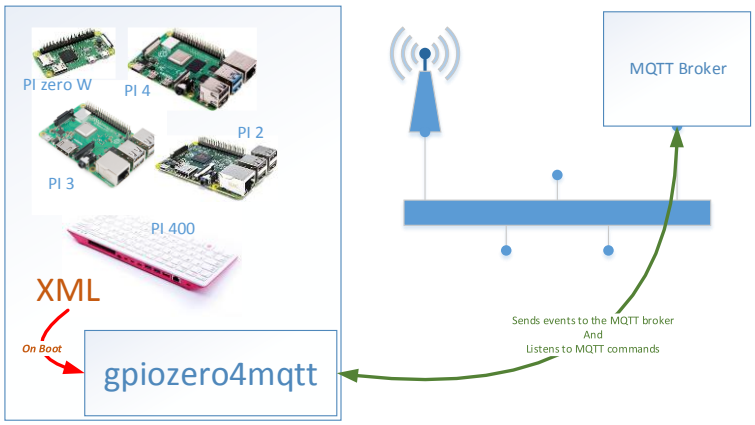
gpiozero4mqtt



Using XML to define and connect sensors and actuators, using gpiozero, to a MQTT broker on a local network.

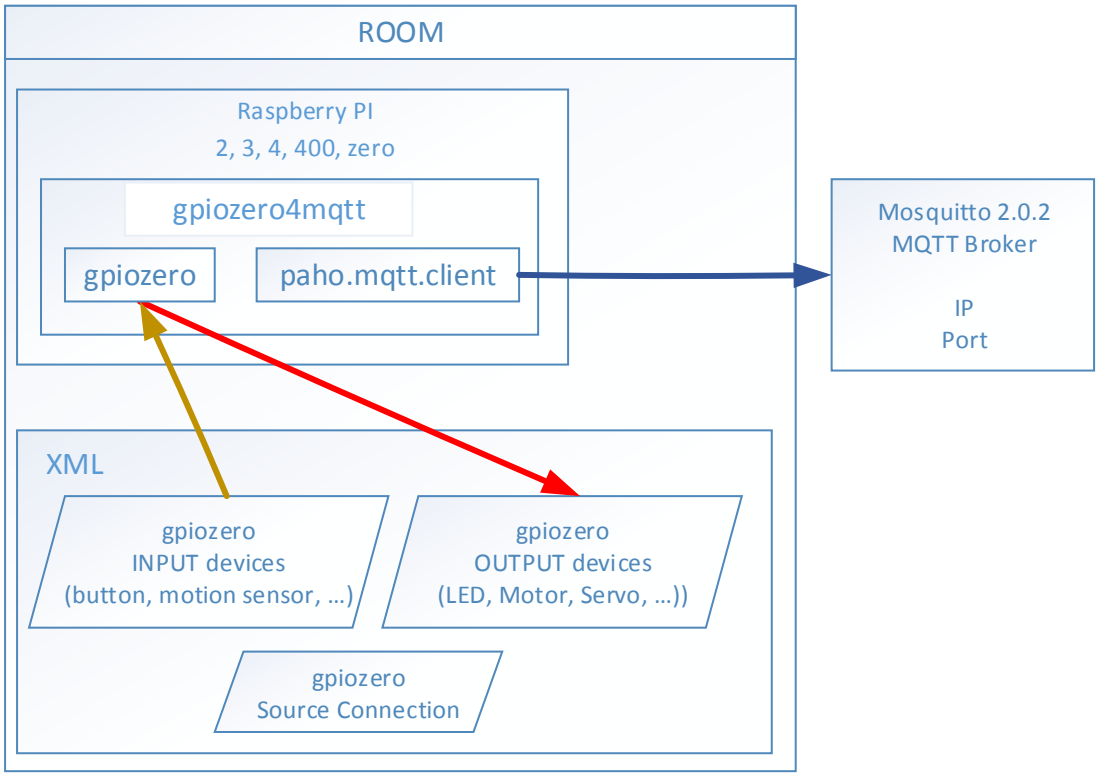
By using a simple XML file to define what is connected to a Raspberry PI (2,3,4,zero,400) you can connect all of the IN and OUT to a local MQTT broker.

By augmenting the gpiozero library with a MQTT layer that defines functions to listen to the MQTT broker or send information to it, we can completely automate the coding needed for the communication between the Raspberry pi and the local MQTT broker. Only configure the XML and run the framework … voila ! No coding !

The library gpiozero4mqtt.py is a Python layer added to the gpiozero library, used on many Raspberry Pi projects, to allow it to use an MQTT broker by only defining the resource and its connections in an XML file. It also allows usage of the source connection found in the gpiozero library.

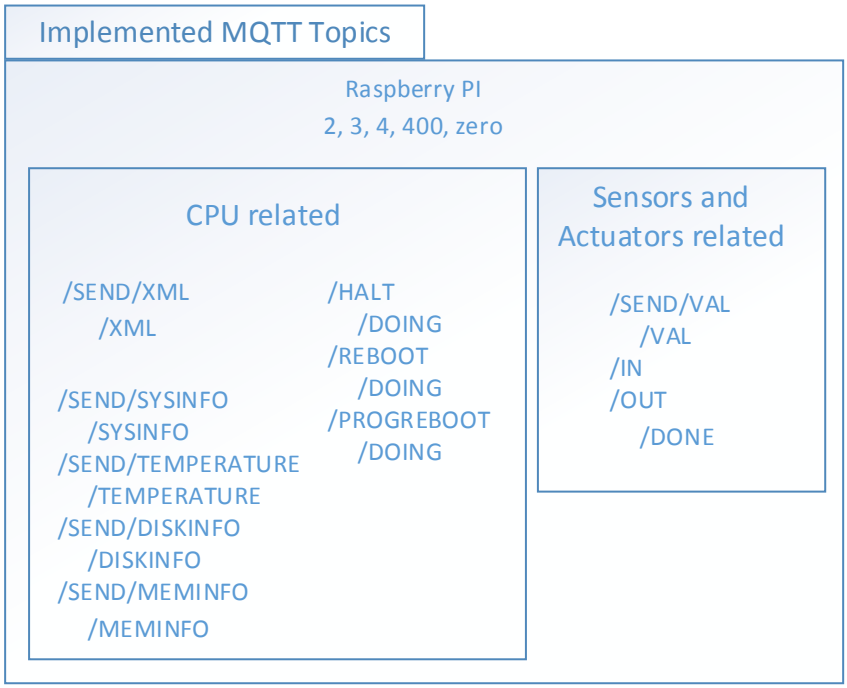
The concept implemented here is that a Raspberry Pi is usually associated to a room. There can be many Raspberry Pi in a room and there can be many rooms. Each Raspberry Pi has its own XML relating to what is connected to it and what MQTT broker to send information to or listen to for commands.

