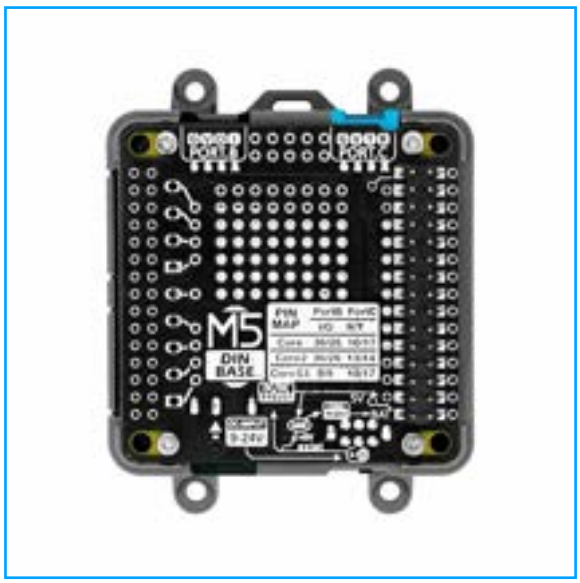
Right Side of the Core S3

On the right side you will find three rows of holes which cover the speaker.

The CoreS3 Base



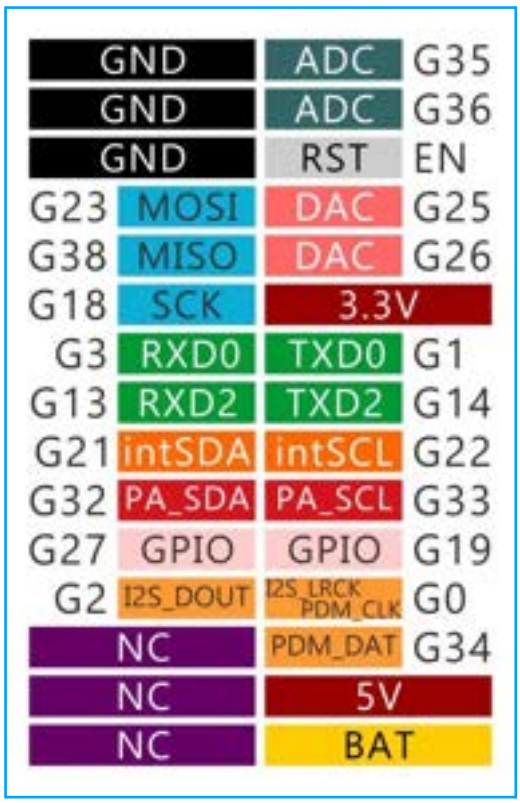
On the base of the Core S3 you will find for removable mounting lugs for surface mounting the CoreS3, two M3 threaded brass inserts for mounting from behind and the recess section is for mounting to “Top Hat” Din Rail.



On the fight hand side you can see the “MBUS” connector which is a 15X2 pin connector with the following pinout:

GND			ADC	G10
GND			PB_IN	G8
GND			RST/EN	
G37	MOSI		GPIO	G5
G35	MISO		PB_OUT	G9
G36	SCK		3.3V	
G44	RXD0		TXD0	G43
G18	PC_RX		PC_TX	G17
G12	intSDA		intSCL	G11
G2	PA_SDA		PA_SCL	G1
G6	GPIO		GPIO	G7
G13	I2S_DOUT		I2S_LRCK	G0
NC			I2S_DIN	G14
NC			5V	
NC			BAT	

Unfortunately you will see that the pinout is different to the Core2 meaning that not all modules will be compatible.



You can find a sticker on the inside of the CoreS3 stuck to the shield which has the MBus pinout printed on it.

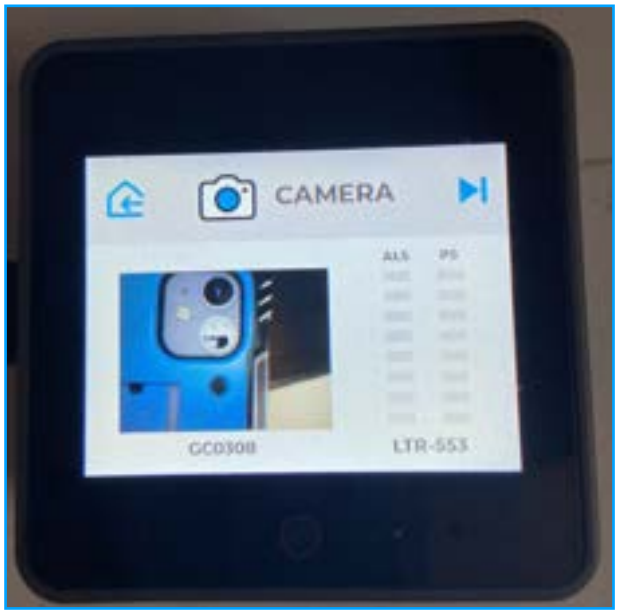
Factory Demo

The CoreS3 Development kit will normally come with the factory test firmware preinstalled from M5Stack. When first powered up the factory test will display a screen showing the following:

This is the initial boot screen for the factory demo and shows the internal hardware laid on top of the ESP logo. Tapping the screen brings us into the factory demo:

As the WIFI logo is the first icon on the right, tapping on it will bring up the wifi scanner that shows wifi networks in the area. Unfortunately, due to the limitations of the WIFI stack used in the ESP32, only 2.4GHz channels will be detected.

The next screen is an example displaying the image captured from the built in camera located in the bottom bezel.



Here you can see the camera recording the camera used to take the screen photos. Due to the overhead lighting on the desk the LTR-553 light sensor doesn't seam to want to read anything.

This screen is showing the audio being picked up by the duel microphones.

Next we have the power monitoring example. If you remove the USB cable when powered you will see the screen change to show power coming from the battery.

Next we have the IMU example. The scrolling message is informing users that the IMU needs calibrating first by pressing on the centre of the screen until you hear a bleep and then moving the CoreS3 in a figure of 8 until the second bleep to calibrate.

Next we have the SDCard example which list the files on an SDCard along with the file type and the size of the file.

After the SDCard example we have the touchscreen example which draws on the screen when you touch the white area.

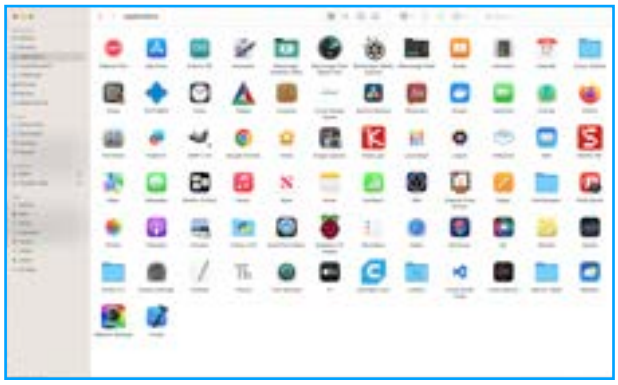
Last is the I2C scanner which detects I2C devices connected to the internal I2C bus. Connecting any I2C devices to the Red I2C port will not show the new devices address.

Tapping the icon on the top right will open the “Shutdown menu”

In this screen we can change the screen backlight brightness, shutdown the CoreS3 or Put it to sleep. Unfortunately the backlight setting is now saved and when the CoreS3 is rebooted, it will return to full brightness.

Instilling the UIFlow 2.0 Firmware with M5Burner.

In order to install the UIFlow2.0 firmware used to program the CoreS3 we need to download M5Burner from the M5Stack website here: https://docs.m5stack.com/en/download?ref=pfpqkvphmgr
Once downloaded, you need to save M5Burner to a stable location, On OSX based computer, M5Burner need to be installed into the “Applications menu so that it can be found in the applications launcher.



If you don’t install it to the application folder on OSX based computers, M5Burner will throw up errors about missing development file or just not run at all.

Open M5Burner and if not already logged in, fill in your forum log in details to connect M5Burner to the UIFlow firmware.

You need to log in to the UIFlow server using M5Burner because S3 based devices connect to UIFlow2.0 in a different way that automatically add them to UIFLow 2.0 unlike with previous generation M5Stack controllers.

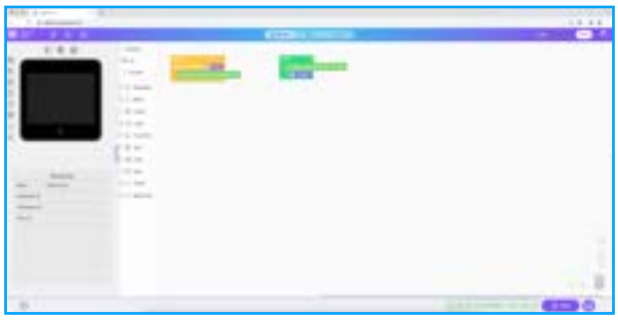
In the above screen shoot you can see what firmware is currently available to download. UIFlow firmware is normally the first firmware available with the factory demo next in the list. The Last two (at time of writing) are firmware for the StackChan personal assistants.
Click on the download button to download the latest UIFlow2.0 version (in the screenshot you can see that I have already downloaded the UIFlow2.0 firmware.
Once downloaded you need to hold the “Reset/Boot Mode” button on the bottom of the CoreS3 to get the CoreS3 into boot loader mode. You can see in this video: https://www.youtube.com/watch?v=SgDRYp2tiQw That you need to hold the button down in order to get the green led to light up for M5Burner to connect but even then it can be tricky.

Once M5Burner connects, you will need to erase the CoreS3 first before installing the new firmware, You will also notice in the video that I mistimed the button press and ended up “Softbricking” the CoreS3 and spent ages trying to reset it to restart the burn process. Once you erase the CoreS3 you will be brought back to the main screen shown above. Click on the “Burn” button again and this time you will see the following message asking if you want to bind the CoreS3 to your account.

Unlike with previous generations of M5Stack controllers where you had to manually add the device code into UIFlow, UIFlow2 automatically adds any ESP32-S3 based controller to your account which is created when you order direct from M5Stack or when you register with the forums found here: https://community.m5stack.com/

Click the “yes” button to add the CoreS3 to your account and continue the burning process.

Programming In UIFlow 2.0

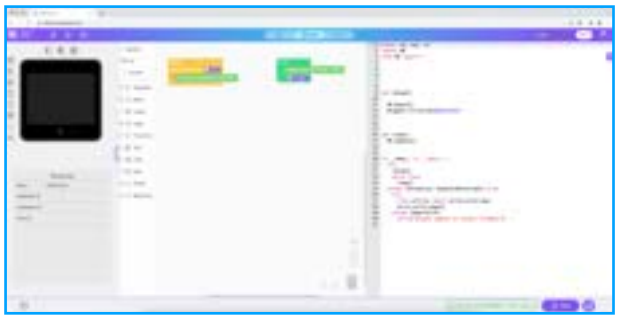


The Primary method for programming the CoreS3 Development kit is to use UIFLow which is a graphical IDE (integrated Development Environment) built on top of Micropython. Because UIFlow uses the Micropython programming language it means that if you have used any Adafruit or Raspberry pi devices, then you are already familiar and experienced in the Micropython.

UIFlow consist of three main parts as shown in the screenshot above. On the left is the GUI (Graphical User Interface) editor showing the CoreS3 which is used for laying out GUI elements while below we have the new “Resources” selector that allows you to chose which function and units to add to your programs.

In the middle we have the block menu where you can select which code blocks to add to the program. When you select function and units with the resource selector additional blocks will be added to this list.

On the right we have the programming environment which is used to assemble blocks into code.
The three icons above the environment allow you to select between the default mode as show, block and Micropython Code view:



Or just Micropython code view:

While you can make changes in the block mode and view the code changes in the Micropython mode, you cant make changes in the micropython and see changes in the block mode as any new code you add will not have the corresponding code blocks.

Selecting the CoreS3 for programming.

Once you have the latest UIFlow2 firmware installed and bound to you account, you need to select it. At the bottom of the page next to the “Run” button you may find an red shape with the text “Select Device”, click on this to open the UIFlow2 device menu:

If this is your first time then it may be empty or showing one device. I have four ESP32-S3 based devices that I am testing and so there are four visible. Click on the CoreS3 and you should see the code shown on the Core3’s screen but with most of the characters replaced with stars. The circle at the top fight of the icon should show green if the CoreS3 is connected or red if it cant be found by the M5Stack server.

Make sure the Core S3 is still selected and click on confirm to use the CoreS3 in UIFlow.

