# Domoticz MicroPython Projects

Explore | Build | Share | Reference

**Robert W.B. Linn**

**06.05.2023**

**DISCLAIMER**

# Table of Contents

# Introduction

## Purpose

To explore how to use the MicroPython programming language running on embedded hardware interfacing with the Domoticz Home Automation System.

★ The core of the projects uses the Raspberry Pi Pico W microcontroller, with components like actuators & sensors.
The microcontroller is acting as a web server or MQTT Autodiscover client communicating with the Domoticz Home Automation System.

The intention is to provide some practical guidance, inspire ideas .. but not to explain Domoticz nor programming languages.

## Prerequisites

It is expected to have basic knowledge of

- Domoticz Home Automation System.
- Domoticz Automation Event system dzVents & Lua.
- Programming languages Python and MicroPython.
- Raspberry Pi Pico / Pico W and ESP microcontrollers.
- Thonny Integrated Development Environment.
- JavaScript Object Notation (JSON).
- Message Queuing Telemetry Transport MQTT and MQTT Autodiscover.

## Remarks

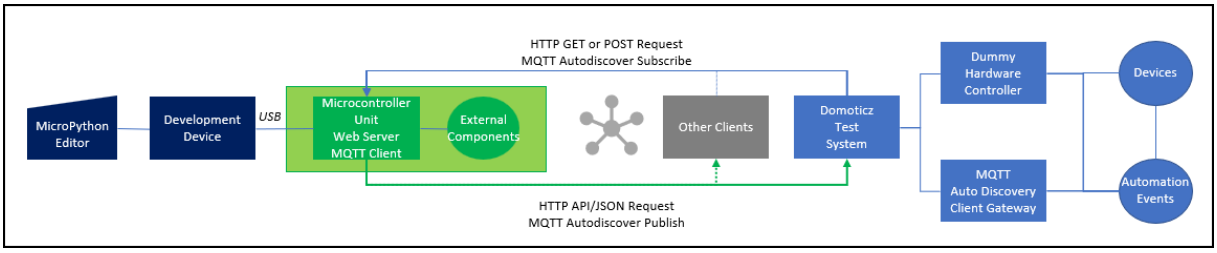
- This is a working document = conceptual changes & new idea's whilst progressing.
- There might be better solutions = changes depend on the author's learning curve.
- To-Do actions are tagged with [TODO] and captured in the file TODO.md.
- Hard- and Software versions are subject to change.
- Drawings are created with Fritzing.
- Sources are included. Get the latest sources from GitHub repository folder src.

## Credits

THANKS to the developers of the Raspberry Pi & ESP Microcontroller, Domoticz Home Automation System, MicroPython Language/Libraries/Tools and to all sharing related information. Without these, it would not be possible to write this document.

## Licence

# Concept




The block diagram starts at the left with the MicroPython editor (Thonny) running on the development device (Notebook with Windows 11).

Connected to the development device is a Microcontroller Unit (MCU) with external components (actuators & sensors).
For the Pico W projects, the Pico Breadboard Kit or the Pico IO Shield is used. These are a rather handy boards, not only for experimenting but also for building prototypes.

The MCU communicates with the Domoticz Test System via HTTP or MQTT.
The MCU acts as a

- Web Server by sending HTTP GET/POST requests to connected clients or receiving HTTP GET/POST requests from connected clients,
- MQTT Autodiscover Client by publishing device configuration or state messages and subscribing to state messages.

The connected clients can be any client (like a Web Browser, Node-RED or Application), but for this book the client is a dedicated Domoticz Test System running on a Raspberry Pi 4B 4GB with Raspberry Pi OS version 11 (bullseye).

The Domoticz hardware and related devices are added, either manually via the “Dummy Hardware Controller” or automatic via the “MQTT Auto Discovery Client Gateway with LAN interface” depending on the requirements of the project as described in this book.
*Note*
Whilst starting to write this book, most of the devices are virtual sensors assigned to the Dummy Hardware Controller, but also gradually start to use the (new) MQTT Autodiscover feature.

The Automation events are developed in dzVents (Domoticz Easy Events).
Event scripting with dzVents is well integrated in Domoticz and good documentation with many examples is available.
The Domoticz editor (GUI > Setup > More Options > Events) is used to develop and test the scripts (My Automation Scripts).

In addition, Node-RED and MQTT broker mosquitto (with clients mosquitto_pub and mosquitto_sub) are running on the Raspberry Pi.

The software is regularly updated to stay at the latest versions – for Domoticz the release channel Beta 2023.1 (build 15200 or higher) is set (at the time of writing).

# Components

- 1x Raspberry Pi 4B Domoticz Test System
- 1x Raspberry Pi Pico W 2022.
- 1x Pico Breadboard Kit GeeekPi with LEDs (4 named LED1-4), Pushbuttons (4 named Button K1 - K4), Buzzer (not used for now).
  *Hint*: If the buzzer is making unexpected noise, connect to ground.
- 1x Pico IO Shield KEYESTUDIO.
- 1x DHT22 Temperature & Humidity sensor.
- 1x LCD 20x4 LCD display (I2C) 20 columns & 4 rows.
- 1x TM1637 4-digit 7-segment LED display (I2C).
- 1x Servo Motor Tower Pro Micro Servo 9g SG90.
- 1x RFID-RC522 Reader for MIFARE RFID Cards and Tokens.
- 1x TM1638 LED&KEY 8x 7-segment decimal LED component with 8x individual LEDs and 8x push buttons.
- 1x PIR Motion Sensor.
- 1x Potentiometer.
- 2x DS18B20 1-wire digital thermometers.
- 1x 28BYJ-48 Stepper Motor with ULN2003 motor driver.
  and more …

# MicroPython

The MCU program code (scripts) is developed in MicroPython.

As most projects use the Raspberry Pi Pico W, recommend reading Get Started with MicroPython on Raspberry Pi Pico and to explore the MicroPython examples.

Please note, that the author is new to MicroPython. Whilst evolving, solutions can change.

# Development Tools

Thonny is used mostly on a Windows 11 notebook to develop the MicroPython scripts running on the MCU.
The MCU is connected to COM11 and uses the latest MicroPython firmware (nightly builds).
The Thonny view files option is active to list the files structure on the development device and the MCU.

**Example Thonny IDE Views Files, Editor & Log.**
The MicroPython script ds18b20.py is running on a Pico W, which updates in regular intervals the temperature of the Domoticz temperature device via HTTP API/JSON request.