It has no intelligence about what to do with those messages. That is another problem altogether, which will be part of another document soon.



MQTT Topics

Here is a high-level view of the MQTT Topics implemented into gpiozero4mqtt. You can get the XML from the Raspberry and questions many aspect of the CPU like the temperature, the available disk space and memory, or the full blown sysinfo. You can also ask the Raspberry to either Halt, Reboot or program a reboot for later … Every sensor connected to the Raspberry and defined in the XML can respond to a query about its value (/SEND/VAL) and sends an /IN message for the Input devices, and responds to /OUT messages for every Output devices connected and defined in the XML.



Here is what the XML looks like for a simple Button and Led

```
<GPIO4MQTT name="Raspi2_1" room="Room1" logTo="./log/gpiozero4mqtt" atLevel="10" >
        <!-- MQTT broker connection information -->
        <MQTT host="10.148.82.21" port="1883" keepalive="60" qos="2"/>
        <!-- MQTT LED  -->
        <MQTT_LED name="Blue1" pin="5">
                <Static active_high="True" initial_value="False" pin_factory="None"/>                    <!-- Optional -->
        </MQTT_LED>
        <!-- MQTT Button  -->
        <MQTT_Button name="bigButton" pin="18" when_pressed="True" when_held="True" when_released="True">
                <Static pull_up="False" active_state="None" pin_factory="None"/>                         <!-- Optional -->
                <Time bounce_time="0.05" hold_time="0.25" hold_repeat="False"/>                          <!-- Optional -->
        </MQTT_Button>
</GPIO4MQTT>
```

The XML defines the mqtt connection host's address and the port to use as well as the quality of service. Then it defines a Led named Blue1 using pin 5 on the Raspberry PI. The Static tag is optional and is displayed to show what else can be set for a led. The same can be said for the Button.

With this XML, the button will generate three MQTT messages (Pressed, Held, Released) and respond to one /SEND/VAL. The LED will respond to 5 messages (On, Off, Toggle, Blink, /SEND/VAL).

The MQTT message has the following format in this library:

/ Direction / Room / Who / What

Messages to the cpu or widget	Messages from the cpu or widget
/HALT/Room_ID/ALL /HALT/Room_ID/cpu_ID /REBOOT/Room_ID/ALL /REBOOT/Room_ID/cpu_ID /PROGREBOOT/ACTIVATEM/Room_ID/ALL /PROGREBOOT/ACTIVATET/Room_ID/cpu_ID /PROGREBOOT/CANCEL/Room_ID/cpu_ID	/DOING/HALT/Room_ID/cpu_ID /DOING/REBOOT/Room_ID/cpu_ID /DOING/PROGREBOOT/MINUTES/Room_ID/cpu_ID /DOING/PROGREBOOT/TIME/Room_ID/cpu_ID /DOING/PROGREBOOT/CANCEL/Room_ID/cpu_ID
/SEND/XML/Room_ID/ALL /SEND/XML/Room_ID/cpu_ID	/XML/Room_ID/cpu_ID
/SEND/SYSINFO/Room_ID/cpu_ID /SEND/TEMPERATURE/Room_ID/cpu_ID /SEND/DISKINFO/Room_ID/cpu_ID ← a parfois ho /SEND/MEMINFO/Room_ID/cpu_ID	/SYSTATUS/SYSINFO/Room_ID/cpu_ID /SYSTATUS/TEMPERATURE/Room_ID/cpu_ID /SYSTATUS/DISKINFO/Room_ID/cpu_ID /SYSTATUS/MEMINFO/Room_ID/cpu_ID
/SEND/VAL/Room_ID/GPIOs[name]	/VAL/Room_ID/GPIOs[name]
/OUT/Room_ID/GPIOs[name]/What	/DONE/Room_ID/GPIOs[name]/What
	/IN/Room_ID/GPIOs[name]/What
/SEND/HELDTIME/Room_ID/GPIOs[name] /SEND/MOTIONDETECTED/Room_ID/GPIOs[name] /SEND/LIGHTDETECTED/Room_ID/GPIOs[name] /SEND/DISTANCE/Room_ID/GPIOs[name] /SEND/ACTIVETIME/Room_ID/GPIOs[name] /SEND/FREQUENCY/Room_ID/GPIOs[name] /SEND/COLOR/Room_ID/GPIOs[name]	/STATUS/HELDTIME/Room_ID/GPIOs[name] /STATUS/MOTIONDETECTED/Room_ID/GPIOs[name] /STATUS/LIGHTDETECTED/Room_ID/GPIOs[name] /STATUS/DISTANCE/Room_ID/GPIOs[name] /STATUS/ACTIVE_TIME/Room_ID/GPIOs[name] /STATUS/FREQUENCY/Room_ID/GPIOs[name] /STATUS/COLOR/Room_ID/GPIOs[name]
/SEND/ISHELD/Room_ID/GPIOs[name] /SEND/ISPRESSED/Room_ID/GPIOs[name] /SEND/ISLIT/Room_ID/GPIOs[name] /SEND/ISACTIVE/Room_ID/GPIOs[name] /SEND/ISINACTIVE/Room_ID/GPIOs[name]	/STATUS/ISHELD/Room_ID/GPIOs[name] /STATUS/ISPRESSED/Room_ID/GPIOs[name] /STATUS/ISLIT/Room_ID/GPIOs[name] /STATUS/ISACTIVE/Room_ID/GPIOs[name] /STATUS/ISINACTIVE/Room_ID/GPIOs[name]
/HEART/Room_ID/cpu_ID	/BEAT/Room_ID/cpu_ID

The *GPIOs[name]* can be any of the following which are found in the gpiozero library.

For INPUT

Button, LineSensor, MotionSensor, LightSensor, DistanceSensor, DigitalInputDevice

For OUTPUT

LED, RGBLED, PWMLed, Buzzer, TonalBuzzer, Motor, PhaseEnableMotor, Servo, AngularServo, DigitalOutputDevice, PWMOutputDevice , OutputDevice

It can also be anyone of the following, which I added to augment the accessible functionality.