While UIFlow2 is the preferred method for programming the CoreS33 based devices, because the firmware is built on micropython, you can use other programming environments to program the CoreS3. With each block I will show the API call as shown by UIFlow in the Micropython View.

Hello World.

In order to get used to UIFlow, lets start with the basic “Hello World” example.

To the Left of the virtual GUI editor we have eight GUI icons representing the eight GUI elements we can use while above we have three icons for changing how the GUI editor is viewed. By default, when UIFlow is loaded the first view will be set making the GUI editor take up between 1/4 to 1/3 of the available screen. The middle icon looks like it is supposed to set the layout to use half of the screen but doesn’t seam to work while the right icon makes the GUI editor take over the full screen.

In micropython the GUI elements are part of the “Widgets” submodule found inside of the “M5”Module.
To access the modules directly in micropython you need to use another IDE like Thonny or Mu, connect to the CoreS3 in shell mode and then type:

```
import M5
```

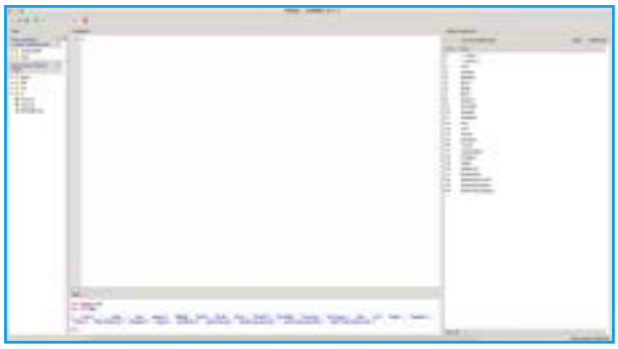
To import the module and then

```
dir(M5)
```

to show the submodules.

Micropython is running in REPL mode in the shell and is case sensitive. If you get a “Traceback” error check that you have spelled the modules with the correct upper or lower case letters.

When using Thonny we also get a second list shown on the right of the screen with the modules:

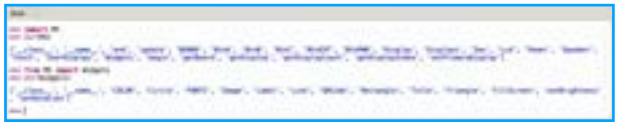


To view the Widgets modules type:

```
from M5 import Widgets
```

Followed by:

```
dir(Widgets)
```



The eight GUI elements available in the editor are as follows:

Title.

The Title elements add a title bar to the top of the CoreS3's screen. By default the title bar will be blue (#0000ff) with white (#ffffff) text in the Dejavu Sans 18 font.

The GUI editor give the ability to change the initial setting and colours for basic operation but for more functions we need to switch back to the normal UIFlow2.0 mode by click on the "Back" button.

Here we can see a new menu item has been added called "Title". Clicking on "Title" we display the following new block functions:

Set Title Text X Position

The set title text X position block is used to align the title text in the title bar. The position can be any number but over 320 will place the text off the screen.

The Micropython api for set title position is as follows:

```
title0.setTextCursor(x=0)
```

Set Title Text Colour and Background Colour

The Set tile text and background colour block is used to change the text and background colours and can be used for indicating a status change.

The Micropython api for set title text colour is as follows:

```
title0.setColor(text_c=0x6600cc,
bg_c=0x6600cc)
```

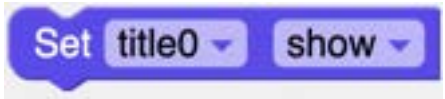
Set Title Text

The set title text block is used to change the text shown in the title bar. The text can be any string or any values returned from sensors and converted to a string.

The Micropython api for set title text is as follows:

```
title0.setText('Title')
```

Set Title Show



The Set title Show block is used to set the visibility of the title bar so that it can be hidden when not needed.

The Micropython api for set title show is as follows:

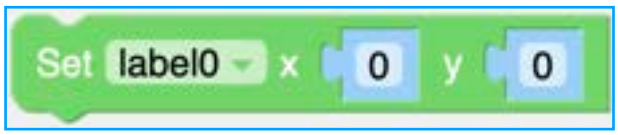
```
title0.setVisible(True)
```

Label

Next we have the Label blocks. There are two different Label blocks, the Label and Label+ for regular text display you use the Label where as for retrieving data from a page, you use Label+.

To add a Label or Label+ to the screen, drag the Label or Label+ on to the screen and the text label will appear in white with a black background.

Set Label X Y

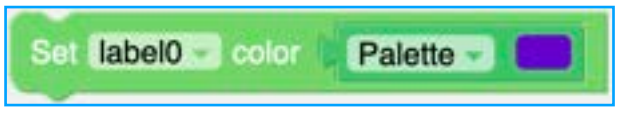


The set label XY block is used to manually override the position of the label on a screen allowing the position to be changed and moved around.

The Micropython API for the set label X Y is:

```
label0.setCursor(x=0, y=0)
```

Set Label Color



The Set Label colour block is used to change the colour of the labels text. If you click on the “Palette” block’s dropdown arrow you get the option to change colour by selecting from the pallet, setting using a hexadecimal value:

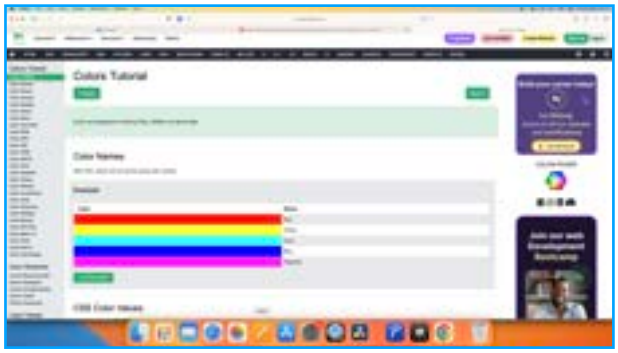
Or by setting the decimal value:

The Micropython API for the Set colour block is:

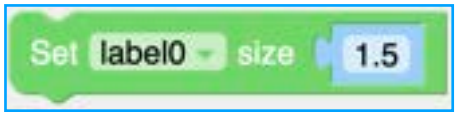
```
label0.setColor(0xff0000)
```

And is always a Hexadecimal value.

For a list of the colours and values you can find many charts and tables online but the best resource so far is the colour subset of https://www.w3schools.com/colors/default.asp



Set Label Size



The set label size block is used to set the font size of the label se in the Set Label Font block and is used to increase or decrease the default fault set by that block.

The Micropython API for this block is:

```
label0.setSize(1.5)
```

Set Label Text

The set label text block is used to show a string of text. Because the text section is also a space for a value block, it can be used to show the value of a string formatted variable or the values returned from sensors.

The Micropython API for the set label text box is:

```
label0.setText(str(''))
```

Set Label Font

The Set Label Font is used to control which font is used by the Label to display the text. Currently their only seven different sizes of the DejaVu font along with three none English fonts for the Chinese, Japanese and Korean languages.

The Micropython code for the font block is:

```
label0.setFont(Widgets.FONTS.DejaVu9)
```

Set Label Show

The set label show block is used to make text visible or hidden during a programs runtime. Set the block to “Show” to make the text visible or “hide” to make the text invisible.

The Micropython API for the Set label show block is:

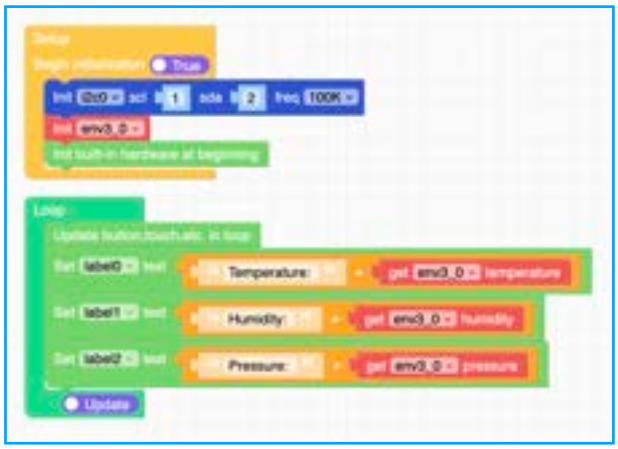
```
label0.setVisible(True)
```

Or

```
label0.setVisible(False)
```

Label Example

In the following example I use three labels to show the temperature, pressure and humidity from the ENVIII unit.



When run on the Core S3, the screen will display the following:

The Micropython code for this example is as follows:

```
import os, sys, io
import M5
from M5 import *
from hardware import *
from unit import *


label0 = None
label1 = None
label2 = None
i2c0 = None
env3_0 = None


def setup():
```

```
  global label0, label1, label2,
i2c0, env3_0

  i2c0 = I2C(0, scl=Pin(1),
sda=Pin(2), freq=100000)
  env3_0 = ENV(i2c=i2c0, type=3)
  M5.begin()
  Widgets.fillScreen(0x222222)
  label0 = Widgets.Label("Text",
3, 38, 1.0, 0xffffff, 0x222222,
Widgets.FONTS.DejaVu18)
  label1 = Widgets.Label("Text",
11, 87, 1.0, 0xffffff, 0x222222,
Widgets.FONTS.DejaVu18)
  label2 = Widgets.Label("Text",
6, 129, 1.0, 0xffffff, 0x222222,
Widgets.FONTS.DejaVu18)



def loop():
  global label0, label1, label2,
i2c0, env3_0
  M5.update()

label0.setText(str(((str('Temperat
ure: ') +
str((env3_0.read_temperature()))))
))

label1.setText(str(((str('Humidity
: ') +
str((env3_0.read_humidity()))))))

label2.setText(str(((str('Pressure
: ') +
str((env3_0.read_pressure()))))))


if __name__ == '__main__':
  try:
    setup()
    while True:
      loop()
  except (Exception,
KeyboardInterrupt) as e:
    try:
      from utility import
print_error_msg
      print_error_msg(e)
    except ImportError:
      print("please update to
latest firmware")
```

Label+

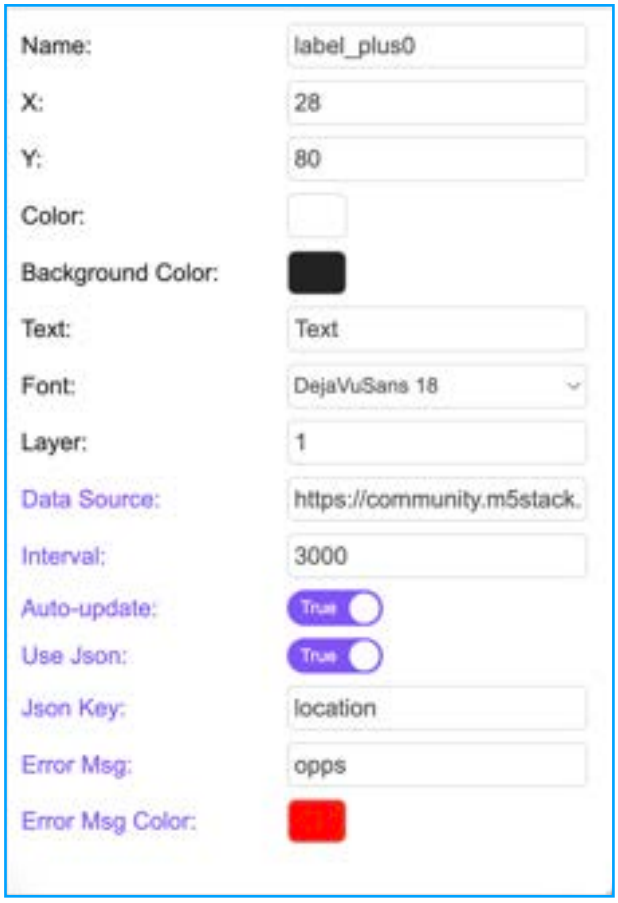
The label+ blocks are used to retrieve data from a source and display them on the screen as a label.