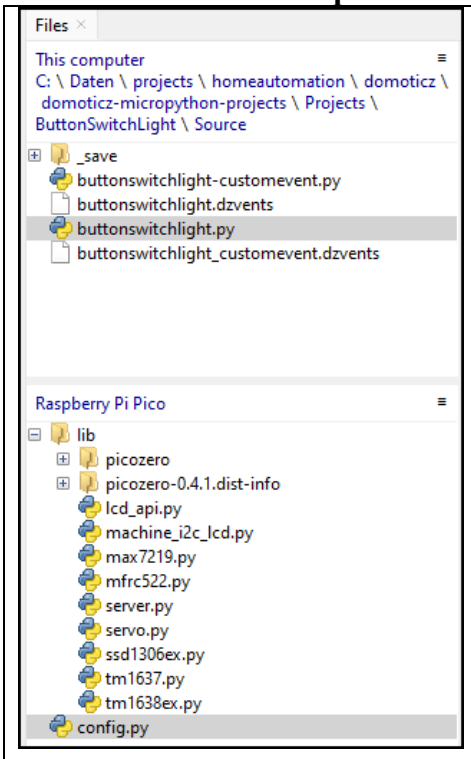

# Configuration Script

A MicroPython configuration script *config.py* is used by **all projects** shared.

The MicroPython configuration script is stored on the development device in folder Config and is uploaded to the main folder of the MCU.

⚠The *config.py* must be installed on the MCU else the projects running from the development device will fail.

## Folder Structure Development Device and Pico W - Config



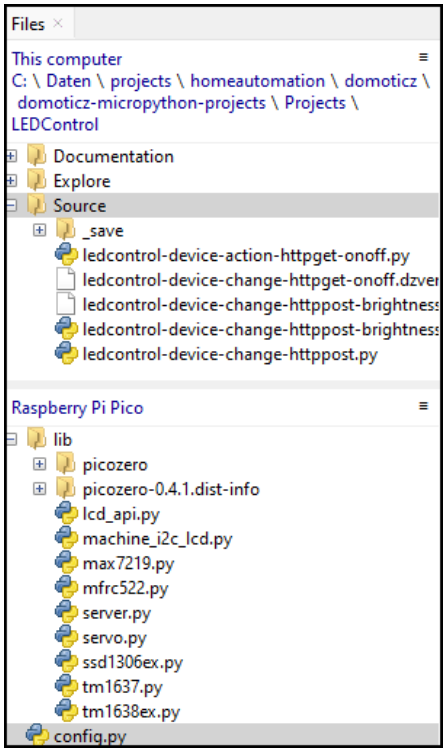

Folder structure on the development device and the Pico W.
Go to Thonny > View > Files.

The *config.py* is stored in the dedicated folder *Config* on the development device and then uploaded to the Pico W.
Upload a file to the Pico W by selecting the file, right click, select “Upload to /”.

If changes to the Pico W *config.py* file are made, ensure to download to the development device.

Download a file from the Pico W by selecting the file, right click, select “Download to…”.

## Folder Structure Development Device and Pico W – Project LEDControl




Folder structure on the development device and the Pico W. Go to Thonny > View > Files.

**This computer**

The folder structure of the projects on the development device.
The projects are running from the development device.
The picture shows the files for the project LEDControl.

**Raspberry Pi Pico**

The folder structure with the *config.py* file and additional libraries.
Whilst building the projects, no *main.py* is used (this to avoid autostart).

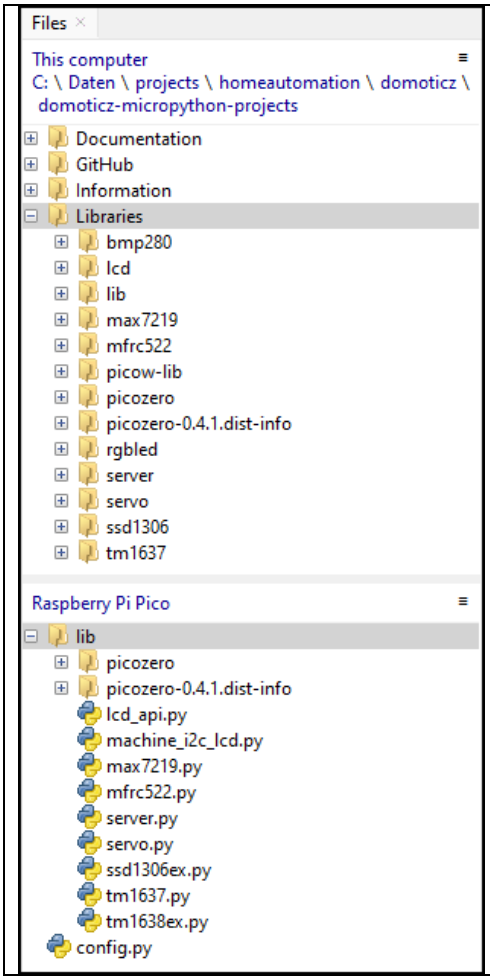
**Source Code Configuration Script – config.py**

```
"""
File:config.py
Date:202303124
Author: Robert W.B. Linn

:description
Constants for the MCU Web Server.
Specific constants for the Pico Breadboard Kit.

Import the configuration file: import config.py
Access configuration item: config.PIN_LED1
"""

# Import the const package
from micropython import const

# Network
WIFI_SSID       = const('SSID')
WIFI_PASSWORD   = const('password')

# Domoticz IP + Port
DOMOTICZ_IP  = const('domoticz-ip:port')

# Pico W onboard LED
PIN_LED_ONBOARD = const('LED')

# Pico Breadboard Kit LEDs connected to GPnn (Pin #nn)
PIN_LED1   = const(16)      #Pin 21
PIN_LED2   = const(17)      #Pin 22
PIN_LED3   = const(18)      #Pin 24
PIN_LED4   = const(19)      #Pin 25

# Pico Breadboard Kit Buttons connected to GPnn (Pin #nn)
PIN_BUTTON_K1= const(20)   #Pin 26

# Domoticz
# HTTP response JSON keys
KEY_STATE  = const('status')
KEY_TITLE  = const('title')
KEY_MESSAGE  = const('message')

# Messages used for HTTP response
STATE_ERR   = const('ERROR')
STATE_OK    = const('OK')
MESSAGE_EMPTY  = const('')
MESSAGE_UNKNOWN  = const('Unknown')
MESSAGE_CMD_UNKNOWN = const('Unknown command.')
MESSAGE_ON   = const('On')
MESSAGE_OFF    = const('Off')
```

[TODO] Optimize RAM usage for constants (see MicroPython for Microcontrollers).

# External Libraries

Several external MicroPython libraries are used and stored on the development device and the Pico W.




On the development device the MicroPython libraries are stored in the folder *Libraries*.

The content of this folder is uploaded to the Pico W folder *lib*.

If changes made to the libraries on the Pico W, then download the updates library files to the development device folder Libraries.

The folder structure on the left shows the additional libraries for using the LCD 20x4 I2C, web server, controlling a servo motor, TM1637 7-segment display.
In addition, starting to use the picozero library (BETA).