For INPUT

Keypad, RFID, USB_GPS, MCP3008-PressurePlate, PiCamera

For OUTPUT

SoundMixer, Text2x16, ShiftRegister-7SegmentDisplay

For each type of Input or Output device, they either respond to messages (output) or generate messages (input). The next section lists all the messages for each type of devices.

mqtt_Button	PRESSED, HELD, RELEASED
mqtt_LineSensor	LINE, NO_LINE
mqtt_MotionSensor	MOTION, NO_MOTION
mqtt_LightSensor	LIGHT, DARK
mqtt_DistanceSensor	IN_RANGE, OUT_OF_RANGE
mqtt_DigitalInputDevice	ACTIVATED, DEACTIVATED

mqtt_Keypad	
mqtt_RFID	
mqtt_USB_GPS	
mqtt_MCP3008	

mqtt_LED	ON, OFF, TOGGLE, ***BLINK***
mqtt_RGBLED	ON, OFF, TOGGLE, ***BLINK***, ***PULSE***
mqtt_PWMLED	ON, OFF, TOGGLE, ***BLINK***, ***PULSE***
mqtt_Buzzer	ON, OFF, TOGGLE, ***BEEP***
mqtt_TonalBuzzer	***PLAY***, STOP
mqtt_Motor	***FORWARD***, ***BACKWARD***, REVERSE, STOP
mqtt_PhaseEnableMotor	***FORWARD***, ***BACKWARD***, REVERSE, STOP
mqtt_Servo	DETACH, MIN, MID, MAX, ***SetValue***
mqtt_AngularServo	***ANGLE***, MIN, MID, MAX
mqtt_DigitalOutputDevice	ON, OFF, ***BLINK***
mqtt_PWMOutputDevice	ON, OFF, TOGGLE, ***BLINK***, ***PULSE***
mqtt_OutputDevice	ON, OFF, TOGGLE

mqtt_SoundMixer	***PLAY***, STOP, PAUSE, UNPAUSE
mqtt_2x16_lcd_I2C	CLEAR, ***BACKLIT***, ***DISPLAYTHISATTHATLINE***, ***DISPLAYTHISATTHATPOSITION***, ***LCDLOADCUSTOMCHARS***
mqtt_ShiftRegiter	

All ***Output message in purple*** have dictionary to pass values to the functions.

It is our goal to augment the type of things that can be connected to the Raspberry Pi and that we can control over MQTT. It is an ongoing process !

MQTT PAYLOAD

The MQTT Payload is agnostic. That is why we can send the XML as is, and send the needed parameters and their values in the form most appropriate for the purpose. We use Python Dictionary since the library is written in Python and that json string can carry Python dictionary that can be used without any special code to make it so.

We therefore decided on the following Payload Dictionary structure.

```
PayloadDict = {
        “TimeStamp”: {},
        "Payload": {}
        }
```

Once defined, we use json to transfer everything to a string to use as the mqtt payload.

```
jsonPayload = json.dumps(PayloadDict)
```

The TimeStamp has the following structure :

```
“TimeStamp”: {
        "Y": 2020,
        "M": 12,
        "D":17,
        "H":21,
        "m":03,
        "s":0,
        "x":0
        }
```

The usual format for the “Payload”: value is another dictionary of the needed parameter and its value like so:

```
“Payload”: {“a_float_param”: 1.0, “a_int_param”: 1}
```

The receiving end on the message that uses the payload has to rebuild the Python dictionary before using it like follows.

```
jsonPayload = str(msg.payload.decode("utf-8", "ignore"))
PayloadDict = json.loads(jsonPayload)

a_float_param = float(PayloadDict["Payload"]["a_float_param"])
a_int_param = int(PayloadDict["Payload"]["a_int_param"])
```

mqtt_Button

You instantiate a mqtt_Button by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_Button name="bigButton" pin="18" when_pressed="True" when_held="True" when_released="True">
        <Static pull_up="False" active_state="None" pin_factory="None"/>          <!-- Optional -->
        <Time bounce_time="0.05" hold_time="0.25" hold_repeat="False"/>           <!-- Optional -->
</MQTT_Button>
```

This defines a mqtt_Button object with the name “bigButton” at pin 18. The when_pressed, when_held, and when_released are to get thoses messages or not, they are Boolean in nature.

The Static sub tag allows you to change the default way the button is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

The Time sub tag allows you to change the time related settings of the button like the bounce_time and the hold_time.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics published by this class:

/IN/Room_ID/GPIOs[name]/What

/IN/Room_ID/bigButton/PRESSED
/IN/Room_ID/bigButton/HELD
/IN/Room_ID/bigButton/RELEASED

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]

/SEND/VAL/Room_ID/bigButton

/VAL/Room_ID/GPIOs[name]

/VAL/Room_ID/bigButton { “Value”: value }

/SEND/HELDTIME/Room_ID/GPIOs[name]

/SEND/HELDTIME/Room_ID/bigButton

/STATUS/HELDTIME/Room_ID/GPIOs[name]

/STATUS/HELDTIME/Room_ID/bigButton {“held_time”: value }

/SEND/ISHELD/Room_ID/GPIOs[name]

/SEND/ISHELD/Room_ID/bigButton

/STATUS/ISHELD/Room_ID/GPIOs[name]

/STATUS/ISHELD/Room_ID/bigButton {“is_held”: value }

/SEND/ISPRESSED/Room_ID/GPIOs[name]

/SEND/ISPRESSED/Room_ID/bigButton

/STATUS/ISPRESSED/Room_ID/GPIOs[name]

/STATUS/ISPRESSED/Room_ID/bigButton {“is_pressed”: value }

mqtt_LineSensor

You instantiate a mqtt_LineSensor by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_LineSensor name="LineSensor1" pin="12" when_line="True" when_no_line="True">
        <Static pull_up="False" active_state="None" pin_factory="None"/>          <!-- Optional -->
        <Queue queue_len="5" sample_rate="100" threshold="0.5" partial="False"/>  <!-- Optional -->
</MQTT_LineSensor>
```

This defines a mqtt_LineSensor object with the name “LineSensor1” at pin 12. The when_line, and when_no_line are to get thoses messages or not, they are Boolean in nature.