In order to use the Label+ blocks you need to supply a source address in the setting panel on the right. For example, the Label+can be used to retrieve forum public data for example:

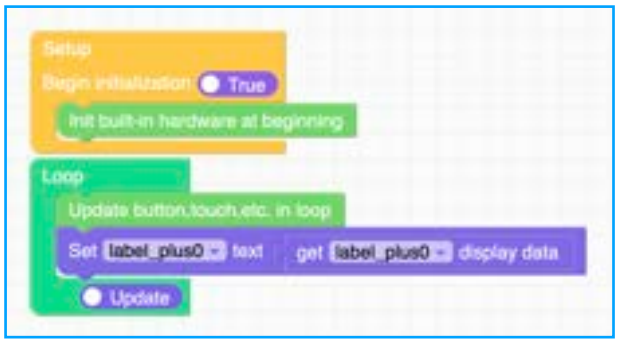
Here I have set the address to the user public data file which is a JSON file with:

`https://community.m5stack.com/api/user/<user_name>/`

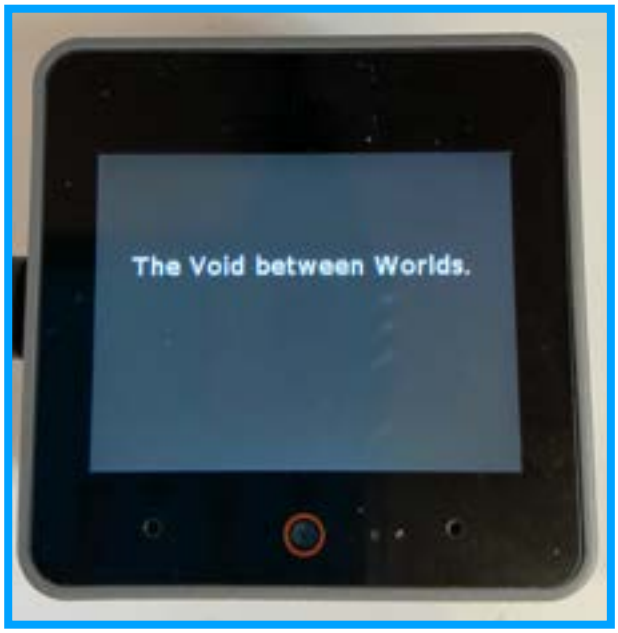
And chose the JSON field of Location:



Then using the following UIFLow2 code (UIFlow1.X code is almost the same):



The Core S3 shows on the screen what my location is set to in the forum:



Get Display Data

The get display data block is used to return data from the field specified in the settings. In the previous example I used the “location” field and so the location “The Void between Worlds.” That I specified in my forum profile is shown on screen.

You can retrieve numerical data from a field but it needs to be converted into a text string with the text blocks as shown below.

The Micropython api for the Get display data block is:

```
label_plus0.get_data()
```

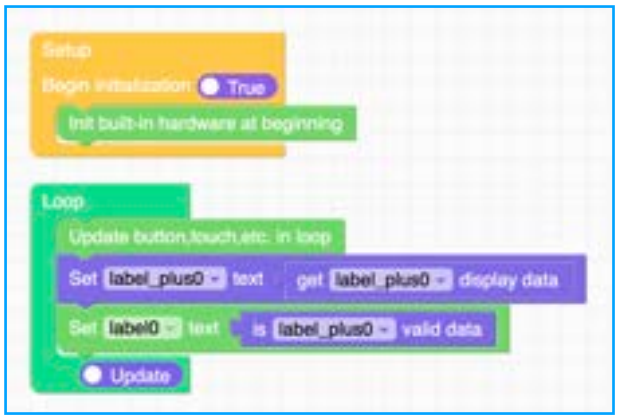
The Micropython code for the example code is as follows:

```
import os, sys, io
import M5
from M5 import *
from label_plus import LabelPlus

label_plus0 = None
label0 = None

def setup():
  global label_plus0, label0

  M5.begin()
  Widgets.fillScreen(0x222222)
  label_plus0 = LabelPlus("Text",
10, 28, 1.0, 0xffffff, 0x222222,
Widgets.FONTS.DejaVu18, "https://
community.m5stack.com/api/user/
ajb2k3/", 3000, True, "location",
"error", 0xFF0000)
  label0 = Widgets.Label("Text",
11, 108, 1.0, 0xffffff, 0x222222,
Widgets.FONTS.DejaVu18)
```

```
def loop():
  global label_plus0, label0
  M5.update()

label_plus0.setText(str((label_plu
s0.get_data())))


if __name__ == '__main__':
  try:
    setup()
    while True:
      loop()
  except (Exception,
KeyboardInterrupt) as e:
    try:
      from utility import
print_error_msg
      print_error_msg(e)
    except ImportError:
      print("please update to
latest firmware")
```

Is Valid Data

The Is Valid data block is used to check if there is a valid data stream at the location and will return True or False depending on what it finds.
In the following example I have just extended the previous example to run a check to see if the data is valid.



You can see here that I just used a basic label for the check as adding a second Set Label+ box will send a second stream request to the stream host instead of just checking for the presence of the valid data stream.

The Micropython api call for this block is:

```
label_plus0.is_valid_data()
```

And the Micropython complete code for the example is:

```
import os, sys, io
import M5
from M5 import *
from label_plus import LabelPlus

label_plus0 = None
label0 = None

def setup():
  global label_plus0, label0

  M5.begin()
  Widgets.fillScreen(0x222222)
  label_plus0 = LabelPlus("Text",
10, 28, 1.0, 0xffffff, 0x222222,
Widgets.FONTS.DejaVu18, "https://
community.m5stack.com/api/user/
ajb2k3/", 3000, True, "location",
"error", 0xFF0000)
  label0 = Widgets.Label("Text",
11, 108, 1.0, 0xffffff, 0x222222,
Widgets.FONTS.DejaVu18)

def loop():
  global label_plus0, label0
  M5.update()

label_plus0.setText(str((label_plu
s0.get_data())))

label0.setText(str((label_plus0.is
_valid_data())))

if __name__ == '__main__':
  try:
    setup()
    while True:
      loop()
  except (Exception,
KeyboardInterrupt) as e:
    try:
      from utility import
print_error_msg
      print_error_msg(e)
    except ImportError:
      print("please update to
latest firmware")
```

Set Update Enable

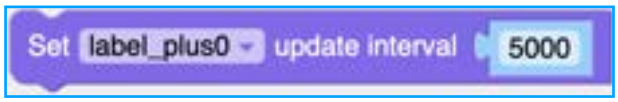
The Set Update Enable block is used to control if the Label+ will constantly check the source JSON data file for changes overriding the initial setting in the dialog.

The Micropython code for this block is:

```
label_plus0.set_update_enable(False
)

Or

label_plus0.set_update_enable(True)
```

This block can be used with the Set Update Interval block to control how often the code will check the source JSON data file.

Set Update Interval

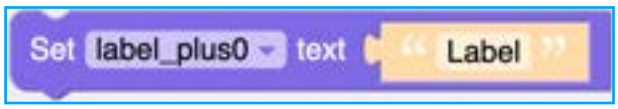


The Set Update Interval block is used to control the delay between update request sent by the code. The block can be used with the previous block to control when and how ofter to look for updates.

The Micropython code for this block is:

```
label_plus0.set_update_period(5000
)
```

Set Label+ Text

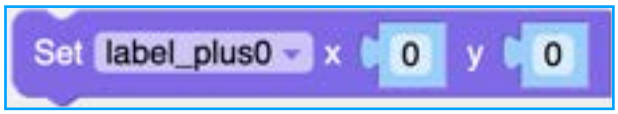


The set label text block is used to show a string of text. Because the text section is also a space for a value block, it can be used to show the value of a string formatted variable or the values returned from sensors.

The Micropython API for the set label text box is:

```
label_plus0.setText(str(''))
```

Set Label+ X Y

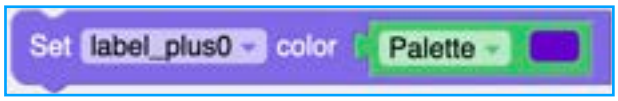


The set label+ XY block is used to manually override the position of the label on a screen allowing the position to be changed and moved around.

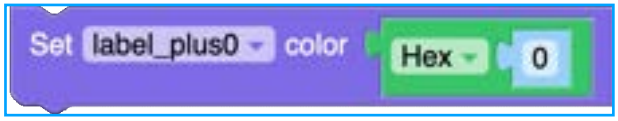
The Micropython API for the set label+ X Y is:

```
label_plus0.setCursor(x=0, y=0)
```

Set Label+ Color



The Set Label+ colour block is used to change the colour of the labels text. If you click on the “Palette” block’s dropdown arrow you get the option to change colour by selecting from the pallet, setting using a hexadecimal value:



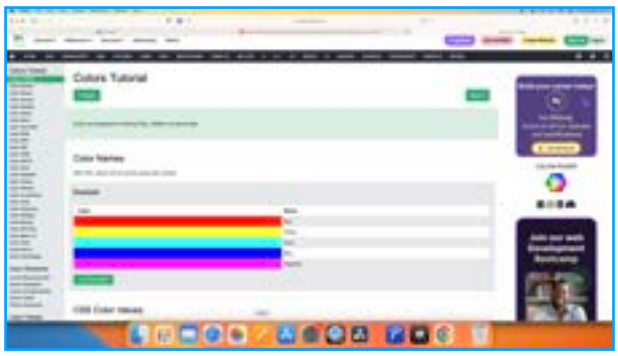
Or by setting the decimal value:

The Micropython API for the Set colour block is:

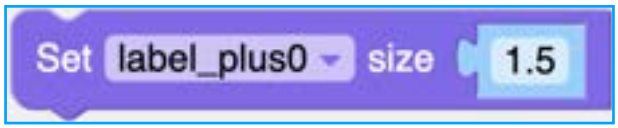
```
label_plus0.setColor(0xff0000)
```

And is always a Hexadecimal value.

For a list of the colours and values you can find many charts and tables online but the best resource so far is the colour subset of https://www.w3schools.com/colors/default.asp



Set Label+ Size



The set label+ size block is used to set the font size of the label se in the Set Label Font block and is used to increase or decrease the default fault set by that block.

The Micropython API for this block is:

```
label_plus0.setSize(1.5)
```

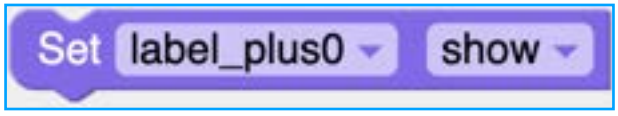
Set Label+ Font

The Set Label+ Font is used to control which font is used by the Label+ to display the text. Currently their only seven different sizes of the DejaVu font along with three none English fonts for the Chinese, Japanese and Korean languages.

The Micropython code for the font block is:

```
label0.setFont(Widgets.FONTS.DejaVu9)
```

Set Label+ Show



The set label+ show block is used to make text visible or hidden during a programs runtime. Set the block to “Show” to make the text visible or “hide” to make the text invisible.

The Micropython API for the Set label show block is:

```
label_plus0.setVisible(True)
```

Or

```
label_plus0.setVisible(False)
```

Image

The Image element is used to load and display images from memory onto the CoreS3’s screen.

There are requirements for images in that they must be less than 320x240Pixels in size, must be less than 100KB in size when saved and have a filename of less than 30 characters long.

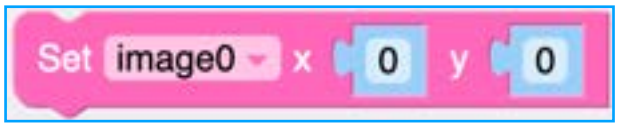
The default settings for the Image element is as follows:

The settings allow you to change the name that the functions will use, the default X and Y location, and the GUI layer that the image will be stored on. The dropdown box is for selecting the initial image to be displayed from those in the memory. The first button is for uploading an image to the CoreS3 and the next button is used to refresh the list if an uploaded image doesn’t show.

There are four code blocks that are available for controlling images consisting of:

Set Image X and Y,
Set Image (textbox),
Set Image (dropdown menu):
Set Image Show.

Set Image X and Y



The Set Image X and Y block is used to set or move images around the screen from within code.

The Micropython API for the set label X Y is:

```
Image0.setCursor(x=0, y=0)
```

Set Image

The two Set Image blocks which looking the same have slightly different operating methods. While the Micropython API looks the same for both:

```
image0.setImage("res/img/
extio2.jpeg")
image0.setImage("res/img/
default.png")
```

The big difference is that the dropdown version of the block will only allow images to be selected from the res/img folder, the text block version allows users to set images stored in other locations.

Set Image Show/Hide

The Set image Show/Hide block is used for setting if an image is visible or hidden from within code.
The Micropython API for this block is:

```
image0.setVisible(True)
```

Or

```
image0.setVisible(False)
```

Image+

The Image+ element are used to load images from an online source. While this opens up possibilities, the same image restrictions still apply.