It is mandatory to store the additional library files on the Pico W else the projects will fail to run.

**Picozero Library**
The picozero library is used in some of the projects. It is easy to use for external components.

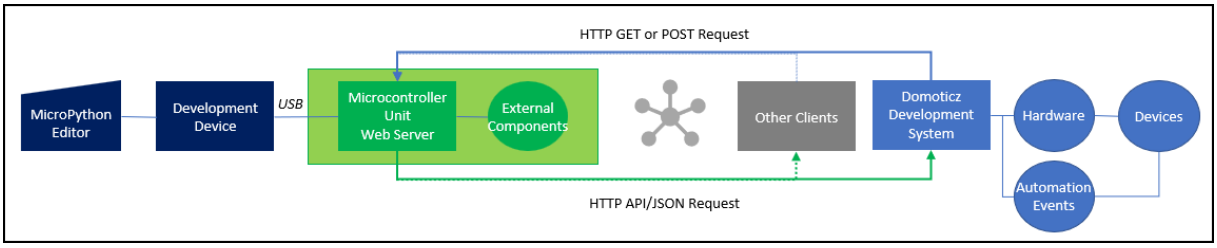
**Credits**
Thanks to the developers of the open-source libraries used in several projects.

# Web Server

The projects use, in most cases, a web server (server.py) running on the MCU developed with MicroPython.

The communication between

- The MCU Web Server and Domoticz uses Domoticz HTTP API/JSON requests,
- Domoticz and the MCU Web Server uses HTTP GET or POST requests.




# Class Server

The MCU Web Server makes use of an own developed MicroPython Class Server (server.py) to handle HTTP GET or POST requests.
This class is the core of the web server and makes it easier to create the projects.

- Create a server objected called network,
- Connect to the network,
- Get network client connection,
- Handle the GET or POST request,
- Send response back to the client.

The client in this book is by default Domoticz, but can be any other as well (curl, Node-RED, applications).

**Example**
Code Snippet to control LED1 of the Pico Breadboard Kit via HTTP command:

```
http://picow-ip/led1/on or off or state
```

```
def HandleRequest(cmd):
    if cmd == CMD_LED_ON:
        led1.on()
        response[config.KEY_MESSAGE] = config.MESSAGE_ON
        response[config.KEY_STATE] = config.STATE_OK
    elif cmd == CMD_LED_OFF:
        led1.off()
        response[config.KEY_MESSAGE] = config.MESSAGE_OFF
        response[config.KEY_STATE] = config.STATE_OK
    elif cmd == CMD_LED_STATE:
        if led1.value() == 1:
            response[config.KEY_MESSAGE] = config.MESSAGE_ON
        else:
            response[config.KEY_MESSAGE] = config.MESSAGE_OFF
```

```
        response[config.KEY_STATE] = config.STATE_OK
    else:
        response[config.KEY_STATE] = config.STATE_ERR
        response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN
    return response

network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
server = network.Connect()
while True:
    try:
        cl, request = network.GetClientConnection(server)
        response = {}
        cmd, status = network.ParseGETRequest(request)
        response[config.KEY_TITLE] = cmd
        if status == 1:
            response = HandleRequest(cmd)
        else:
            response[config.KEY_STATE] = config.STATE_ERR
            response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN
        network.SendResponse(cl, response, True)
    except OSError as e:
        cl.close()
        print('[ERROR] Network Connection closed')
```

# Pseudo Code

The steps performed when starting the web server and listening for client requests.

1. Import config.py (stored on the MCU) to get the network parameter,
2. Connect to the Network & turn MCU onboard LED (as status indicator) on,
3. If the network connection fails, the MCU onboard LED is off. Check the Thonny log,
4. Listen to incoming client connections submitted HTTP GET or POST requests,
5. Accept the client connection,
6. Get the data from the HTTP request,
7. Parse the data from the GET or POST request,
8. Act accordingly,
9. Sent HTTP response, aligning with the Domoticz HTTP API/JSON response, to the client. Example:
   {"status": "OK", "title": "/led1/on", "message": "On"}
   status: OK or ERROR
   title: Command
   message: OK or error message

[TODO] Check out the new Raspberry Pi Debug Probe Device.

### HTTP GET / POST Request Examples

Two HTTP request examples handled by the MCU Web Server.
The GET request URL contains the command to be executed by the MicroPython script.
The POST request URL contains post-data as JSON object parsed by the MicroPython script.

**HTTP GET request to set LED1 state on**
**(Project LEDControl)**
The command is sent for example from the direct On action of a Domoticz Switch Type On/Off.

```
HTTP Command: http://picow-ip/led1/on
```

```
HTTP Response: {"status": "OK", "title": "/led1/on", "message": "On"}
```

**HTTP POST request to set text on a LCD2004 I2C display**
**(Project LCDLEDControl)**
The last line of the received data contains the post-data (JSON object) sent by the Domoticz Automation Events dzVents openURL function.
The JSON object has several key:value pairs for selected Domoticz motherboard devices (Domoticz runs on a Raspberry Pi).
For each of the displayed items, the position col:row and text are defined. This is done by the Domoticz event, which enables flexibility in positioning items on the display. There is no need to change the MicroPython script for the MCU web server.

```
POST / HTTP/1.1
Host: picow-ip
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/603.36 (KHTML, like
Gecko) Chrome/53.0.34736.0 Safari/603.38
Accept: */*
Accept-Encoding: deflate, gzip, br
content-type: application/json
Content-Length: 193

{"cpuusage":{"col":6,"row":3,"text":"C:0.54"},"internaltemperature":{"col":0,"row":
3,"text":"T:40"},"memoryusage":{"col":14,"row":3,"text":"M:21"},"timestamp":{"col":
15,"row":1,"text":"18:45"}}'
```

```
HTTP Response: {"status": "OK", "title": "Set LCD", "message": ""}
```

### Domoticz Automation Event dzVents Snippet

```
local function getMotherboardData(domoticz)
  local data = {}
 data['timestamp'] = setText(1, 15, string.sub(domoticz.time.rawTime, 1, 5))
  data['internaltemperature'] = setSensor(3, 0, 'T:',
round(domoticz.devices(IDX_INTERNALTEMPERATURE).temperature, 0))
 data['cpuusage'] = setSensor(3, 6, 'C:',
round(domoticz.devices(IDX_CPUUSAGE).percentage, 2))
 data['memoryusage'] = setSensor(3, 14, 'M:',
round(domoticz.devices(IDX_MEMORYUSAGE).percentage, 0))
 return data
end

if (item.isTimer) then
  domoticz.openURL({
    url = URL_SERVER, method = 'POST', headers = { ['content-type'] =
'application/json' },
```

```
    postData = getMotherboardData(domoticz), callback = RES_HTTP })
end
```