The Static sub tag allows you to change the default way the LineSensor is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

The Queue tag allows to set many attributes of the sensor as the length of the queue used to store values read from the sensor, the default is 5, the sample_rate (100 per second) and the threshold (0.5) to consider the average as “active”.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics published by this class:

/IN/Room_ID/GPIOs[name]/What

/IN/Room_ID/LineSensor1/LINE
/IN/Room_ID/LineSensor1/NO_LINE

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***	
/SEND/VAL/Room_ID/LineSensor1	/VAL/Room_ID/LineSensor1	{ “Value”: value }

mqtt_MotionSensor

You instantiate a mqtt_MotionSensor by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_MotionSensor name="PIR1" pin="12" when_motion="True" when_no_motion="True">
        <Static pull_up="False" active_state="None" pin_factory="None"/>          <!-- Optional -->
        <Queue queue_len="1" sample_rate="10" threshold="0.5" partial="False"/>   <!-- Optional -->
</MQTT_MotionSensor>
```

This defines an mqtt_MotionSensor object with the name “PIR1” at pin 12. The when_motion, and when_no_motion are to get thoses messages or not, they are Boolean in nature.

The Static sub tag allows you to change the default way the MotionSensor is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

The Queue tag allows to set many attributes of the sensor as the length of the queue used to store values read from the sensor, the default is 1, the sample_rate (10 per second) and the threshold (0.5) to consider the average as “active”.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics published by this class:

/IN/Room_ID/GPIOs[name]/What

/IN/Room_ID/PIR1/MOTION
/IN/Room_ID/PIR1/NO_MOTION

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***	
/SEND/VAL/Room_ID/PIR1	/VAL/Room_ID/PIR1	{ “Value”: value }

/SEND/MOTIONDETECTED/Room_ID/GPIOs[name]	***/STATUS/MOTIONDETECTED/Room_ID/GPIOs[name]***	
/SEND/MOTIONDETECTED/Room_ID/PIR1	/STATUS/MOTIONDETECTED/Room_ID/PIR1	{“motion_detected”: value}

mqtt_LightSensor

You instantiate a mqtt_LightSensor by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_LightSensor name="Ls1" pin="12" when_light="True" when_dark="True">
        <Static pull_up="False"" active_state="None" pin_factory="None"/>          <!-- Optional -->
        <Queue queue_len="1" charge_time_limit="0.01" threshold="0.1" partial="False"/>   <!-- Optional -->
</MQTT_LightSensor>
```

This defines a mqtt_LightSensor object with the name “LightSensor1” at pin 12. The when_light, and when_dark are to get thoses messages or not, they are Boolean in nature.

The Static sub tag allows you to change the default way the LightSensor is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

The Queue tag allows to set many attributes of the sensor as the length of the queue used to store values read from the sensor, the default is 1, the charge_time_limit(0.01) and the threshold (0.1) to consider the average as “active”.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics published by this class:

/IN/Room_ID/GPIOs[name]/What

/IN/Room_ID/Ls1/LIGHT
/IN/Room_ID/Ls1/DARK

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name] ***/VAL/Room_ID/GPIOs[name]***

/SEND/VAL/Room_ID/Ls1 /VAL/Room_ID/LightSensor1 { “Value”: value }

/SEND/LIGHTDETECTED/Room_ID/GPIOs[name] ***/STATUS/LIGHTDETECTED/Room_ID/GPIOs[name]***

/SEND/LIGHTDETECTED/Room_ID/Ls1 /STATUS/LIGHTDETECTED/Room_ID/Ls1 {“light_detected”: value}

mqtt_DistanceSensor

You instantiate a mqtt_DistanceSensor by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_DistanceSensor name="DistanceSensor1" echo="12" trigger="6" when_in_range="True" when_out_of_range="True" send_every="None">
        <Static pull_up="False" active_state="None" pin_factory="None"/>                    <!-- Optional -->
        <Queue queue_len="30" max_distance="1" threshold_distance="0.3" partial="False" />  <!-- Optional -->
</MQTT_DistanceSensor>
```

This defines a mqtt_DistanceSensor object with the name “DistanceSensor1” at pin 6 (trigger) and 12 (echo). The when_in_range, and when_out_of_range are to get thoses messages or not, they are Boolean in nature.

You can get the distance at every x seconds using the “send_every” parameter. It is the same as asking the distance using the /SEND/DISTANCE message at every x seconds. The code to manage that aspect is in the loop_process function.

The Static sub tag allows you to change the default way the DistanceSensor is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

The Queue tag allows to set many attributes of the sensor as the length of the queue used to store values read from the sensor, the default is 30, the max_distance (1) and the threshold_distance (0.3) to consider the average as “active”.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics published by this class:

/IN/Room_ID/GPIOs[name]/What

/IN/Room_ID/DistanceSensor1/IN_RANGE
/IN/Room_ID/DistanceSensor1/OUT_OF_RANGE

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***	
/SEND/VAL/Room_ID/DistanceSensor1	/VAL/Room_ID/DistanceSensor1	{ “Value”: value }

/SEND/DISTANCE/Room_ID/GPIOs[name]	***/STATUS/DISTANCE/Room_ID/GPIOs[name]***	
/SEND/DISTANCE/Room_ID/DistanceSensor1	/STATUS/DISTANCE/Room_ID/DistanceSensor1	{“distance”: value}

mqtt_DigitalInputDevice

You instantiate a mqtt_DigitalInputDevice by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_DigitalInputDevice name="selector1" pin="22" when_activated="True" when_deactivated="True" bounce_time="0.02">
        <Static pull_up="True" active_state="None" pin_factory="None"/>                    <!-- Optional -->
</MQTT_DigitalInputDevice>
```

This defines a mqtt_DigitalInputDevice object with the name “Selector1” at pin 22. The when_activated, and when_deactivated are to get thoses messages or not, they are Boolean in nature.

The Static sub tag allows you to change the default way the DistanceSensor is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics published by this class:

/IN/Room_ID/GPIOs[name]/What

/IN/Room_ID/Selector1/ACTIVATED
/IN/Room_ID/Selector1/DEACTIVATED

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name] ***/VAL/Room_ID/GPIOs[name]***

/SEND/VAL/Room_ID/Selector1 /VAL/Room_ID/Selector1 { “Value”: value }

/SEND/ACTIVETIME/Room_ID/GPIOs[name] ***/STATUS/ACTIVETIME/Room_ID/GPIOs[name]***

/SEND/ACTIVETIME/Room_ID/Selector1 /STATUS/ACTIVETIME/Room_ID/Selector1 { “active_time”: value}

/SEND/INACTIVETIME/Room_ID/GPIOs[name] ***/STATUS/INACTIVETIME/Room_ID/GPIOs[name]***