The Default settings for the Image+ element is as follows.

In order to use the Image+ blocks you need to supply a source address in the setting panel on the right. For example, the Image+can be used to retrieve forum public data for example, in order to display the forum avatar image on a CoreS3 screen, you need to type in the location of the image into the Data Source box shown in the image above. To find the address, the simple way is to open your forum profile in a web browser,

right click the image to open in a new tab and then just copy the address shown in the address bar.

The blocks for controlling the Image+ functions are as follows:

Set Image+ Update Enable

The Set Image+ Update enable function can be used used if the source image changes for example, a camera feed. If as in my example above, you are only referencing a fixed none changing image, make sure this is set to false otherwise the source will be inundated with request which could result in the server hosting the image going offline.

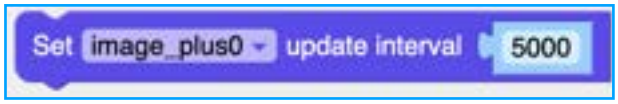
The Micropython API for the Image+ update function is:

```
image_plus0.set_update_enable(False)
```

Or

```
image_plus0.set_update_enable(True)
```

Set Image+ Update Interval



The Set Image+ update Interval block is used as a delay between request for updated images from the image source. This period is measured in milliseconds.

The Micropython api for the Set Update Image Interval is:

```
image_plus0.set_update_period(5000
)
```

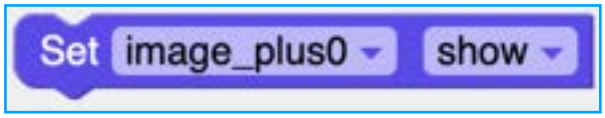
Set Image+ X and Y

The Set Image+ Image X and Y block is used to set or move images around the screen from within code.

The Micropython API for the Set Image+ X and Y is:

```
image_plus0.setCursor(x=0, y=0)
```

Set Image Show/Hide



The Set Image Show/Hide block is used for setting if an image is visible or hidden from within code.
The Micropython API for this block is:

```
image_plus0.setVisible(True)
```

Or

```
image_plus0.setVisible(False)
```

Rectangle.

The Rectangle GUI element shares many functions with the other units but is designed to create square or rectangle elements on the screen.

The default settings for the Rectangle element is as follows:

Unlike previous elements we not have a border and body colour.

Set Rectangle X and Y

The Set Rectangle Image X and Y block is used to set or move images around the screen from within code.

The Micropython API for the Set Rectangle X and Y is:

```
rect0.setCursor(x=0, y=0)
```

Set Rectangle Boarder Colour and Body Fill Colour

The set Rectangle Border Colour and Body Fill colour block is used to control the border and infill colour of the rectangle. By default it is set to white and can be changed in the GUI editor but this block allows that set colour to be over ridden in code.

You can also set the colour using RGB:

Or with Hexadecimal:

The Micropython api for set Rectangle Border Colour and Body Fill colour is as follows:

```
rect0.setColor(color=0x6600cc,
fill_c=0x6600cc)
```

Set Rectangle Width and Height

The Set Rectangle Width and Heigh block is used to set or move images around the screen from within code.

The Micropython API for the Set Rectangle Width and Height is:

```
rect0.setSize(w=0, h=0)
```

Set Rectangle Show/Hide

The Set Rectangle Show/Hide block is used for setting if an image is visible or hidden from within code.
The Micropython API for this block is:

```
rect0.setVisible(True)
```

Or

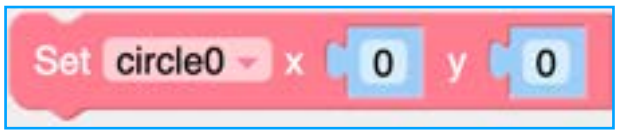
```
rect0.setVisible(False)
```

Circle

Like the rectangle, the Circle GUI element shares functions similar to the other units. The big difference between the Rectangle element and the Circle element is that the Circle does not have a height and width block but instead has a radius block.

The default settings for the Circle element is as follows:

Set Circle X and Y



The Set Circle X and Y block is used to set or move images around the screen from within code.

The Micropython API for the set Circle X and Y is:

```
circle0.setCursor(x=0, y=0)
```

Set Circle Boarder Colour and Body Fill Colour

The set Circle Border Colour and Body Fill colour block is used to control the border and infill colour of the circle. By default it is set to white and can be changed in the GUI editor but this block allows that set colour to be over ridden in code.

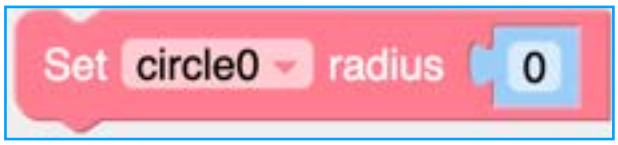
You can also set the colour using RGB:

Or with Hexadecimal:

The Micropython api for set Circle Border Colour and Body Fill colour is as follows:

```
circle0.setColor(color=0x6600cc,
fill_c=0x6600cc)
```

Set Circle Radius.



The Set Circle Radius is used to set the size of the circle. The radius is the distance from the middle of the circle to the outside edge.

The Micropython API for Set Circle radius is:

```
circle0.setRadius(r=0)
```

Set Circle Show/Hide

The Set Circle Show/Hide block is used for setting if an image is visible or hidden from within code.

The Micropython API for this block is:

```
circle0.setVisible(True)
```

Or

```
rcircle0.setVisible(False)
```

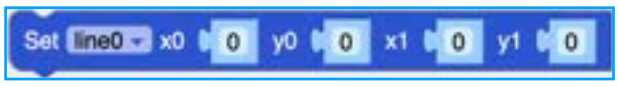
Line

The line GUI element, unlike other elements doesn't have a Set XY or a Height and width block to control the placement and size. Like the Triangle element that follows, the line instead has a block that is used to set the X and Y coordinates of each of the lines ends.

You can also see that the default setting are also less then other elements.

Because of the simplicity of the line, there is only three blocks available in UIFlow to control the line.

Set Line X0,Y0, X1,Y1



As mentioned earlier, the line element has one block that control the X&Y location for each end of the line and then Micropython draws the line between the defined coordinates.

The Micropython code for this block is:

```
line0.setPoints(x0=0, y0=0, x1=0,
y1=0)
```

Set Line Colour

The Set Line colour block is used to change the colour of the line drawn between the X and Y locations. If you click on the "Palette" block's dropdown arrow you get the option to change colour by selecting from the pallet, setting using a hexadecimal value:

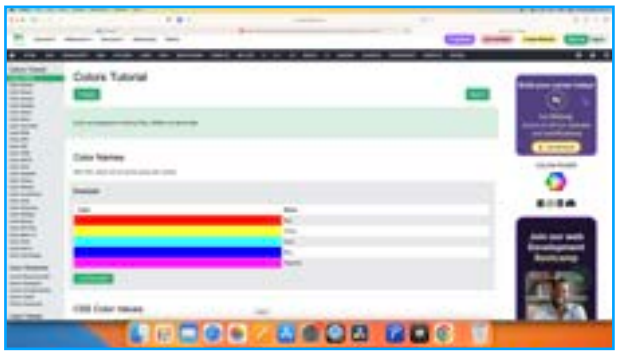
Or by setting the decimal value:

The Micropython API for the Set colour block is:

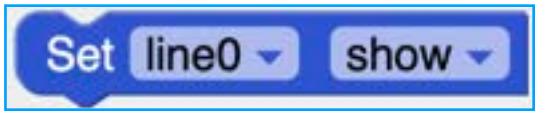
```
line0.setColor(0xff0000)
```

And is always a Hexadecimal value.

For a list of the colours and values you can find many charts and tables online but the best resource so far is the colour subset of https://www.w3schools.com/colors/default.asp



Set Line Show/Hide



The Set line Show/Hide block is used for setting if an image is visible or hidden from within code.

The Micropython API for this block is:

```
line0.setVisible(True)
```

Or

```
line0.setVisible(False)
```

Triangle

Line the Line element, the Triangle element doesn't have a SET X&Y for the whole element but instead has a block for setting the X&Y locations for the three separate corners of the triangle.
The default settings for the Triangle element are shown below:

The Triangle has three blocks available to use.

Set Triangle X0, Y0, X1, Y1, X2, Y3

As mentioned earlier, the Triangle element has one block that controls the X&Y location for each corner of the triangle and then Micropython draws the line between the defined coordinates filling the inside space in code.

The Micropython code for this block is:

```
triangle0.setPoints(x0=0, y0=0,
x1=0, y1=0, x2=0, y2=0)
```

Set Triangle Boarder Colour and Body Fill Colour

The set triangle Border Colour and Body Fill colour block is used to control the border and infill colour of the circle. By default it is set to white and can be changed in the GUI editor but this block allows that set colour to be over ridden in code.

You can also set the colour using RGB:

Or with Hexadecimal:

The Micropython api for set Circle Border Colour and Body Fill colour is as follows:

```
triangle0.setColor(color=0x6600cc,
fill_c=0x6600cc)
```

Screen

The Last set of blocks is always added to the menu as it control the screen and consists of three blocks.

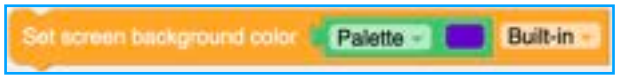
Set Screen Brightness

The set screen Brightness allows programs to control the brightness of the screen. By default, this is set to read the value from the internal setting in the firmware.

The Micropython api for set brightness is as follows:

```
Widgets.setBrightness(0)
```

Set Screen Colour



The set screen colour is used to control the background colour of the screen. By default it is set to black and can be changed in the GUI editor but this block allows that set colour to be over ridden in code.

The Micropython api for set screen colour is as follows:

```
Widgets.fillScreen(0x6600cc)
```

Set Screen Rotation

The Last screen function is the set screen rotation block and is used to control how the GUI elements are displayed on the screen. By default, the screen will show text from left to right (western style) when the camera is at the bottom of the screen.

The Micropython api for set screen rotation is as follows:

```
Widgets.setRotation(0)
```

Camera

As mentioned earlier, the CoreS3 has a 2Mp camera mounted in the centre of the bottom bezel where the “B Button” zone is found.

Support for the camera was added in release Alpha16 and can be used by dragging the camera icon onto the screen area.

To access the camera under Micropython without UIFlow you need to import the m5camera module with:

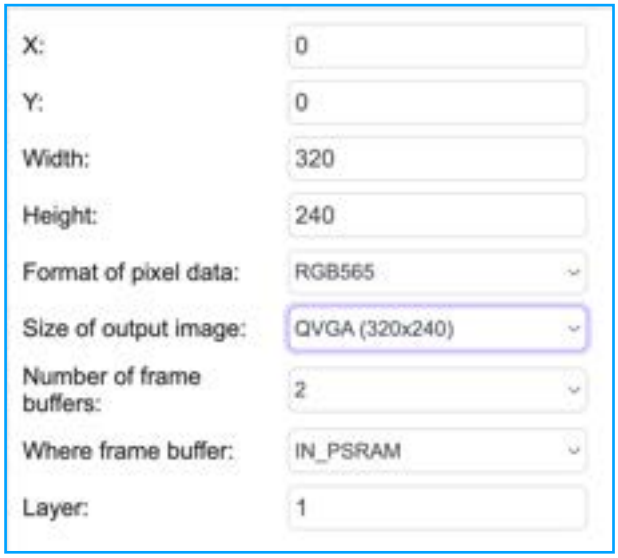
```
Import m5camera
```

And then you can view the functions with:

```
dir(m5camera)

>>> dir(m5camera)
      __class__
      __name__
      __file__
      FRAME_240X240
      FRAME_96X96
      FRAME_CIF
      FRAME_HQVGA
      FRAME_HVGA
      FRAME_QCIF
```

```
FRAME_QQVGA
FRAME_QVGA
FRAME_VGA
GRAYSCALE
M5
RGB565
YUV422
camera
capture
capture_to_bmp
capture_to_jpg
colorbar
contrast
deinit
framesize’
gc
global_gain
hmirror
init
namedtuple
pixformat
setCursor
setVisible
skip_frames
vflip
_x
_y
FrameSize
disp_to_screen
IN_DRAM
IN_PSRAM'
_frame_sizes
_width
_height
_max_width
_max_height
_visible
```

Looking at the default settings:



we see the normal X and Y position options along with the default screen size of 320x240 pixel.