# MicroPython Script

This is a stripped down MicroPython script example of a MCU Web Server class Server.

```
"""
File: server.py
Date: 20230425
Author: Robert W.B. Linn
"""

import network
import urequests
import socket
import time
from machine import Pin
import json

class Server:
    # Constants
    NAME = 'Server'
    VERSION = 'v20230425'

    CRLF = chr(13) + chr(10)
    SPACE = chr(32)

    # Domoticz
    # HTTP response JSON keys
    KEY_STATE= 'status'
    KEY_TITLE= 'title'
    KEY_MESSAGE = 'message'

    # Messages used for HTTP response
    STATE_OK    = 'OK'
    STATE_ERR   = 'ERROR'
    MESSAGE_EMPTY  = ''
    MESSAGE_UNKNOWN = 'Unknown'
    MESSAGE_CMD_UNKNOWN    = 'Unknown command.'
    MESSAGE_ON   = 'On'
    MESSAGE_OFF   = 'Off'

    def __init__(self, wifi_ssid, wifi_password, STATUS_PIN="LED", DEBUG=True):
        """
        Init the network with defaults.

        :param string wifi_ssid
            SSID of the network to connect

        :param string wifi_password
            Passord of the network to connect

        :param string | int STATUS_PIN
            Pin number of the LED indicating network status connected

        :param bool DEBUG
            Flag to set the log for debugging purposes
        """
        self.debug = DEBUG
        self.wifi_ssid = wifi_ssid
        self.wifi_password = wifi_password

        # Create the onboard LED object to indicate controller is up and network
connected
        self.ledstatus = Pin(STATUS_PIN, Pin.OUT)
        self.ledstatus.off()
```

```
    def log(self, msg):
        """
        Log to the console if debug flag is true.

        :param string msg
            Message to print
        """
        if self.debug:
            print(msg)

    def connect(self):
        """
        Connect to the network using the class SSID and password.
        If connected start listening for incoming connections.

        :param None

        :return object server
            Server object.

        :example
            # Create network object
            network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
            # Connect to the network and get the server object
            server = network.connect()
        """
        try:
            wlan = network.WLAN(network.STA_IF)
            wlan.active(True)
            wlan.connect(self.wifi_ssid, self.wifi_password)

            # Network connection
            self.log(f'Network waiting for connection...')
            max_wait = 10
            while max_wait > 0:
                if wlan.status() < 0 or wlan.status() >= 3:
                    break
                max_wait -= 1
                time.sleep(1)

            if wlan.status() != 3:
                self.ledstatus.off()
                raise RuntimeError('[ERROR] Network connection failed!')
            else:
                self.ledstatus.on()
                self.log(f'Network connected OK')
                status = wlan.ifconfig()
                self.log(f'Network IP ' + status[0] )

            # Network Get address
            addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]

            # Network Create the server socket
            server = socket.socket()

            # Option to reuse addr to avoid error EADDRINUSE
            server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

            # Bind the address befor starting to listen for icoming client
connections
            server.bind(addr)
            server.listen(1)
            self.log(f'Network listening on {addr}')
            # self.log(server)
            return server
        except OSError as e:
            self.ledstatus.off()
            cl.close()
```

```
            raise RuntimeError('[ERROR] Network connection closed')

    def connect2(self):
        """
        Connect to the network using the class SSID and password.

        :param None

        :example
            # Create network object
            network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
        """
        try:
            wlan = network.WLAN(network.STA_IF)
            wlan.active(True)
            wlan.connect(self.wifi_ssid, self.wifi_password)

            # Network connection
            self.log(f'Network waiting for connection...')
            max_wait = 10
            while max_wait > 0:
                if wlan.status() < 0 or wlan.status() >= 3:
                    break
                max_wait -= 1
                time.sleep(1)

            if wlan.status() != 3:
                self.ledstatus.off()
                raise RuntimeError('[ERROR] Network connection failed!')
            else:
                self.ledstatus.on()
                self.log(f'Network connected OK')
                status = wlan.ifconfig()
                self.log(f'Network IP ' + status[0] )

        except OSError as e:
            self.ledstatus.off()
            cl.close()
            raise RuntimeError('[ERROR] Network connection closed')

    def parse_get_request(self, request):
        """
        Parse the command from the HTTP GET Request.
        The first line of the request contains the command.
        The first line is split and the 2nd item holds the command + data.
        Example first line with the command:
        GET /led1/on HTTP/1.1
        The command is /led1/on.

        :param string request
            HTTP GET request

        :return string command
            Command, i.e. /led1/on

        :return int status
            0 = Error, 1 = OK

        :example
            # Parse the get data. In case of error, the status is 0.
            cmd, status = network.parse_get_request(request)
        """
        status = 0
        cmd = self.MESSAGE_CMD_UNKNOWN

        # Split the decoded request string into a list
        data = str(request.decode()).split(self.CRLF)

        # Check if there is data to get the first item
```

```
        if (len(data) > 0):
            # print(data[0])
            # Split the first item which is the command string into a list with 3
items
            cmds = data[0].split(self.SPACE)
            # Check length and get the 2nd item, i.e. /led1/on
            if len(cmds) == 3:
                cmd = cmds[1]
                status = 1
            else:
                print(f'[ERROR] HTTP GET number of command items invalid. Expect 3,
got {len(cmds)}.')
        else:
            print(f'[ERROR] HTTP GET request not valid.')
        self.log(f'HTTP Command={cmd}')

        # Return the command, i.e. /led1/on etc.
        return cmd, status

    def parse_post_request(self, request):
        """
        Parse the command from the HTTP POST Request.
        The last line of the HTTP request contains the command + data.
        The HTTP request is decoded and split as a string list.
        The last line is a JSON object with key:value pair(s).

        :param string request
            HTTP request

        :return string command
            Command as JSON key:value pair(s), i.e. {"led":1}

        :return int status
            0 = Error, 1 = OK

        :example
            # Parse the post data. In case of error, the status is 0.
            data, status = network.parse_get_request(request)
        """
        status = 0
        cmd = self.MESSAGE_CMD_UNKNOWN

        # Split the decoded request string into a list
        data = str(request.decode()).split(self.CRLF)

        # Check if there is data to get the last item
        # At least 8 items
        if (len(data) > 7):
            # JSON parse the last list item holding the command as JSON string
            # Convert the string to a JSON object
            try:
                cmd = json.loads(data[len(data) - 1])
                status = 1
            except ValueError:
                # In case the JSON data can not be parsed
                cmd = data[len(data) - 1]
                print('[ERROR] HTTP POST request not valid (ValueError).')
        else:
            print(f'[ERROR] HTTP POST request not valid (Not enought items, must be
8 or more).')
        self.log(f'HTTP Command={cmd}')

        # Return the command as JSON object, i.e. HTTP Command: {'state': 'on'}
        return cmd, status

    def get_client_connection(self, server):
        """
        Get the client connection.
```

```
        :param object server
            Server object which is listening

        :return object cl

        :return string data
            The requested data format depends on the request

        :example
            cl, request = network.get_client_connection(server)
        """
        # Get client connection
        cl, addr = server.accept()
        self.log(f'Network client connected from {addr[0]}')

        # Get the request data used to extract the command
        request = cl.recv(1024)
        # Return cl and the request data
        return cl, request

    def send_response(self, cl, response, close):
        """
        Send the response to the client, i.e. Domoticz, curl etc. as JSON object.

        :param object cl

        :param JSON response

        :param bool close
        """
        self.log(f'HTTP Response={json.dumps(response)}')

        # Important to have a blank line prior JSON response string
        # Note the use of json.dumps for the response
        cl.send('HTTP/1.1 200 OK'+self.CRLF+'content-type:
application/json'+self.CRLF+self.CRLF+json.dumps(response))

        # If flag close is set, ensure to close the connection
        if close == True:
            cl.close()
            self.log(f'Network connection closed')

    def send_get_request(self, url):
        """
        Network submit http get request to the domoticz server.

        :param string url
            URL of the HTTP request

        :return int status
            0 = Error, 1 = OK

        :return string content
            HTTP response content sent by Domoticz engine

        :example
            Update the Domoticz temp+hum device with IDX 15
            http://domoticz-
ip:port/json.htm?type=command&param=udevice&idx=15&nvalue=0&svalue=16;55;1
        """
        status = 0
        content = ''
        self.log(f'Send GET request url={url}')
        try:
            # URL encode space
            url = url.replace(' ', '%20')
            r = urequests.get(url)
            j = json.loads(r.content)
            content = j
```

```
            self.log(f'Send GET request status={j['status']}')
            r.close()
            status = 1
        except OSError as e:
            # print(f'[ERROR] Sending data {e}')
            raise Exception(f'[ERROR] Sending data {e}')
        except ValueError as e:
            # print(f'[ERROR] {e}, {r.content.decode()}')
            raise Exception(f'[ERROR] {e}, {r.content.decode()}')
        return status, content

    def send_post_request(self, url, postdata):
        """
        Network submit http post request to the domoticz server.

        :param string url
            URL of the HTTP request

        :param string postdata
            postdata as JSON object

        :return int status
            0 = Error, 1 = OK

        :example
            Trigger the Domoticz custom event named DHT22 with data JSON object
            http://domoticz-
ip:port/json.htm?type=command&param=customevent&event=DHT22&data={"h": 58, "t": 16,
"s": 0}
        """
        status = 0
        self.log(f'Send POST request url={url}, postdata={postdata}')
        try:
            r = urequests.post(url, data=json.dumps(postdata))
            j = json.loads(r.content)
            self.log(f'Send POST request status={j['status']}')
            r.close()
            status = 1
        except OSError as e:
            print(f'[ERROR] Sending data {e}')
            # raise Exception('Network Connection failed.')
        return status
```