/SEND/INACTIVETIME/Room_ID/Selector1 /STATUS/INACTIVETIME/Room_ID/Selector1 { “inactive_time”: value}

mqtt_LED

You instantiate a mqtt_LED by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_LED name="Blue1" pin="5">
        <Static active_high="True" initial_value="False" pin_factory="None"/>          <!-- Optional -->
</MQTT_LED>
```

This defines a mqtt_LED object with the name “Blue1” at pin 5.

The Static sub tag allows you to change the default way the LED is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/Blue1/ON
/OUT/Room_ID/Blue1/OFF
/OUT/Room_ID/Blue1/TOGGLE
/OUT/Room_ID/Blue1/***BLINK*** ***Blink Dictionary***

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name] ***/VAL/Room_ID/GPIOs[name]***

/SEND/VAL/Room_ID/Blue1 /VAL/Room_ID/Blue1 { “Value”: value }

/SEND/ISLIT/Room_ID/GPIOs[name] ***/STATUS/ISLIT/Room_ID/GPIOs[name]***

/SEND/ISLIT/Room_ID/Blue1 /STATUS/ISLIT/Room_ID/Blue1 {“is_lit”: value}

The ***Blink Dictionary*** has the following structure:

```
“Payload”: {
        “on_time”: 1.0,
        “off_time”: 1.0,
        “n”: None,
        “background”: False
        }
```

mqtt_PWMLED

You instantiate a mqtt_PWMLED by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_PWMLED name="PWMLED1" pin="12">
        <Static active_high="True" initial_value="0" frequency="100" pin_factory="None"/>        <!-- Optional -->
</MQTT_PWMLED>
```

This defines a mqtt_PWMLED object with the name “PWMLED1” at pin 5.

The Static sub tag allows you to change the default way the PWMLED is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/PWMLED1/ON
/OUT/Room_ID/PWMLED1/OFF
/OUT/Room_ID/PWMLED1/TOGGLE
/OUT/Room_ID/PWMLED1/***BLINK*** ***PWM Blink Dictionary***
/OUT/Room_ID/PWMLED1/***PULSE*** ***Pulse Dictionary***

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name] ***/VAL/Room_ID/GPIOs[name]***

/SEND/VAL/Room_ID/PWMLED1 /VAL/Room_ID/PWMLED1 { “Value”: value }

/SEND/ISLIT/Room_ID/GPIOs[name] ***/STATUS/ISLIT/Room_ID/GPIOs[name]***

/SEND/ISLIT/Room_ID/PWMLED1 /STATUS/ISLIT/Room_ID/PWMLED1 {“is_lit”: value}

The ***PWM Blink Dictionary*** has the following structure:

```
“Payload”: {
        “on_time”: 1.0,
        “off_time”: 1.0,
        “fade_in_time”: 0,
        “fade_out_time”: 0,
        “n”: None,
        “background”: False
        }
```

The ***Pulse Dictionary*** has the following structure:

```
“Payload”: {
        “fade_in_time”: 1.0,
        “fade_out_time”: 1.0,
        “n”: None,
        “background”: False
        }
```

mqtt_RGBLED

You instantiate a mqtt_RGBLED by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_RGBLED name="RGB1" r="16" g="20" b="21">
        <Static active_high="True" initial_value="(0, 0, 0)" pwm="True" pin_factory="None"/>        <!-- Optional -->
</MQTT_RGBLED>
```

This defines a mqtt_RGBLED object with the name “RGBLED1” at pin R=16, G=20, B=21.

The Static sub tag allows you to change the default way the RGBLED is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/RGBLED1/ON	
/OUT/Room_ID/RGBLED1/OFF	
/OUT/Room_ID/RGBLED1/TOGGLE	
/OUT/Room_ID/RGBLED1/***BLINK***	***RGB Blink Dictionary***
/OUT/Room_ID/RGBLED1/***PULSE***	***Pulse Dictionary***
/OUT/Room_ID/RGBLED1/***COLOR***	***Color Dictionary***

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***	
/SEND/VAL/Room_ID/RGBLED1	/VAL/Room_ID/RGBLED1	{ “Value”: value }

/SEND/ISLIT/Room_ID/GPIOs[name]	***/STATUS/ISLIT/Room_ID/GPIOs[name]***	
/SEND/ISLIT/Room_ID/RGBLED1	/STATUS/ISLIT/Room_ID/RGBLED1	{“is_lit”: value}

The ***RGB Blink Dictionary*** has the following structure:

```
“Payload”: {
        “on_time”: 1.0,
        “off_time”: 1.0,
        “fade_in_time”: 0,
        “fade_out_time”: 0,
        “on_color”: (1, 1, 1),
        “off_color”: (0, 0, 0),
        “n”: None,
        “background”: False
        }
```

The ***RGB Pulse Dictionary*** has the following structure:

```
“Payload”: {
        “fade_in_time”: 1.0,
        “fade_out_time”: 1.0,
        “on_color”: (1, 1, 1),
        “off_color”: (0, 0, 0),
        “n”: None,
        “background”: False
        }
```

The ***COLOR Dictionary*** has the following structure:

```
“Payload”: {
        “color”: (1.0, 1.0, 1.0)
        }
```

mqtt_Buzzer

You instantiate a mqtt_Buzzer by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_Buzzer name="Buzzer1" pin="12">
        <Static active_high="True" initial_value="False" pin_factory="None"/>          <!-- Optional -->
</MQTT_Buzzer>
```

This defines a mqtt_Buzzer object with the name “Buzzer1” at pin 12.

The Static sub tag allows you to change the default way the Buzzer is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/Buzzer1/ON
/OUT/Room_ID/Buzzer1/OFF
/OUT/Room_ID/Buzzer1/TOGGLE
/OUT/Room_ID/Buzzer1/***BEEP*** ***Beep Dictionary***

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***	
/SEND/VAL/Room_ID/Buzzer1	/VAL/Room_ID/Buzzer1	{ “Value”: value }

/SEND/ISACTIVE/Room_ID/GPIOs[name]	***/STATUS/ISACTIVE/Room_ID/GPIOs[name]***	
/SEND/ISACTIVE/Room_ID/Buzzer1	/STATUS/ISACTIVE/Room_ID/Buzzer1	{“is_active”: value}