Next we have an option to set the pixel format as RGB565 or YUV422 which is the colour palette encoding.

RGB565 is used to define separate 16bit colour values for the RGB channels where as YUV422 or Y UV uses 16bit values for brightness (Y) and Red and blue chrominance values.

Size of output image is used to set the captured image pixel resolution. By default this is set to QVGA (320x240) but can be changed to:

- 96x96,
- QQVGA(160x120),
- QCIF(170x144),
- HQVGA(20x176),
- 240x240,
- QVGA(320x240)
- CIF(400x296)
- HVGA(480x360)
- VGA(640x480)

Number of frame buffers is a value of sections of ram put aside for background graphics work before the graphics are sent to the screen. By default there are two buffers, one for the camera and one for the screen. The more frame buffers used, the last ram is available for program use.

Next we have the frame buffer location which allows up to place the frame buffers in PSRAM or the inbuilt SRAM.

The ESP32S3 only has 512KB of SRAM built in but M5Stack have added 8MB of PSRAM so don’t change this setting.

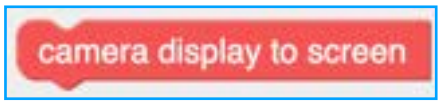
The following demo program access the content of the cameras frame buffer, passes it to the screen frame buffer before passing it to the screen for display.

You can see this example running in the June 2023 M5Stack roundup video here: June 09-06-2023 roundup

The Micropython code for this example is as follows:

```
import os, sys, io
import M5
from M5 import *
import m5camera

def setup():

  M5.begin()
  Widgets.fillScreen(0x222222)
  m5camera.init(0, 0, 320, 240,
pixformat=m5camera.RGB565,
framesize=m5camera.FRAME_QVGA,
fb_count=2,
fb_location=m5camera.IN_PSRAM)

def loop():
  M5.update()
  m5camera.disp_to_screen()


if __name__ == '__main__':
  try:
    setup()
    while True:
      loop()
  except (Exception,
KeyboardInterrupt) as e:
    try:
      from utility import
print_error_msg
      print_error_msg(e)
    except ImportError:
      print("please update to
latest firmware")
```

Display Camera to Screen.



I have started with the camera display to screen block because this is the only one used in the basic example. As mentioned previously, this block retrieves the content of the cameras frame buffer and send it to the screens frame buffer so that the screen can show the captured image data.

The Micropython API for this block is:

```
m5camera.disp_to_screen()
```

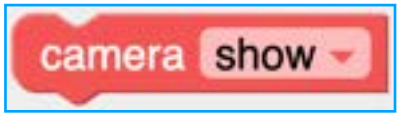
Camera Set X,Y width and Height.

The camera set X,Y, width and height block is used to override the default settings for X, Y, width and height.

The Micropython code for this is:

```
m5camera.setCursor(x=0, y=0, w=0,
h=0)
```

Camera Show



Just like with many of the other GUI elements, the camera show block is used to make the output visible or hidden.

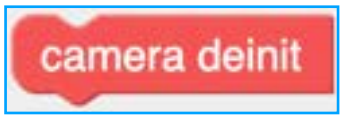
The Micropython code for this block is:

```
m5camera.setVisible(True)
```

Or

```
m5camera.setVisible(False)
```

Camera Deinit



The camera deist block is used to deactivate the camera so that it is not recording all the time.

This is useful when you need to save memory and/or battery life by deactivating the camera when its task is done.

The Micropython code for this block is:

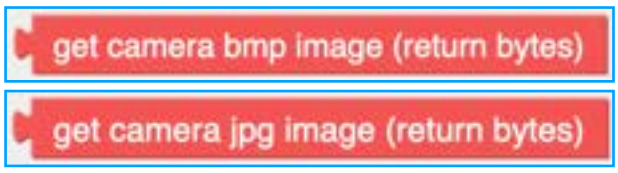
```
m5camera.deinit()
```

But to reinitialise the camera, you will need to use:

```
m5camera.init(0, 0, 320, 240,
pixformat=m5camera.RGB565,
framesize=m5camera.FRAME_QVGA,
fb_count=2,
fb_location=m5camera.IN_PSRAM)
```

In a custom code block for example:

Get camera JPG bytes and BMP Bytes



These two value blocks are used to return the number of bytes that an image is taking up in memory. Currently these blocks are not of use in UIFLow2 Alpha as the corresponding blocks for saving images and jpg's or bmp's do not exist even though the api's are in the firmware.

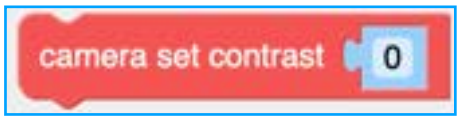
The Micropython code for using these functions is:

```
label0.setText(str((m5camera.captu
re_to_bmp())))
```

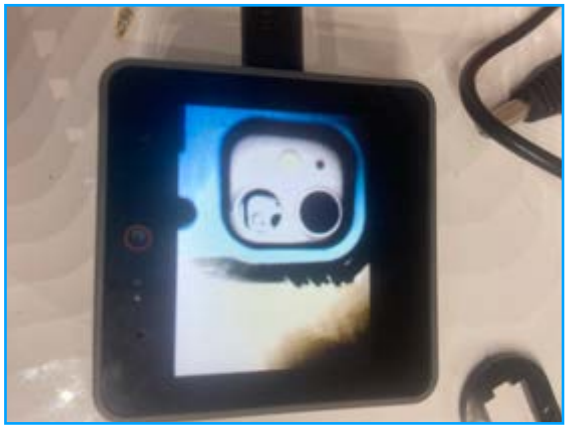
Or

```
label0.setText(str((m5camera.captu
re_to_jpg())))
```

Set Camera Contrast



The set camera contrast block is used to control the camera's contrast. The range is from -2 to +2.



Contrast Set to -2.

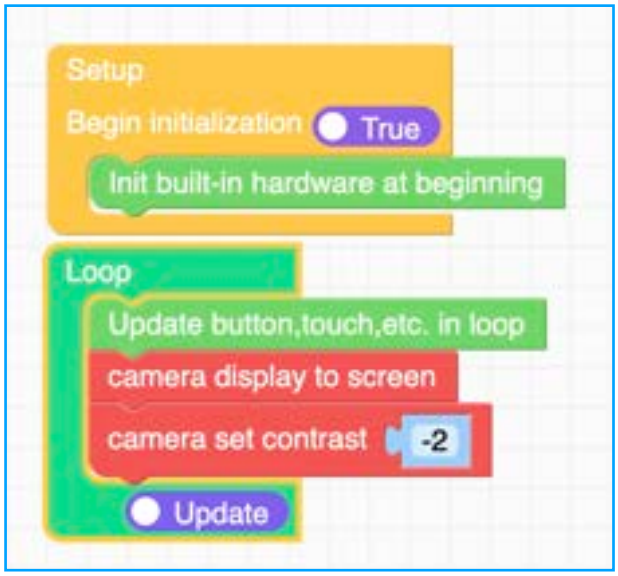
Contrast set to 0.

Contrast set to +2.

The Micropython code for the set camera contrast block is:

```
m5camera.contrast(0)
```

By changing the value in the “Set Contrast” block in the following example to can test contrast settings.



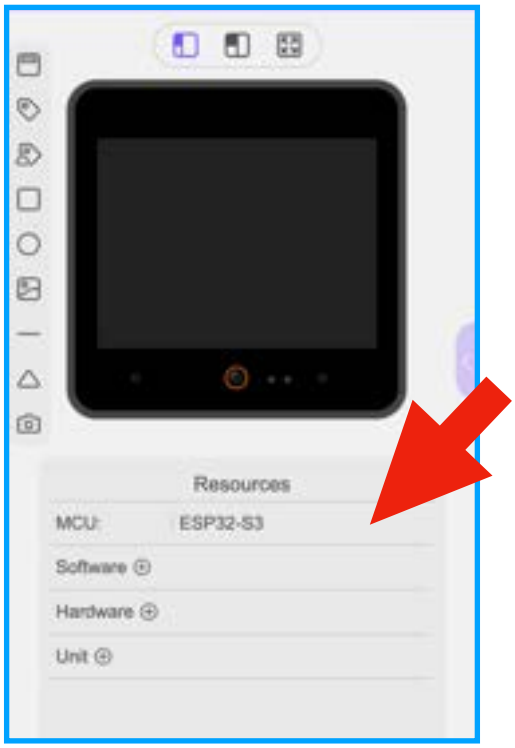
And the Micropython code for the example above is:

```
import os, sys, io
import M5
from M5 import *
import m5camera

def setup():

  M5.begin()
  Widgets.fillScreen(0x222222)
  m5camera.init(0, 0, 320, 240,
pixformat=m5camera.RGB565,
framesize=m5camera.FRAME_QVGA,
fb_count=2,
fb_location=m5camera.IN_PSRAM)

def loop():
  M5.update()
  m5camera.disp_to_screen()
  m5camera.contrast(-2)

if __name__ == '__main__':
  try:
    setup()
    while True:
      loop()
  except (Exception,
KeyboardInterrupt) as e:
    try:
      from utility import
print_error_msg
      print_error_msg(e)
    except ImportError:
      print("please update to
latest firmware")
```

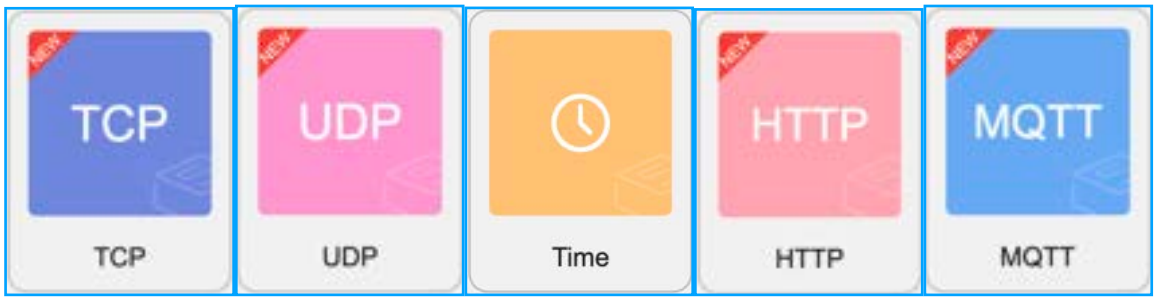
Software and Hardware Modules

Just like UIFlow 1.X, the software and hardware modules depend on what hardware controller is connected to UIFLow at the time of programming and the available functions can change.

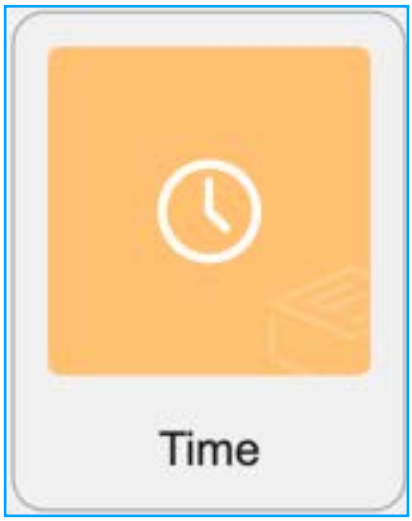
Unlike UIFlow1.XX, UIFlow2 moved the software and hardware specific modules to a selector under the GUI along with the Units.



While I'm using a CoreS3 here for the examples, I will be listing all available modules and functions available in UIFlow2.x.Software Modules



Time



The Time functions found in the section are used to control most time related actions from setting the Real Time clock to setting Sleep delays and measuring time periods that a function runs for.

Get Time Zone.

The Get Time Zone block returns the current timezone reading. If no time zone is set using one of the Set TimeZone blocks then Get Timezone will return “None”.