# Pico W Projects

## LED Control

### Description

This project switches, triggered by a Domoticz Switch Device, an LED connected to the Pico W (running as a web server).

**Ideas for Use**

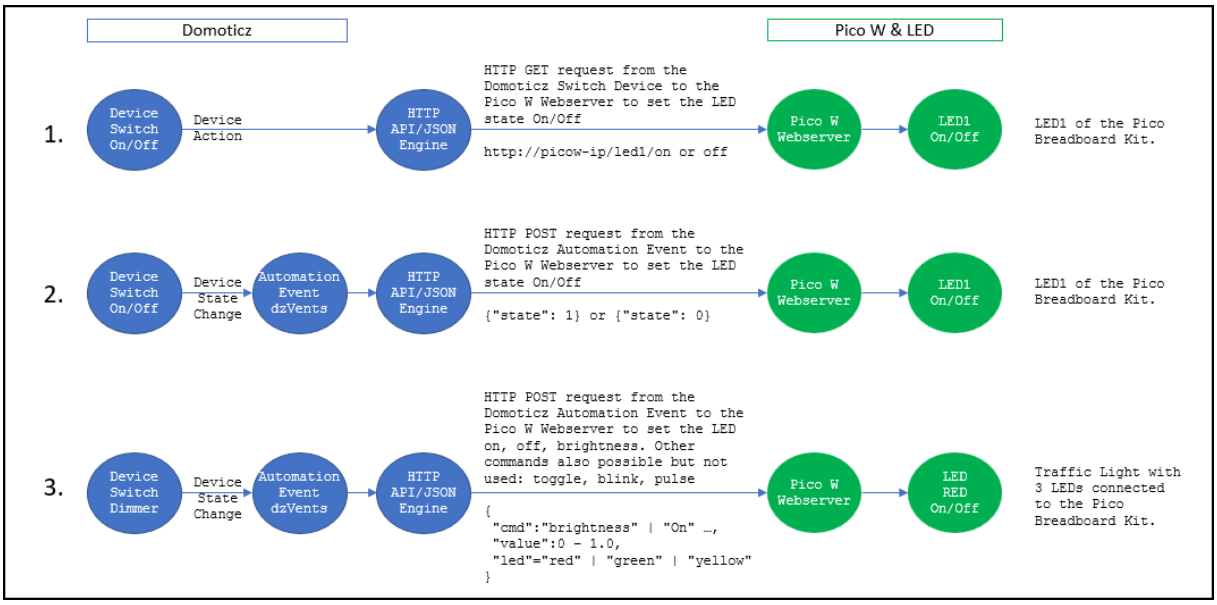
- Alarm indicators
- Status indicator

### Solution

Various solutions are developed:

1. Domoticz Switch Action triggers HTTP GET request to the Pico W web server to set the state of the LED to on or off,
2. Domoticz Switch State Change is handled by an Domoticz Automation Event dzVents which triggers HTTP POST request to the Pico W web server to set the state of the LED to on or off,
3. Domoticz Switch, configured as Dimmer, State Change is handled by an Domoticz Automation Event dzVents which triggers HTTP POST request to the Pico W web server to set the brightness of the LED between 0-100%.
   This solution also accepts other commands: on, off, toggle, blink, pulse.