The ***Beep Dictionary*** has the following structure:

```
“Payload”: {
        “on_time”: 1.0,
        “off_time”: 1.0,
        “n”: None,
        “background”: False
        }
```

mqtt_TonalBuzzer

You instantiate a mqtt_TonalBuzzer by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_TonalBuzzer name="TonalBuzzer1" pin="12">
        <ToneSpec initial_value="None" mid_tone="A4" octaves="1"/>          <!-- Optional -->
        <Static pin_factory="None"/>                                        <!-- Optional -->
</MQTT_TonalBuzzer>
```

This defines a mqtt_TonalBuzzer object with the name “TonalBuzzer1” at pin 12.

The ToneSpec tag allows to set initial_value, the mid_tone and the octaves.

The Static sub tag allows you to change the default way the TonalBuzzer is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/TonalBuzzer1/***PLAY*** ***Play Dictionary***
/OUT/Room_ID/TonalBuzzer1/STOP

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***	
/SEND/VAL/Room_ID/TonalBuzzer1	/VAL/Room_ID/TonalBuzzer1	{ “Value”: value }

/SEND/ISACTIVE/Room_ID/GPIOs[name]	***/STATUS/ISACTIVE/Room_ID/GPIOs[name]***	
/SEND/ISACTIVE/Room_ID/TonalBuzzer1	/STATUS/ISACTIVE/Room_ID/TonalBuzzer1	{“is_active”: value}

The ***Play Dictionary*** has the following structure:

```
“Payload”: {
        “tone”: “A4”
        }
```

mqtt_Motor

You instantiate a mqtt_Motor by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_Motor name="Motor1" forward="12" backward="6" enable="None" pwm="True">
        <Static active_high="True" pin_factory="None"/> <!-- Optional -->
</MQTT_Motor>
```

This defines a mqtt_Motor object with the name “Motor1” at pin forward=12, backward=6, using pwm.

The Static sub tag allows you to change the default way the Motor is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/Motor1/***FORWARD***	***Speed Dictionary***
/OUT/Room_ID/Motor1/***BACKWARD***	***Speed Dictionary***
/OUT/Room_ID/Motor1/REVERSE	
/OUT/Room_ID/Motor1/STOP	

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***	
/SEND/VAL/Room_ID/Motor1	/VAL/Room_ID/Motor1	{ “Value”: value }

/SEND/ISACTIVE/Room_ID/GPIOs[name]	***/STATUS/ISACTIVE/Room_ID/GPIOs[name]***	
/SEND/ISACTIVE/Room_ID/Motor1	/STATUS/ISACTIVE/Room_ID/Motor1	{“is_active”: value}

The ***Speed Dictionary*** has the following structure:

```
“Payload”: {
        “speed”: 0.80
        }
```

mqtt_PhaseEnableMotor

You instantiate a mqtt_PhaseEnableMotor by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_PhaseEnableMotor name="PhaseEnableMotor1" phase="12" enable="13" pwm="True">
        <Static active_high="True" pin_factory="None"/>                    <!-- Optional -->
</MQTT_PhaseEnableMotor>
```

This defines a mqtt_PhaseEnableMotor object with the name PhaseEnableMotor1" at pin phase=12, enable=13, using pwm.

The Static sub tag allows you to change the default way the PhaseEnableMotor is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/PhaseEnableMotor1/***FORWARD***	***Speed Dictionary***
/OUT/Room_ID/PhaseEnableMotor1/***BACKWARD***	***Speed Dictionary***
/OUT/Room_ID/PhaseEnableMotor1/REVERSE	
/OUT/Room_ID/PhaseEnableMotor1/STOP	

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***	
/SEND/VAL/Room_ID/PhaseEnableMotor1	/VAL/Room_ID/PhaseEnableMotor1	{ "Value": value }

/SEND/ISACTIVE/Room_ID/GPIOs[name]	***/STATUS/ISACTIVE/Room_ID/GPIOs[name]***	
/SEND/ISACTIVE/Room_ID/PhaseEnableMotor1	/STATUS/ISACTIVE/Room_ID/PhaseEnableMotor1	{"is_active": value}

The ***Speed Dictionary*** has the following structure:

```
"Payload": {
        "speed": 0.80
        }
```

mqtt_Servo

You instantiate a mqtt_Servo by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_Servo name="Servo1" pin="12" initial_value="0" min_pulse_width="1/1000" max_pulse_width="2/1000" frame_width="20/1000">
        <Static active_high="True" pin_factory="None"/>                                    <!-- Optional -->
</MQTT_Servo>
```

This defines a mqtt_Servo object with the name “Servo1” at pin 12 with an initial_value of zero, a min_pulse_width to 1 thousandth ms, a max_pulse_width of 2 thousandth ms and a frame_width of 20 thousandth ms.

The Static sub tag allows you to change the default way the Servo is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/Servo1/DETACH
/OUT/Room_ID/Servo1/MIN
/OUT/Room_ID/Servo1/MID
/OUT/Room_ID/Servo1/MAX
/OUT/Room_ID/Servo1/***SetValue*** ***Value Dictionary***

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***	
/SEND/VAL/Room_ID/Servo1	/VAL/Room_ID/Servo1	{ “Value”: value }

/SEND/ISACTIVE/Room_ID/GPIOs[name]	***/STATUS/ISACTIVE/Room_ID/GPIOs[name]***	
/SEND/ISACTIVE/Room_ID/Servo1	/STATUS/ISACTIVE/Room_ID/Servo1	{“is_active”: value}

The ***Value Dictionary*** has the following structure:

```
“Payload”: {
        “value”: 0.80          between -1 and 1
        }
```

mqtt_AngularServo

You instantiate a mqtt_AngularServo by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_AngularServo name="AngularServo1" pin="12" initial_angle="0" min_angle="-90" max_angle="90" min_pulse_width="1/1000"
                                                                    max_pulse_width="2/1000" frame_width="20/1000">
        <Static active_high="True" pin_factory="None"/>                                                    <!-- Optional -->
</MQTT_AngularServo>
```

This defines a mqtt_AngularServo object with the name “AngularServo1” at pin 12 with an initial_angle of 0, a min_angle of -90, a max_angle of 90, a min_pulse_width to 1 thousandth ms, a max_pulse_width of 2 thousandth ms and a frame_width of 20 thousandth ms.

The Static sub tag allows you to change the default way the AngularServo is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/AngularServo1/MIN
/OUT/Room_ID/AngularServo1/MID
/OUT/Room_ID/AngularServo1/MAX
/OUT/Room_ID/AngularServo1/***ANGLE*** ***Value Dictionary***

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***	
/SEND/VAL/Room_ID/AngularServo1	/VAL/Room_ID/AngularServo1	{ “Value”: value }

/SEND/ISACTIVE/Room_ID/GPIOs[name]	***/STATUS/ISACTIVE/Room_ID/GPIOs[name]***	
/SEND/ISACTIVE/Room_ID/AngularServo1	/STATUS/ISACTIVE/Room_ID/AngularServo1	{“is_active”: value}

The ***Value Dictionary*** has the following structure:

```
“Payload”: {
        “value”: 45          between min_angle and max_angle
        }
```

mqtt_DigitalOutputDevice

You instantiate a mqtt_DigitalOutputDevice by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_DigitalOutputDevice name="DigitalOutputDevice1" pin="12">
        <Static active_high="True" initial_value="False" pin_factory="None"/> <!-- Optional -->
</MQTT_DigitalOutputDevice>
```

This defines a mqtt_DigitalOutputDevice object with the name “DigitalOutputDevice1” at pin 12.