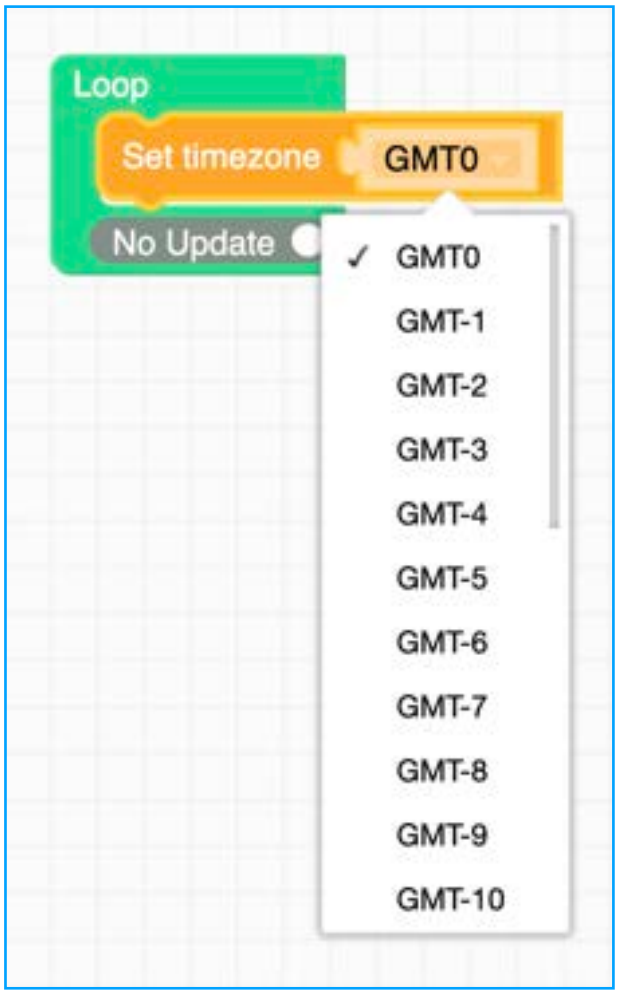
The Micropyhton API for this block is:

```
(time.timezone())
```

Set Time Zone

The two Set TimeZone Blocks perform the same function but In two different ways.

The first block is used to set the timezone using a dropdown box:



Where as the second is used to just type in a text string for the timezone.

The Micropyhton API for these two blocks is exactly the same:

```
time.timezone('GMT0')
```

It is worth noting that the Micropython API looks exactly the same as the Get Timezone function.

In the following example I set the Timezone using the dropdown box block and then use the Get timezone block to show the timezone I set.

The Micropython code for this example is as follows:

```
import os, sys, io
import M5
from M5 import *
import time

label0 = None
label1 = None


def setup():
  global label0, label1

  M5.begin()
  Widgets.fillScreen(0x222222)
  label0 = Widgets.Label("Text", 25,
46, 1.0, 0xffffff, 0x222222,
Widgets.FONTS.DejaVu18)
  label1 = Widgets.Label("Text", 24,
80, 1.0, 0xffffff, 0x222222,
Widgets.FONTS.DejaVu18)

def loop():
  global label0, label1
  M5.update()
  time.timezone('GMT0')

label0.setText(str((time.timezone())))

if __name__ == '__main__':
  try:
    setup()
    while True:
      loop()
  except (Exception,
KeyboardInterrupt) as e:
    try:
      from utility import
print_error_msg
      print_error_msg(e)
    except ImportError:
      print("please update to latest
firmware")
```

Sleep

The following blocks called Sleep serve the same purpose to delay a function being triggered after a previous function has been run. These blocks are also know as Wait or Delay blocks. It is worth noting that they are not Deep Sleep control functions.

The function has been separated into three different blocks in order to save the need for working out the difference is seconds, milliseconds and microseconds.

The Micropython api code for the sleep blocks is as follows:

```
time.sleep(1)
time.sleep_ms(1)
time.sleep_us(1)
```

us (lowercase) is used instead of µs for microseconds as the µ symbol is not universally available or not known how to find.

Get UTC Time

The Get UTC Time block returns the current date as a string of number in the following order:

- Year,
- Month,
- Day,
- Hour,
- Minute,
- Seconds

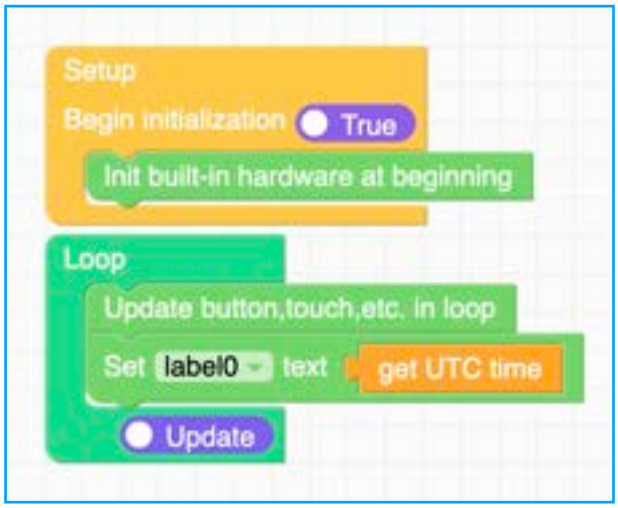
- Weekday
- Yearday.

Weekday is supposed to be 0 to 6 representing Monday to Sunday but it my tests it was working Sunday to Saturday.

The Micropython code for this block is:

```
time.gmtime()
```

As a value block it must be used with other blocks for example, in the following example I use it with a Label block to display a simple clock.



The Micropython code for this example is:

```
import os, sys, io
import M5
from M5 import *
import time

label0 = None

def setup():
  global label0

  M5.begin()
  Widgets.fillScreen(0x222222)
  label0 = Widgets.Label("Text",
8, 15, 1.0, 0xffffff, 0x222222,
Widgets.FONTS.DejaVu18)

def loop():
  global label0
  M5.update()

label0.setText(str((time.gmtime())
))

if __name__ == '__main__':
  try:
    setup()
    while True:
      loop()
  except (Exception,
KeyboardInterrupt) as e:
    try:
      from utility import
print_error_msg
      print_error_msg(e)
    except ImportError:
      print("please update to
latest firmware")
```

Get Local Timestamp Since January 1 1970

The Get Local Time Stamp Since Jan 1 1970 is a value function that returns the amount of milliseconds that have elapsed between 1 Jan 1970 and the current time and date the function was called. This function can not be used on its one and must be used with another block (like a Label block) in order to work.

The Micropython api for this block is:

```
(time.mktime(time.localtime()))
```

In the following example I just use a label block to display the millisecond requested by the block.

The Micropython code for this example is as follows:

```
import os, sys, io
import M5
from M5 import *
import time

label0 = None
```

```
def setup():
  global label0

  M5.begin()
  Widgets.fillScreen(0x222222)
  label0 = Widgets.Label("Text",
53, 66, 1.0, 0xffffff, 0x222222,
Widgets.FONTS.DejaVu18)

def loop():
  global label0
  M5.update()

label0.setText(str(time.mktime(tim
e.localtime())))

if __name__ == '__main__':
  try:
    setup()
    while True:
      loop()
  except (Exception,
KeyboardInterrupt) as e:
    try:
      from utility import
print_error_msg
      print_error_msg(e)
    except ImportError:
      print("please update to
latest firmware")
```

Get Local Time

The Get Local Time function is used to retrieve the local time from a web server that the device is located in. This is not always perfect and can sometimes return the local time for a different timezone.
The Get Local Time block returns the current date as a string of number in the following order:

- Year,
- Month,
- Day,
- Hour,
- Minute,
- Seconds
- Weekday
- Yearday.

Weekday is supposed to be 0 to 6 representing Monday to Sunday but it my tests it was working Sunday to Saturday.

As of the time of writing, it is UK summer time and so the following code example when run, show the time one hour fast:

In order to correct for this error a Set Timezone function is needed in the code to adjust the time zone.

The Micropython code for this example is as follows:

```
import os, sys, io
import M5
from M5 import *
import time

label0 = None

def setup():
  global label0

  time.timezone('GMT-1')
  M5.begin()
```

```
  Widgets.fillScreen(0x222222)
  label0 = Widgets.Label("Text",
6, 68, 1.0, 0xffffff, 0x222222,
Widgets.FONTS.DejaVu18)

def loop():
  global label0
  M5.update()

label0.setText(str(time.localtime(
)))

if __name__ == '__main__':
  try:
    setup()
    while True:
      loop()
  except (Exception,
KeyboardInterrupt) as e:
    try:
      from utility import
print_error_msg
      print_error_msg(e)
    except ImportError:
      print("please update to
latest firmware")
```

Get Time Stamp Since

The Get Time Stamp Since block allows users to specify a custom time and date to count the milliseconds from instead of the default date of the 1 Jan 1970.

The Micropython api for this function is:

```
(time.mktime((2000, 1, 1, 0, 0, 0,
0, 1)))
```

Counters and ticks

The next few blocks and functions are used for measuring time periods from one event to another. Ticks is an arbitrary value used for measuring time and can be set to a known time base like milliseconds, microseconds and/or CPU clock cycles. Ticks can also "Wrap Around" which means that when they reach a certain value, they can reset to 0 and start increasing again. Because of this, direct calculations can not be performed on tics values directly.

Get Ticks in Milliseconds

Returns ticks measured in milliseconds.

The Micropython api for this function is:

```
(time.ticks_ms())
```

Get Ticks in MicroSeconds

Returns ticks measured in microseconds.

The Micropython api for this function is:

```
time.ticks_us())
```

Get CPU Ticks Count

Returns the tics measured in CPU cycles.

The Micropython api for this function is:

```
(time.ticks_cpu())
```

Ticks Add delta

The Tics Add Delta block allows the ticks value to be modified by delta which can be the resultant number from a calculation.

The Micropython api for this function is:

```
(time.ticks_add(1, 1))
```

Ticks Difference

Returns the value of ticks between the two specified ticks values.

The Micropython api for this function is:

```
(time.ticks_diff(1, 1))
```

Get System Uptime.

Returns the internal value on how long a device has been powered on for.

The Micropython api for this function is:

```
(time.time())
```

MQTT

The MQTT functions in UIFlow 2.X allow for devices to use MQTT to communicate and transfer small amounts of data over a network. MQTT stands for Message Queue Telemetry Transport and is a Publish/Subscribe system where sending devices publish to a topic hosted on an MQTT server and receiving devices subscribe and retrieve data from the topic hosted on an MQTT server.

In Order to test and build an MQTT system for my home, I based my MQTT server on an M5Stack CM4 running a raspberry Pi CM4 with Raspbian using Mosquitto as the MQTT server https://mosquitto.org/ .

The hardware modules available to the CoreS3 are as follows:

Common Functions.

In this section I will list the common functions available in UIFlow2.x for all UIFlow2.x compatible controllers.

Variable.

In UIFlow, a variable is a symbolic name that represents a value stored in the memory of an M5Stack microcontroller. Variables in UIFlow, like in regular Micropython devices, are used to store data, and they can hold various types of values such as numbers, strings, lists, dictionaries, and more. Unlike some other programming languages, MicroPython is dynamically typed, meaning you don't need to declare the type of a variable explicitly; the type is determined based on the value it holds.
Here's an example of how you can define and use variables in MicroPython:

The Micropython code for this example is as follows:

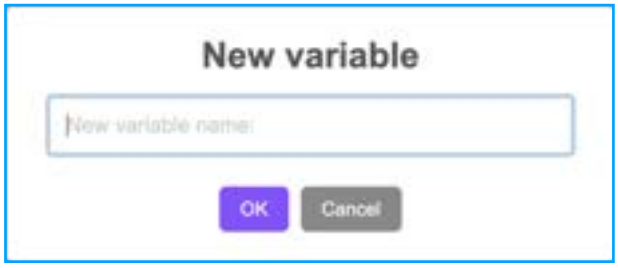
```
Age = None
Name = None
Temperature = None
Fruit = None

def loop():
  global i2c0, env3_0, Age, Name,
Temperature, Fruit
  M5.update()
  Age = 21
  Name = 'John Doe'
  Temperature =
env3_0.read_temperature()
  Fruit = ['Apple', 'Orange',
'Banana']
```

In UIFlow, we first create the variables with the value of “None” and then we populate the variable with initial values.

In the example above I have shown four different types of variable. The “Age” variable has an integer value, the “Name” has a string value, Temperature is a floating point number retrieved from a sensor (in this case the ENVIII) and the “Fruit” contains a list of strings.

To create a variable you click on the “Create Variable” button

And then in the popup window:



Type in a name for the variable. Once you click OK you will be returned to the main screen and three new variable blocks will appear:

Set Variable to

The Set Variable To block is used to set the initial value of the variable. In the above example you can see that I used to block to set the four different example types of initial value for the variable.