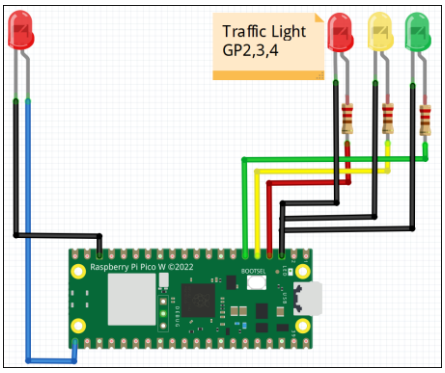
### Block Diagram



# Wiring

| **LED1** | **Pico W** |
|---|---|
| + (Anode) | GP16 (Pin #21) |
| GND (Cathode) | GND (Pin #38) |
| **Traffic Light** | |
| LED RED | GP2 (Pin #4) , GND (Pin #3) |
| LED YELLOW | GP3 (Pin #5) , GND (Pin #3) |
| LED GREEN | GP4 (Pin #6) , GND (Pin #3) |

# Circuit Diagram




# Domoticz Setup

## Devices

Create a virtual sensor, hardware dummy, named LED1 Control from sensor type Switch/Light.

After creating the device, the Domoticz devices list shows the entry:

```
Idx=16, Hardware=VirtualSensors, ID=00014060, Unit=1, Name=LED1 Control,
Type=Light/Switch, SubType=Switch, Data=On
```

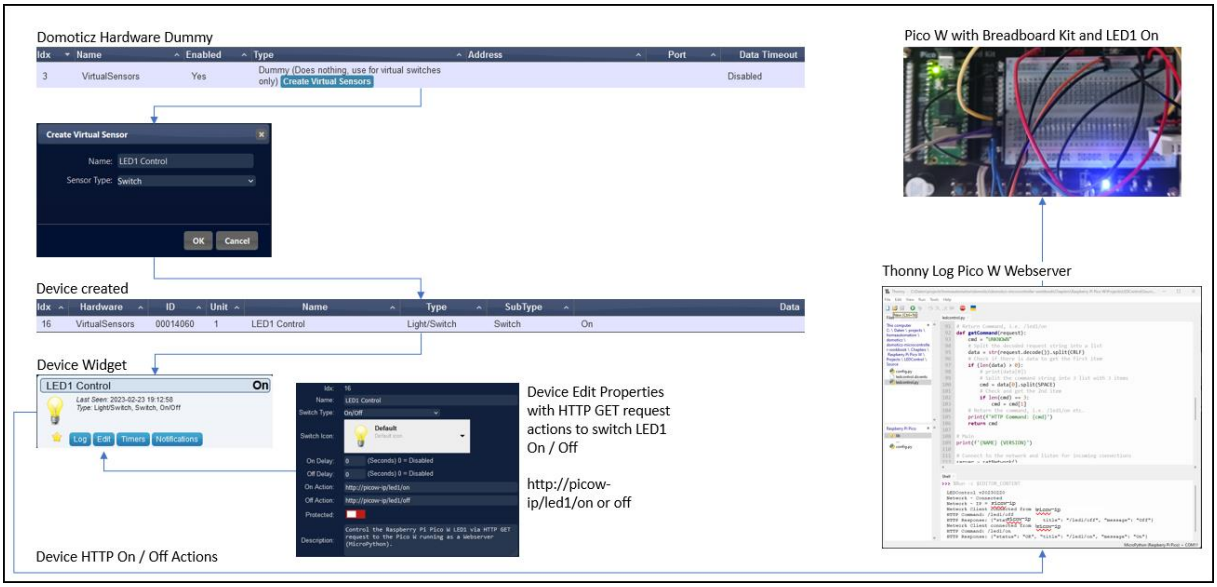
# Domoticz Switch Device Action

## Solution

Set LED1 state On/Off via a Domoticz switch device On/Off Action containing the URL for the HTTP GET request with the command /led1/on or off to the Pico W (running as a web server).

The picture shows the flow of actions from creating a switch device (virtual sensors hardware), edit the switch device On / Off actions (http://picow-ip/led1/on or off) and examples switching LED1 off and on via the device widget logged by Thonny running the Pico W web server.

## Block Diagram




Domoticz Log

```
2023-02-24 10:55:04.790 VirtualSensors: Light/Switch (LED1 Control)
2023-02-24 10:55:04.784 Status: User: admin initiated a switch command (16/LED1
Control/On)
2023-02-24 10:55:07.408 VirtualSensors: Light/Switch (LED1 Control)
2023-02-24 10:55:07.402 Status: User: admin initiated a switch command (16/LED1
Control/Off)
```

Example Log Error – wrong Pico W web server URL for On Action.

```
2023-02-24 10:56:20.321 Error: Error opening url: http://picow-ip/led1/on
```

## MicroPython Script

```
"""
File: ledcontrol-device-action-httpget-onoff.py
Date: 20230409
Author: Robert W.B. Linn

PicoW RESTful web server listening to control an LED via Domoticz Switch.
Commands are set via HTTP GET request with HTTP response JSON object.

:commands
LED ON:
HTTP Request:http://picow-ip/led1/on
HTTP response: {"status": "OK", "title": "/led1/on", "message": "On"}

LED OFF:
HTTP Request:http://picow-ip/led1/off
HTTP response: {"status": "OK", "title": "/led1/off", "message": "Off"}

LED STATE:
HTTP Request:http://picow-ip/led1/state
HTTP response: {"status": "OK", "title": "/led1/state", "message": "On"}

In case of an error:
HTTP response: {"status": "ERROR", "title": "/led1/x", "message": "Unknown
command."}

Example using curl to turn LED1 on:
curl -v http://picow-ip/led1/on

:log
ledcontrol-device-action-httpget-onoff v20230409
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Network client connected from client-ip
HTTP Command=/led1/on
HTTP Response={"title": "/led1/on", "message": "On", "status": "OK"}
Network connection closed
Network client connected from client-ip
HTTP Command=/led1/off
HTTP Response={"title": "/led1/off", "message": "Off", "status": "OK"}
Network connection closed
Network client connected from client-ip
HTTP Command=/led1/state
HTTP Response={"title": "/led1/state", "message": "Off", "status": "OK"}
Network connection closed
"""

# Libraries
import network
import socket
import time
from machine import Pin
import json
# Import server class from server.py
from server import Server
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
NAME = 'ledcontrol-device-action-httpget-onoff'
VERSION = 'v20230409'

# URL params to switch LED1 on or off or request state
# http://pico-ip/command
CMD_LED_ON  = '/led1/on'
CMD_LED_OFF  = '/led1/off'
```

```
CMD_LED_STATE= '/led1/state'

# Create the LED1 object using config.py settings
led1 = Pin(config.PIN_LED1, Pin.OUT)
# Turn the LED off
led1.value(0)

"""
Handle the request containing the command.
The LED is turned on/off or the state is requested.
The response JSON object is updated.

:param string cmd
    Command to set the LED1 state on/off or get the state.

:return JSON object response
"""
def handle_request(cmd):
    # Turn the LED on
    if cmd == CMD_LED_ON:
        led1.on()
        response[config.KEY_MESSAGE] = config.MESSAGE_ON
        response[config.KEY_STATE] = config.STATE_OK
    # Turn the LED off
    elif cmd == CMD_LED_OFF:
        led1.off()
        response[config.KEY_MESSAGE] = config.MESSAGE_OFF
        response[config.KEY_STATE] = config.STATE_OK
    # Get the LED state
    elif cmd == CMD_LED_STATE:
        if led1.value() == 1:
            response[config.KEY_MESSAGE] = config.MESSAGE_ON
        else:
            response[config.KEY_MESSAGE] = config.MESSAGE_OFF
        response[config.KEY_STATE] = config.STATE_OK
    else:
        response[config.KEY_STATE] = config.STATE_ERR
        response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN
    return response

# Main
print(f'{NAME} {VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
# Connect to the network and get the server object
server = network.connect()

while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Create the HTTP response JSON object
        response = {}

        # Parse the get data. In case of error, the status is 0.
        cmd, status = network.parse_get_request(request)

        # Assign the command to the response KEY_TITLE
        response[config.KEY_TITLE] = cmd

        # If the status is 1, handle the command
        if status == 1:
            response = handle_request(cmd)
        else:
            response[config.KEY_STATE] = config.STATE_ERR
            response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN
```

```
        # Send response to the client and close the connection
        network.send_response(cl, response, True)

    except OSError as e:
        network.ledstatus.off()
        cl.close()
        print('[ERROR] Network Connection closed')
```