The Static sub tag allows you to change the default way the DigitalOutputDevice is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/DigitalOutputDevice1/ON
/OUT/Room_ID/DigitalOutputDevice1/OFF
/OUT/Room_ID/DigitalOutputDevice1/***BLINK*** ***Blink Dictionary***

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name] ***/VAL/Room_ID/GPIOs[name]***

/SEND/VAL/Room_ID/DigitalOutputDevice1 /VAL/Room_ID/DigitalOutputDevice1 { “Value”: value }

The ***Blink Dictionary*** has the following structure:

```
“Payload”: {
        “on_time”: 1.0,
        “off_time”: 1.0,
        “n”: None,
        “background”: False
        }
```

mqtt_PWMOutputDevice

You instantiate a mqtt_PWMOutputDevice by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_PWMOutputDevice name="PWMOD1" pin="12">
        <Static active_high="True" initial_value="0" frequency="100" pin_factory="None"/> <!-- Optional -->
</MQTT_PWMOutputDevice>
```

This defines a mqtt_PWMOutputDevice object with the name "PWMOD1" at pin 12.

The Static sub tag allows you to change the default way the PWMOutputDevice is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/PWMOD1/ON	
/OUT/Room_ID/PWMOD1/OFF	
/OUT/Room_ID/PWMOD1/TOGGLE	
/OUT/Room_ID/PWMOD1/***BLINK***	***PWM Blink Dictionary***
/OUT/Room_ID/PWMOD1/***PULSE***	***Pulse Dictionary***

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***	
/SEND/VAL/Room_ID/PWMOD1	/VAL/Room_ID/PWMOD1	{ "Value": value }
/SEND/ISACTIVE/Room_ID/GPIOs[name]	***/STATUS/ISACTIVE/Room_ID/GPIOs[name]***	
/SEND/ISACTIVE/Room_ID/PWMOD1	/STATUS/ISACTIVE/Room_ID/PWMOD1	{"is_active": value}
/SEND/FREQUENCY/Room_ID/GPIOs[name]	***/STATUS/FREQUENCY/Room_ID/GPIOs[name]***	
/SEND/FREQUENCY/Room_ID/PWMOD1	/STATUS/FREQUENCY/Room_ID/PWMOD1	{"frequency": value}

The ***PWM Blink Dictionary*** has the following structure:

```
"Payload": {
        "on_time": 1.0,
        "off_time": 1.0,
        "fade_in_time": 0,
        "fade_out_time": 0,
        "n": None,
        "background": False
        }
```

The ***Pulse Dictionary*** has the following structure:

```
"Payload": {
        "fade_in_time": 1.0,
        "fade_out_time": 1.0,
        "n": None,
        "background": False
        }
```

mqtt_OutputDevice

You instantiate a mqtt_OutputDevice by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_OutputDevice name="OutputDevice1" pin="12">
        <Static active_high="True" initial_value="False" pin_factory="None"/>          <!-- Optional -->
</MQTT_OutputDevice>
```

This defines a mqtt_OutputDevice object with the name “OutputDevice1” at pin 12.

The Static sub tag allows you to change the default way the OutputDevice is defined in gpiozero. The value used in the example are the default values, therefore the tag is optional. They are for advanced setting like pin_factory.

For more elaborate information on the different values for each attributes see the gpiozero documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/OutputDevice1/ON
/OUT/Room_ID/OutputDevice1/OFF
/OUT/Room_ID/OutputDevice1/TOGGLE

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name] ***/VAL/Room_ID/GPIOs[name]***

/SEND/VAL/Room_ID/OutputDevice1 /VAL/Room_ID/OutputDevice1 { “Value”: value }

mqtt_SoundMixer

You instantiate a mqtt_SoundMixer by defining it into the XML of the Raspberry PI using the following structure:

```
<MQTT_SoundMixer name="SoundMixer1">
        <Static nbr_of_channels="6" frequency="48000" size="-16" mono_or_stereo="1" buffer_size="1024"/>        <!-- Optional -->
</MQTT_SoundMixer>
```

This defines a mqtt_SoundMixer object with the name “SoundMixer1”.

The Static sub tag allows you to change the default way the SoundMixer is defined in pygames. The value used in the example are the default values, therefore the tag is optional.

For more elaborate information on the different values for each attributes see the pygames documentation.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/SoundMixer1/***PLAY*** ***Play Dictionary***
/OUT/Room_ID/SoundMixer1/STOP
/OUT/Room_ID/SoundMixer1/PAUSE
/OUT/Room_ID/SoundMixer1/UNPAUSE

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***	
/SEND/VAL/Room_ID/SoundMixer1	/VAL/Room_ID/SoundMixer1	{ “Value”: value }

The ***Play Dictionary*** has the following structure:

“Payload”: {“filename”: “the file name to play”}

mqtt_2x16_lcd_I2C

You instantiate a mqtt_2x16_lcd_I2C by defining it into the XML of the Raspberry PI using the following structure:

```
<!-- MQTT 2x16_lcd_I2C (uses pins: SDA1(3), SCL1(5). GND, 5V) -->
<MQTT_2x16_lcd_I2C name="LCD1" />
```

This defines a mqtt_2x16_lcd_I2C object with the name “LCD1”.

Since the mqtt_2x16_lcd_I2C object uses the I2C, we assume that it is using the SDA1 (GPIO2, pin3), the SCL1 (GPIO3, pin5) as well as the ground and 5V. Therefore, nothing else is needed than to define only one LCD per raspberry pi.

The following are the MQTT topics listened to by this class:

/OUT/Room_ID/GPIOs[name]/What

/OUT/Room_ID/LCD1/CLEAR	
/OUT/Room_ID/LCD1/***BACKLIT***	***Backlit Dictionary***
/OUT/Room_ID/LCD1/***DISPLAYTHISATTHATLINE***	***Line Dictionary***
/OUT/Room_ID/LCD1/***DISPLAYTHISATTHATPOSITION***	***Position Dictionary***
/OUT/Room_ID/LCD1/***LCDLOADCUSTOMCHARS***	***Chars Dictionary***

The class will also listen to the /SEND/VAL message sending its value in the payload dictionary.