When the block is added to a program, three lines of Micropython code are created. The first Micropython line as mentioned previous creates the variable with the initial value of none:

```
Age = None
```

And then in the main loop, the variable is defined as a global variable:

```
global Age,
```

And then the variable is assigned its initial value:

```
Age = 21
```

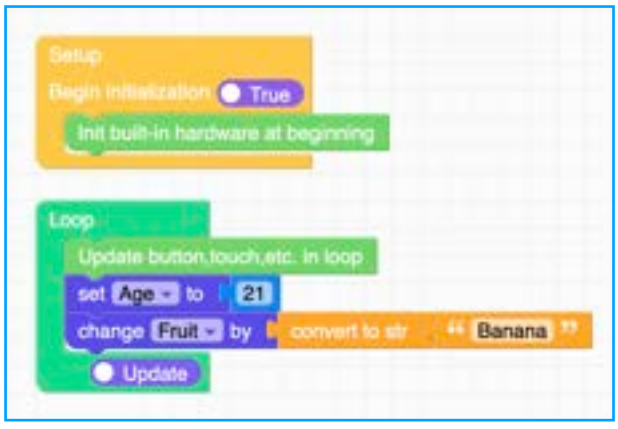
Change Variable.

The Change Variable By block is used to change the value of the variable. By default UIFlow assumes the variable you created with be a floating point value block assigns it a mathematical value of 1.

The Resulting Micropython API will look like this:

```
Fruit = (Fruit if
isinstance(Fruit, (int, float))
else 0) + 1
```

When using the change variable block for text, you must use a convert to str block as shown below:



Which results in the Micropython API for this function looking like this:

```
Fruit = (Fruit if
isinstance(Fruit, (int, float))
else 0) + (str('Banana'))
```

But if used just for integer numbers, then the Micropython API will look like this:

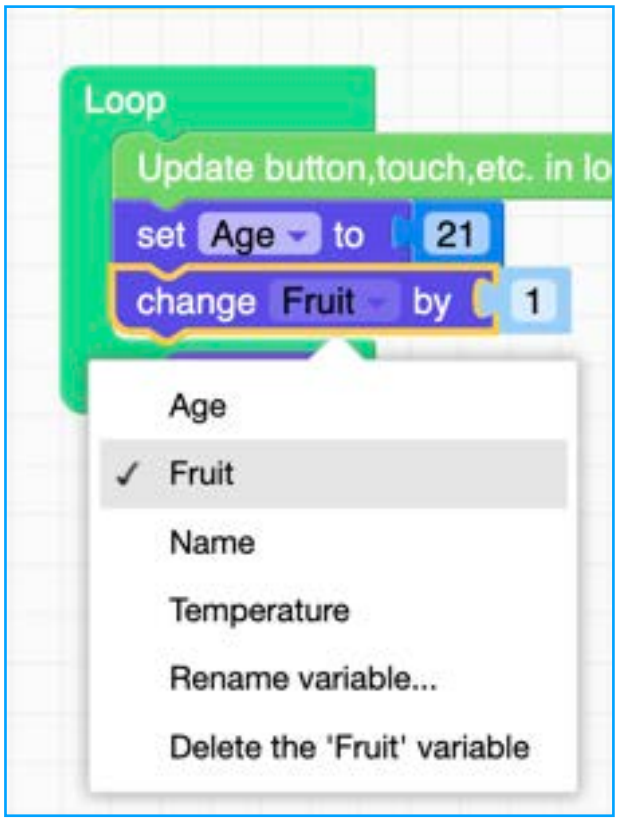
```
Age = 21
```

The Last block is the Variable value placeholder block:

Variable Value

The variable Value place holder block is used with code as a pointer allowing the code to request the current data stored in the variable.

Clicking on the down arrow next to the Variable name allows you to chose from a list of already defined variable, rename a variable (all instances of the renamed variable will also change) or delete a variable.



JSON

JSON (JavaScript Object Notation) in MicroPython refers to the implementation of JSON data serialisation and parsing within the MicroPython programming language. JSON is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is widely used for transmitting data between a server and a web application, as well as for configuration files and data storage.

In UIFlow There are two functions available for working with JSON. The first function we have is:

Dumps To JSON

The Dumps to JSON function is used to encrypt a data collection into a JSON formatted string.

The Micropython API for the Dumps to JSON block in UIFlow is:

```
json.dumps(_)
```

Data that needs to be converted into a JSON string can be formatted into two different data types using the Create MAP functions or the Create List Functions.

To two create a JSON MMAP string you use:

Or with the Create List you would use:

To load and convert the JSON string back into a list we use the Load JSON function.

Load JSON

The Micropython API for the Loads Json function is as follows:

```
json.loads(_)
```

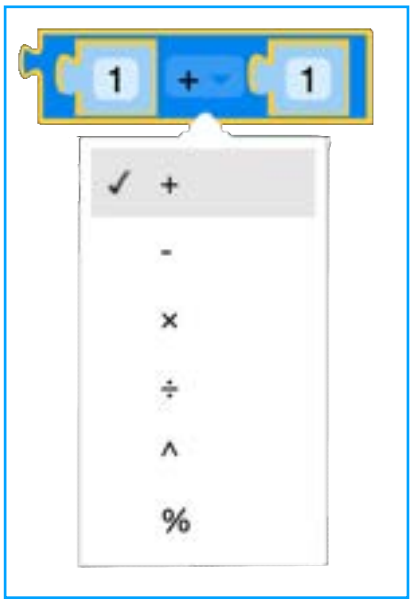
Mathematic Functions

Numbers

Used to hold an user defined integer or floating point number.

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. Clicking on the down arrow next to the “+” symbol reveals a drop down box where the following actions can be chosen.

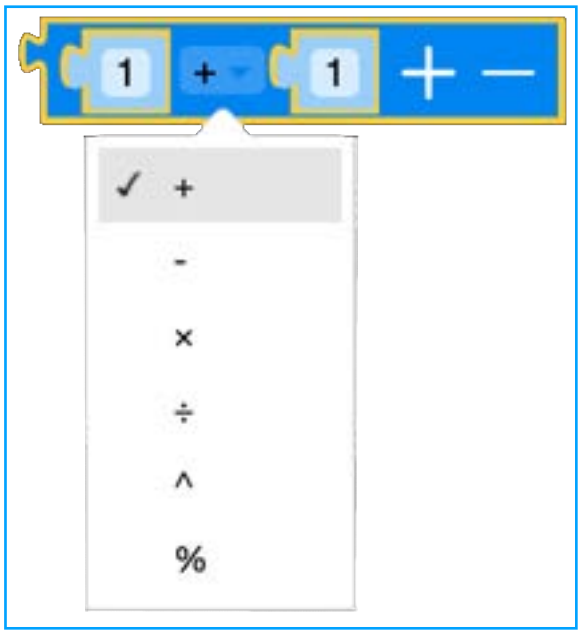


Available options are:

- Addition,
- Subtraction,
- Multiplication,
- Devision,
- Power,
- Percentage.

Formula calculation

The Formula function is used to add a sting of nested calculations together in a single line. Clicking on the down arrow next to the “+” symbol reveals a drop down box where the following actions can be chosen.

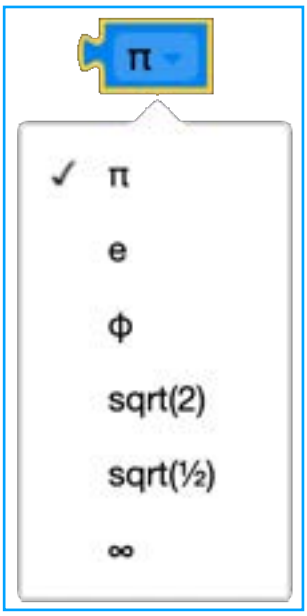


By Clicking on the “+” or the “-“ symbols, more sections can be added to the formula for example:

Which if place in a label block and run would should a result of 4.5 on the CoreS3’s screen.

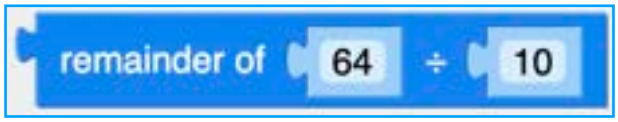
Mathematical Constants

Allows calculations to use special values.
By clicking on the dropdown box next to the symbol reveals the following special symbols:



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

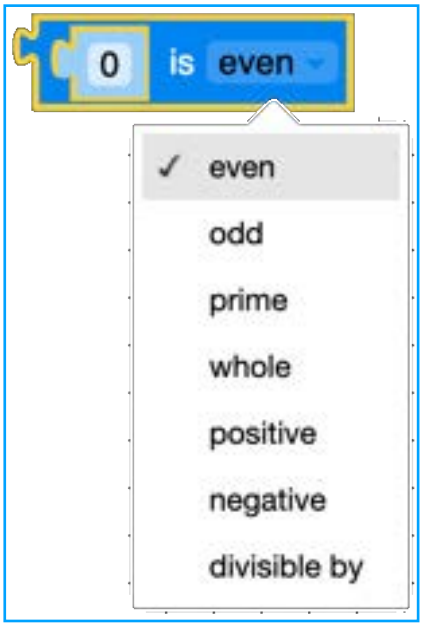
Remainder of



Returns the remainder of one value divided by another value.

Value is even

Used to decide when a value is Even, Odd, a Prime, a whole, a positive, a negative or divisible by dependent on what is set in the block.
By clicking the down arrow next to “even” you get a drop down list of options.



Values are

- Even,
- Odd,
- Prime,
- Whole,
- Positive,
- Negative,
- Divisible by

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 revealed by clicking the down arrow on the right.

To add a list, you would use the List function as shown below.

For more information on List functions, please visit the list section.

- Sum - The sum total of values stored 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.

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.

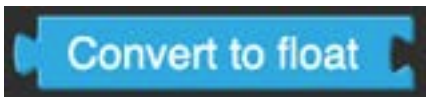
Trigonometric functions

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 footing point number.

Reserve Decimal Fraction

Reserves the user defined number as a decimal fraction.

S3 Schematics.

Full schematics for the Core S3 Develpopment Kit can be found in the document produced by M5 Stack found here: https://m5stack.oss-cn-shenzhen.aliyuncs.com/resource/docs/datasheet/core/K128 CoreS3/Sch_M5_CoreS3_v1.0.pdf

CoreS3/UIFlow Update Log

UIFlow Firmware

Alpha-21 2023-07-20

Feature: Unit GPS support.
Feature: update UI Image+.
Feature: micropython upgraded to 1.20.0.
Feature: Optimize core2 memory.

Alpha-20 2023-07-13

Feature: Core2 support.
Feature: Tough support.
Feature: UI Image+ support.
Feature: BLEUART supports setting name.

Alpha-19 2023-07-06

Feature: Software FileIO support.
Feature: Software WLAN STA support.
Feature: Software WLAN AP support.
Feature: Hardware SDCard support.
Feature: Hardware IR support.
Feature: Unit 8ENCODER support.
Feature: Unit LoRaWAN470 support.
Feature: Unit LoRaWAN868 support.
Feature: Unit LoRaWAN915 support.
Fixed: fix I2C bug.
Fixed: fix timezone bug, and auto save timezone to nvs.

Alpha-18 2023-06-29

Feature: Hardware WDT support.
Feature: Hardware RTC support.
Feature: Hardware I2S support.
Feature: Hardware SPI support.
Fixed: fix weekday error.

Alpha-17 2023-06-15

Feature: Hardware ALS(Ambient Light Sensor) support (CoreS3).
Feature: BtnPWR support.
Feature: Time add timezone support.
Feature: change epoch time from 2000-01-01 to 1970-01-01.
Feature: add APP.LIST.

Alpha-16 2023-06-09

Feature: UI Camera Image support.
Feature: support delete files remotely.

Alpha-15 2023-06-02

Feature: AtomS3U support.
Feature: Unit CardKB v1.1 support.
Fixed: fix widgets background colour bug.

Alpha-14 2023-05-26

Feature: Unit DLIGHT support.
Fixed: optimise the input effect of CardKB.
Fixed: fix CoreS3 touch lag.

Alpha-13 2023-05-19

Feature: CoreS3 support.
Feature: Hardware Touch support.

Alpha-12 2023-05-12

Feature: UI Label+ support.
Feature: Unit NCIR support.
Feature: Unit RELAY support.
Feature: Unit LIGHT support.

Alpha-11 2023-04-21

Feature: Software MQTT support.
Feature: Software HTTP support.
Feature: Software TCP support.
Feature: Software UDP support.
Feature: Unit Dual-BUTTON support.

Alpha-10 2023-04-14

Feature: Software BLE UART support.
Feature: Hardware custom Button support.