# Domoticz Switch State Change

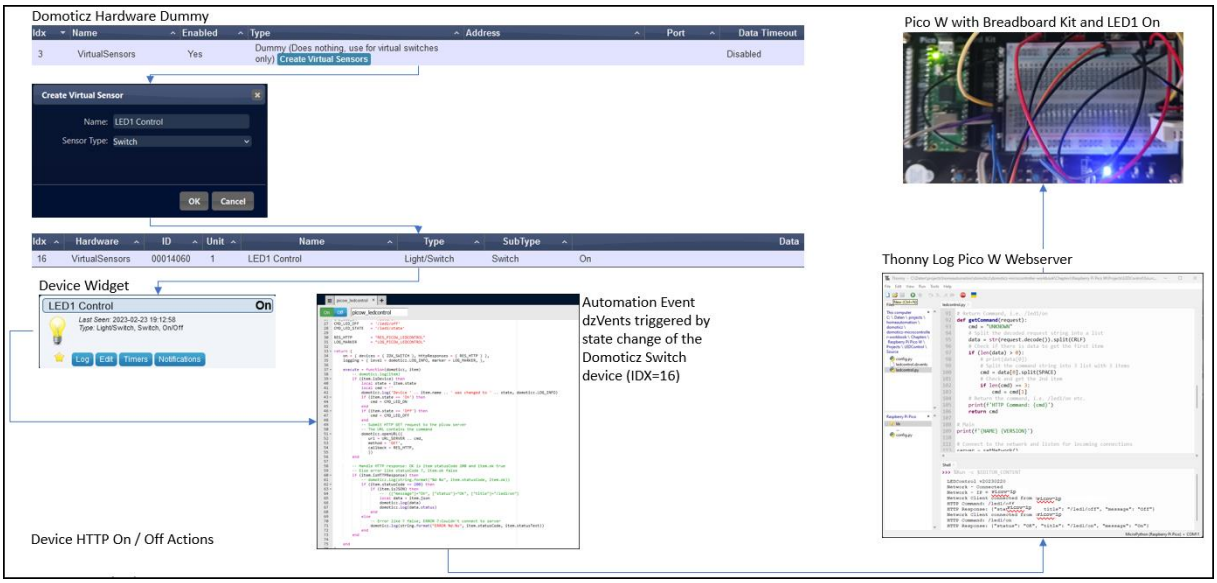
## Solution

Set LED1 state On/Off via an Domoticz Automation Event dzVents listening to the Domoticz Switch device state change. The event

1. submits the URL for the HTTP GET request via the function openURL to the Pico W running as a web server,
2. handles the Pico W web server response.

The Domoticz configuration is as described for the previous solution, except that there are no Switch device On/Off actions defined but an Automation Event dzVents is used.

## Block Diagram




## MicroPython Script

```
The same script is used as for the previous project:
ledcontrol-device-action-httpget-onoff.py
```

## Automation Script

```
-- ledcontrol-device-change-httpget-onoff.dzvents
-- Domoticz IDX of the switch triggering Pico W LED on/off
local IDX_SWITCH  = 16

local URL_SERVER      = "picow-ip"
local CMD_LED_ON      = '/led1/on'
local CMD_LED_OFF     = '/led1/off'
local CMD_LED_STATE   = '/led1/state'

local EXPERIMENT    = "LEDCONTROL-DEVICE-CHANGE"
local RES_HTTP        = "RES_" .. EXPERIMENT
local LOG_MARKER      = "LOG_" .. EXPERIMENT

return {
 on = { devices = { IDX_SWITCH }, httpResponses = { RES_HTTP } },
 logging = { level = domoticz.LOG_INFO, marker = LOG_MARKER, },
 execute = function(domoticz, item)
  if (item.isDevice) then
       local state = item.state
       local cmd = ''
      domoticz.log('Device ' .. item.name .. ' was changed to ' .. state,
domoticz.LOG_INFO)
       if (item.state == 'On') then
         cmd = CMD_LED_ON
       end
       if (item.state == 'Off') then
         cmd = CMD_LED_OFF
       end
       -- Submit HTTP GET request to the picow server
       -- The URL contains the command
       domoticz.openURL({
         url = URL_SERVER .. cmd,
         method = 'GET',
         callback = RES_HTTP,
         })
    end
    -- Handle HTTP response: OK is item statusCode 200 and item.ok true
    -- Else error like statusCode 7, item.ok false
    if (item.isHTTPResponse) then
      -- domoticz.log(string.format("%d %s", item.statusCode, item.ok))
      if (item.statusCode == 200) then
        if (item.isJSON) then
          --  {["message"]="On", ["status"]="OK", ["title"]="/led1/on"}
          local data = item.json
          domoticz.log(data)
          domoticz.log(data.status)
        end
      else
        -- Error like 7 false; ERROR 7:Couldn't connect to server
        domoticz.log(string.format("ERROR %d:%s", item.statusCode,
item.statusText))
      end
    end
 end
}
```