/SEND/VAL/Room_ID/GPIOs[name]	***/VAL/Room_ID/GPIOs[name]***
/SEND/VAL/Room_ID/LCD1	not implemented

The ***Backlit Dictionary*** has the following structure:

“Payload”: {“backlit”: 0}

The ***Line Dictionary*** has the following structure:

“Payload”: {“line”: 0, “text”: “text to display”}

The ***Position Dictionary*** has the following structure:

“Payload”: {“line”: 0, “col”: 5, “text”: “text to display”}

The ***Chars Dictionary*** has the following structure:

“Payload”: {“FontData”: “”}

About Source Connections

gpiozero allows you to connect an input to an output through the device source attribute.

```
<Connect target="Green1" function="" source="Button1"/>
```

The target and source are device names already defined in the XML by the user.

The available function are listed in the following table as well as how to use each function with their own parameters.

negated	<Connect target="Green1" function="negated" source="Button1"/>
absoluted	<Connect target="Green1" function="absoluted" source="Button1"/>
scaled	<Connect target="Green1" function="scaled" source="Button1"/> <Params outmin="-1" outmax="1" inmin="0" inmax="1"/> </Connect>
quantitized	<Connect target="Green1" function="quantitized" source="Button1"> <Params steps="" min="" max=""/> </Connect>
booleanized	<Connect target="Green1" function="booleanized" source="Button1"> <Params min="" max="" hysterisis=""/> </Connect>
clamped	<Connect target="Green1" function="clamped" source="Button1"> <Params min="" max=""/> </Connect>
inverted	<Connect target="Green1" function="inverted" source="Button1"> <Params min="" max=""/> </Connect>
queued	<Connect target="Green1" function="queued" source="Button1"> <Params size=""/> </Connect>

The following is not implemented yet

Smoothed

Furtermore, there are multi device related tags like the following that allow aggregation of multiple device's signal. The general format for those tags is as follows: The tag name, its target, and a list of devices with their function and source. The functions are the same as above. Again, the source has to be already defined in the XML to be used here.

```
<XXXXXX target="Green1">
        <Device function="" source="MotionSensor1"/>
        <Device function="booleanized" source="LightSensor1">
                <Params min="0.3" max="1" hysteresis="0"/>
        </Device>
</ XXXXXX>
```

All_Values	<All_Values target="Blue1"> List of Device tags </All_Values>
Any_Values	<Any_Values target="Red1"> List of Device tags </Any_Values>
Averaged	<Averaged target="Blue1"> List of Device tags </Averaged>
Multiplied	<Multiplied target="Yellow1"> List of Device tags </Multiplied>
Summed	<Summed target="Green1"> List of Device tags </Summed>
Zip_Values	<Zip_Values target="Blue1"> List of Device tags </Zip_Values>
Zip	<Zip target="Blue1"> List of Device tags </Zip>

Configuring the Raspberry PI

There are three versions of the RaspiOS for now … Lite, Standard, and Full.

F	***S***	***L***	Step 1 : Clean RaspiOS	
			sudo apt-get update	
			sudo apt-get install python3 python3-venv python3-pip	Already present in Full and Standard
			Sudo apt-get install python3-pygame	For sound mixer
			Sudo apt-get install i2c-tools	
			pip3 install paho-mqtt==1.3.1	New version available 1.5.0
			pip3 install RPi.GPIO smbus spidev	
			pip3 install gpiozero	Already present in Full and Standard
			pip3 install picamera	Already present in Full and Standard
			pip3 install psutil	
			sudo apt-get install exfat-fuse exfat-utils	For the 2TB and the 480GB

sudo scp michel@10.148.82.21:/home/michel/Documents/IoT/IoTCPU/*.* /home/pi/IoT/IoTCPU

To Add a screen

/boot/config.txt

```
Max_usb_current=1
hdmi_group=2
hdmi_mode=87
hdmi_cvt 1024 600 66 6 0 0 0
hdmi_drive=1
```

No space on either side of the “=” sign

hdmi_cvt width height rate aspect margin interlace rb

Note: when using Pi Imager, try pressing Ctrl_Shift-X for advanced settings !

gpiozero4mqtt.py Code Structure

Imports
Variables
Conversion Functions
Rebooting Functions

Class GPIOList

Init From XML, get basic info and MQTT connection information
From XML, find all instances of widgets defined
Inputs (Button, DigitalInputDevice, LineSensor, MotionSensor, LightSensor, DistanceSensor)
Outputs (LED, RGBLED, PWMLED, PWMOutputDevice, Buzzer, TonalBuzzer, Motor, PhaseEnableMotor, Servo, AngularServo, DigitalOutputDevice, OutputDevice SoundMixer, 2x16_lcd_I2C)
Execute Simple Widget Connect
Execute Multi Widget Connect
Connect MQTT
Add Local CallBacks (Halt, Reboot, ProgReboot/ActivateM, ProgReboot/ActivateT, ProgReboot/Cancel Send/XML, Send/SysInfo, Send/Temperature, Send/DiskInfo, Send/MemInfo)
Add GPIOs CallBacks (Send/VAL, LED, RGBLED, PWMLED, PWMOutputDevice, Buzzer, TonalBuzzer, Motor, PhaseEnableMotor, Servo, AngularServo, DigitalOutputDevice, OutputDevice, SoundMixer, 2x16_lcd_I2C)

BuildTimeStampDict
BuildJSONPayload
mqttSender

On_ local functions
on_send_xml_all, on_halt_all, on_reboot_all, on_progreboot_cancel, on_progreboot_M
on_send_xml_me, on_halt_me, on_reboot_me, on_progreboot_T
on_SysInfo, on_Temperature, on_DiskInfo, on_MemInfo

On_Connect_MQTT

Subscribe to local functions (/SEND/VAL, /HALT, /REBOOT, /PROGREBOOT/ACTIVATEM, /PROGREBOOT/ACTIVATET, /PROGREBOOT/CANCEL, /SEND/XML, /SEND/SYSINFO, /SEND/TEMPERATURE, /SEND/DISKINFO, /SEND/MEMINFO)

Subscribe to all OUTs widgets (LED, RGBLED, PWMLED, PWMOutputDevice, Buzzer, TonalBuzzer, Motor, PhaseEnableMotor, Servo, AngularServo, DigitalOutputDevice, OutputDevice, SoundMixer, 2x16_lcd_I2C)

On_subscribe_mqtt (just log the fact)

MQTTconnectNow (called from Controller)

All Class definitions

Class Input name
__init__, On_GetValue, When_event_occured

Class Output name
__init__, On_GetValue, On_PossibleAction