Alpha-9 2023-04-07

Feature: Unit EARTH support.
Feature: Unit ANGLE support.
Feature: Unit RGB support.
Feature: Unit FINGER support.
Feature: Unit PIR support.
Feature: Unit IR support.

Alpha-8 2023-03-31

Feature: Unit EXT.IO support.
Feature: Unit EXT.IO2 support.

Alpha-7 2023-03-24

Feature: Unit ADC support.
Feature: Unit DAC support.
Feature: Unit Color add method to get HSV colour space.

Fixed: setRotation bug fix.
Fixed: SK6812 colour order fix.

Alpha-6 2023-03-17

Feature: Unit ToF support.
Feature: Unit Color support.
Feature: Button callback function support.
Fixed: some bugs fix.

Alpha-5 2023-03-03

Feature: Unit PaHUB support.
Feature: Hardware Speaker support.
Feature: UserDisplay support.
Fixed: some bugs fix.

Alpha-4 2023-02-17

Feature: StampS3 support.
Feature: Unit ENV I/II/III support.
Fixed: some bugs fix.

Alpha-3 2023-02-10

Feature: download workspace code to device support.
Fixed: some bugs fix.

Alpha-2 2023-02-03

Feature: add AtomS3-Lite support.
Feature: Hardware IMU support.
Feature: Hardware RGB support.
Fixed: some bugs fix.

Alpha-1 2023-01-13

Feature: add startup UI support.
Feature: add new LCD API support.
Fixed: PNG bug fix.
Fixed: others bugs fix.

UIFlow WEB IDE

Alpha-21 2023-07-20

Feature: Unit GPS support.
Feature: update UI Image+.
Fixed: some bugs fix.

Alpha-20 2023-07-13

Feature: Core2 support.
Feature: Tough support.
Feature: UI Image+ support.
Feature: BLEUART supports setting name.
Fixed: some bugs fix.

Alpha-19 2023-07-06

Feature: Software FileIO support.
Feature: Software WLAN STA support.
Feature: Software WLAN AP support.
Feature: Hardware SDCard support.
Feature: Hardware IR support.
Feature: Unit 8ENCODER support.
Feature: Unit LoRaWAN470 support.
Feature: Unit LoRaWAN868 support.
Feature: Unit LoRaWAN915 support.
Fixed: some bugs fix.

Alpha-18 2023-06-29

Feature: Hardware WDT support.
Feature: Hardware RTC support.
Feature: Hardware I2S support.
Feature: Hardware SPI support.
Fixed: some bugs fix.

Alpha-17 2023-06-15

Feature: Hardware ALS(Ambient Light Sensor) support (CoreS3).
Feature: BtnPWR support.
Feature: Time add timezone support.
Fixed: some bugs fix.

Alpha-16 2023-06-09

Feature: UI Camera Image support.
Fixed: some bugs fix.Alpha-152023-06-02

Feature: AtomS3U support.
Feature: Unit CardKB v1.1 support.
Fixed: some bugs fix.

Alpha-14 2023-05-26

Feature: Unit DLIGHT support.
Fixed: some bugs fix.

Alpha-13 2023-05-19

Feature: CoreS3 support.
Feature: Hardware Touch support.
Fixed: some bugs fix.

Alpha-12 2023-05-12

Feature: UI Label+ support.
Feature: Unit NCIR support.
Feature: Unit RELAY support.
Feature: Unit LIGHT support.
Fixed: some bugs fix.

Alpha-11 2023-04-21

Feature: Software MQTT support.
Feature: Software HTTP support.
Feature: Software TCP support.
Feature: Software UDP support.
Feature: Unit Dual-BUTTON support.
Feature: Hardware Display support.
Fixed: some bugs fix.

Alpha-10 2023-04-14

Feature: Software BLE UART support.
Feature: Hardware custom Button support.
Fixed: some bugs fix.

Alpha-9 2023-04-07

Feature: Unit EARTH support.
Feature: Unit ANGLE support.
Feature: Unit RGB support.
Feature: Unit FINGER support.
Feature: Unit PIR support.
Feature: Unit IR support.
Fixed: some bugs fix.

Alpha-8 2023-03-31

Feature: Unit EXT.IO support.
Feature: Unit EXT.IO2 support.
Fixed: some bugs fix.

Alpha-7 2023-03-24

Feature: Unit ADC support.
Feature: Unit DAC support.
Feature: Button callback function support.
Fixed: some bugs fix.

Alpha-6 2023-03-17

Feature: Unit ToF support.
Feature: Unit Color support.
Fixed: some bugs fix.

Alpha-5 2023-03-03

Feature: Unit PaHUB support.
Feature: Hardware Speaker support.
Fixed: some bugs fix.

Alpha-4 2023-02-17

Feature: StampS3 support.
Feature: Unit ENV I/II/III support.
Feature: Hardware I2C support.
Fixed: some bugs fix.

Alpha-3 2023-02-10

Feature: download workspace code to device support.
Fixed: some bugs fix.

Alpha-2 2023-02-03

Feature: add AtomS3-Lite support.
Feature: Hardware IMU support.
Feature: Hardware RGB support.
Fixed: some bugs fix.

Alpha-1 2023-01-13

Feature: Hardware Button support.
Feature: Hardware UART support.
Feature: Software Time support.
Fixed: some bugs fix.

Exploring the Micropython Modules

Functions in Micropython are collected into classes which are then grouped into modules or libraries. In order to use functions you first have to import the module containing the class or directly import only the needed class from the container module.

Finding the internal modules requires issuing command into the **REPL (Read, Evaluate, Print and Loop)**. Looking at the bottom of Thonny at the moment the shell only has the following text:

```
MicroPython v1.19.1 on 2022-06-18; ESP32C3 module with ESP32C3
Type "help()" for more information.
>>>
```

After the 3 arrows (>>>) type in help() and hit return.

```
MicroPython v1.19.1 on 2022-06-18; ESP32C3 module with ESP32C3
Type "help()" for more information.
>>> help()
Welcome to MicroPython on the ESP32!

For generic online docs please visit http://docs.micropython.org/

For access to the hardware use the 'machine' module:

import machine
pin12 = machine.Pin(12, machine.Pin.OUT)
pin12.value(1)
pin13 = machine.Pin(13, machine.Pin.IN, machine.Pin.PULL_UP)
print(pin13.value())
i2c = machine.I2C(scl=machine.Pin(21), sda=machine.Pin(22))
i2c.scan()
i2c.writeto(addr, b'1234')
i2c.readfrom(addr, 4)

Basic WiFi configuration:

import network
sta_if = network.WLAN(network.STA_IF); sta_if.active(True)
sta_if.scan()                             # Scan for available access points
sta_if.connect("<AP_name>", "<password>") # Connect to an AP
sta_if.isconnected()                      # Check for successful connection

Control commands:
  CTRL-A        -- on a blank line, enter raw REPL mode
  CTRL-B        -- on a blank line, enter normal REPL mode
  CTRL-C        -- interrupt a running program
  CTRL-D        -- on a blank line, do a soft reset of the board
```

```
  CTRL-E       -- on a blank line, enter paste mode

For further help on a specific object, type help(obj)
For a list of available modules, type help('modules')
>>>
```

This give us a little bit of details to understand how the REPL/Shell works. The REPL/Shell is actually connected to the M5Stamp and directly running code on the physical M5Stamp.

In order to view the modules pre-compiled into the firmware type:

```
help('modules')
```

And hit return. Micropython is case sensitive so make sure the command is typed exactly as show.

```
>>> help('modules')
__main__        framebuf        uasyncio/stream     uplatform
_boot           gc              binascii            urandom
_onewire        inisetup        ubluetooth          ure
_thread         math            ucollections        uselect
_uasyncio       micropython     ucryptolib          usocket
_webrepl        neopixel        uctypes             ussl
apa106          network         uerrno              ustruct
btree           ntptime         uhashlib            usys
builtins        onewire         uheapq              utime
cmath           uarray          uio                 utimeq
dht             uasyncio/__init__   ujson           uwebsocket
ds18x20         uasyncio/core   umachine            uzlib
esp             uasyncio/event  uos                 webrepl
esp32           uasyncio/funcs  upip                webrepl_setup
flashbdev       uasyncio/lock   upip_utarfile       websocket_helper
Plus any modules on the filesystem
>>>
```

The list above shows us what modules are precompiled into the mainstream Micropython 1.19.1 firmware that I am using with the M5StampC3 and M5StampC3U. In order to view the functions in a module, you first need to import the module with:

```
>>> import esp32
```

And then use the dir() command to view the functions.

```
>>> dir(esp32)
['__class__', '__name__', 'HEAP_DATA', 'HEAP_EXEC', 'NVS', 'Partition', 'RMT',
'WAKEUP_ALL_LOW', 'WAKEUP_ANY_HIGH', 'gpio_deep_sleep_hold', 'idf_heap_info',
'wake_on_ext0', 'wake_on_ext1', 'wake_on_touch']
>>>
```

Most of that list is arguments which can be set in a function but more information on arguments will be covered later.

From the M5 module we have the following GUI API's:

Screen:

```
Widgets.setBrightness(0)
Widgets.fillScreen(0x6600cc)
Widgets.setRotation(0)
```

Title:

```
title0.setTextCursor(x=0)
title0.setColor(text_c=0x6600cc, bg_c=0x6600cc)
title0.setText('Title')
title0.setVisible(True)
title0.setVisible(False)
```

Label:

```
label0.setCursor(x=0, y=0)
label0.setColor(0x6600cc)
label0.setSize(1.5)
label0.setText(str('Hello World'))
label0.setFont(Widgets.FONTS.DejaVu9)
label0.setVisible(False)
label0.setVisible(True)
```

Label+

```
label_plus0.set_update_period(5000)
label_plus0.setCursor(x=0, y=0)
label_plus0.setColor(0x6600cc)
label_plus0.setSize(1.5)
label_plus0.setText(str('Label'))
label_plus0.setFont(Widgets.FONTS.DejaVu9)
label_plus0.setVisible(True)
label_plus0.setVisible(False)
label_plus0.is_valid_data()
```

Rectangle

```
rect0.setCursor(x=0, y=0)
rect0.setColor(color=0x6600cc, fill_c=0x6600cc)
rect0.setSize(w=0, h=0)
rect0.setVisible(True)
rect0.setVisible(False)
```

Circle

```
circle0.setCursor(x=0, y=0)
circle0.setColor(color=0x6600cc, fill_c=0x6600cc)
circle0.setRadius(r=0)
circle0.setVisible(True)
circle0.setVisible(False)
```

Line

```
line0.setPoints(x0=0, y0=0, x1=0, y1=0)
line0.setColor(0x6600cc)
line0.setVisible(True)
line0.setVisible(False)
```

Triangle

```
triangle0.setPoints(x0=0, y0=0, x1=0, y1=0, x2=0, y2=0)
triangle0.setColor(0x6600cc)
triangle0.setVisible(True)
triangle0.setVisible(False)
```

Image

```
image0.setCursor(x=0, y=0)
image0.setImage("res/img/default.png")
image0.setVisible(True)
```

Image+

```
image_plus0.set_update_enable(False)
image_plus0.set_update_period(5000)
image_plus0.setCursor(x=0, y=0)
image_plus0.setVisible(True)
```

Camera

```
m5camera.init(0, 0, 320, 240, pixformat=m5camera.RGB565,
framesize=m5camera.FRAME_QVGA, fb_count=2,
fb_location=m5camera.IN_PSRAM)
m5camera.setCursor(x=0, y=0, w=0, h=0)
m5camera.setVisible(True)
m5camera.setVisible(False)
m5camera.disp_to_screen()
m5camera.deinit()
m5camera.contrast(0)
m5camera.global_gain(0x12)
m5camera.hmirror(True)
m5camera.hmirror(False)
m5camera.vflip(True)
m5camera.vflip(False)
m5camera.colorbar(True)
m5camera.colorbar(False)
```

From the M5 module we have the following Software module API's:

Time

```
(time.timezone())
time.timezone('GMT0')
time.sleep(1)
time.sleep_ms(1)
time.sleep_us(1)
time.gmtime()
time.mktime(time.localtime()))
(time.mktime((2000, 1, 1, 0, 0, 0, 0, 1)))
(time.ticks_ms())
(time.ticks_us())
(time.ticks_cpu())
(time.ticks_add(1, 1))
(time.ticks_diff(1, 1))
(time.time())
```

BLE UART

MQTT