Domoticz Log

```
2023-02-24 11:04:05.512 VirtualSensors: Light/Switch (LED1 Control)
2023-02-24 11:04:05.506 Status: User: admin initiated a switch command (16/LED1
Control/On)
2023-02-24 11:04:05.606 Status: dzVents: Info: Handling events for: "LED1 Control",
value: "On"
2023-02-24 11:04:05.606 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE: ------
Start internal script: picow_ledcontrol: Device: "LED1 Control (VirtualSensors)",
Index: 16
2023-02-24 11:04:05.606 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE: Device
LED1 Control was changed to On
2023-02-24 11:04:05.606 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE: ------
Finished picow_ledcontrol
2023-02-24 11:04:05.606 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
2023-02-24 11:04:05.693 Status: dzVents: Info: Handling httpResponse-events for:
"RES_ LEDCONTROL-DEVICE-CHANGE "
2023-02-24 11:04:05.693 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE: ------
Start internal script: picow_ledcontrol: HTTPResponse: "RES_PICOW_LEDCONTROL"
2023-02-24 11:04:05.694 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE:
{["message"]="On", ["title"]="/led1/on", ["status"]="OK"}
2023-02-24 11:04:05.694 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE: OK
2023-02-24 11:04:05.694 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE: ------
Finished picow_ledcontrol
```

**Example Log Error**

Wrong Pico W web server URL for On Action.

```
2023-02-24 11:05:18.130 VirtualSensors: Light/Switch (LED1 Control)
2023-02-24 11:05:18.124 Status: User: admin initiated a switch command (16/LED1
Control/Off)
2023-02-24 11:05:18.214 Status: dzVents: Info: Handling events for: "LED1 Control",
value: "Off"
2023-02-24 11:05:18.215 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE: ------
Start internal script: picow_ledcontrol: Device: "LED1 Control (VirtualSensors)",
Index: 16
2023-02-24 11:05:18.215 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE: Device
LED1 Control was changed to Off
2023-02-24 11:05:18.215 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE: ------
Finished picow_ledcontrol
2023-02-24 11:05:18.215 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
2023-02-24 11:05:21.440 Status: dzVents: Info: Handling httpResponse-events for:
"RES_ LEDCONTROL-DEVICE-CHANGE "
2023-02-24 11:05:21.440 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE: ------
Start internal script: picow_ledcontrol: HTTPResponse: "RES_PICOW_LEDCONTROL"
2023-02-24 11:05:21.440 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE: ERROR
7:Couldn't connect to server
2023-02-24 11:05:21.440 Status: dzVents: Info: LEDCONTROL-DEVICE-CHANGE: ------
Finished picow_ledcontrol
2023-02-24 11:05:21.361 Error: Error opening url: http://picow-ip/led1/off
2023-02-24 11:05:21.440 Error: dzVents: Error: (3.1.8) LEDCONTROL-DEVICE-CHANGE:
HTTP/1.1 response: 7 ==>> Couldn't connect to server
```

# Domoticz Dimmer State Change

## Solution

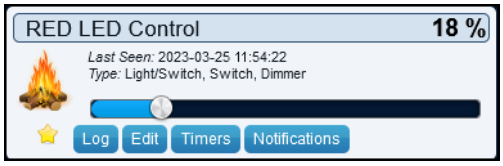
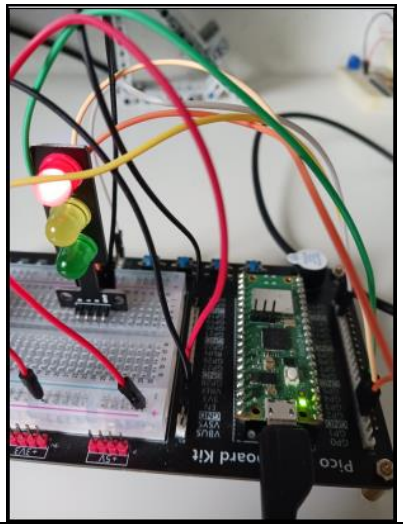
Set the brightness of an RED LED (from a Traffic Light) via a Domoticz dimmer.
A Domoticz Automation Event dzVents listens to the state change of the Domoticz dimmer.
If the state is changed, the dimmer level 0-100 is converted to a brightness value 0-1.0.
The event submits a HTTP POST request to the Pico W web server, which sets the brightness of the RED LED.
The post data submitted by the Domoticz automation event dzVents.:

```
{["cmd"]="brightness", ["value"]=0 - 1.0, ["led"]="red"}
```

The Pico W web server sends response back:

```
{
"status": "OK", "title": {"led": "red", "value": 0.18, "cmd": "brightness"},
"message": "brightness"
}
```

This solution makes use of the MicroPython picozero package to control the LED.

| Domoticz Dimmer to set the RED LED Brightness 0-100% converted to 0-1.0. | Pico W with Traffic Light LEDs connected. RED LED brightness 0-1.0 controlled. |
|---|---|
|  |  |

## Automation Script

```
-- ledcontrol-device-change-httppost-brightness.dzvents
-- Remote control, via Domoticz switch dimmer, the state of a RED LED connected the
Pico W.
-- Action triggered by the switch running function openURL with POST request:
-- http://picow-ip with postdata: {["cmd"]="brightness", ["value"]=0.39,
["led"]="red"}
-- 20230325 rwbl

-- Domoticz IDX of the dimmer triggering Pico W LED
local IDX_SWITCH  = 1

local URL_SERVER    = 'http://picow-ip'
local EXPERIMENT    = 'LEDBRIGHTNESS'
local RES_HTTP      = 'RES_' .. EXPERIMENT
local LOG_MARKER    = 'LOG_' .. EXPERIMENT

return {
 on = { devices = { IDX_SWITCH }, httpResponses = { RES_HTTP } },
    logging = { level = domoticz.LOG_INFO, marker = LOG_MARKER,    },

 execute = function(domoticz, item)
  if (item.isDevice) then
      -- Create the postdata table
           local postdata = {}
           postdata['led'] = 'red'
           postdata['cmd'] = 'brightness'
           local brightness = item.level
           if brightness > 0 then brightness = brightness / 100 end
           if item.state == 'Off' then brightness = 0 end
           postdata['value'] = brightness
      domoticz.log('Device ' .. item.name .. ' brightness changed.')
      domoticz.log(postdata)
           -- Submit HTTP POST request to the picow server
           domoticz.openURL({
              url = URL_SERVER, method = 'POST',
              postData = postdata, callback = RES_HTTP })
       end

       -- Handle HTTP response: OK is item statusCode 200 and item.ok true
       -- Else error like statusCode 7, item.ok false
       if (item.isHTTPResponse) then
           -- domoticz.log(string.format("%d %s", item.statusCode, item.ok))
           if (item.statusCode == 200) then
               if (item.isJSON) then
                   --  {["message"]="On", ["status"]="OK", ["title"]="/led1/on"}
                   local data = item.json
                   domoticz.log(data)
                   -- domoticz.log(data.status)
               end
           else
               -- Error like 7 false; ERROR 7:Couldn't connect to server
               domoticz.log(string.format("ERROR %d:%s", item.statusCode,
item.statusText))
           end
       end

 end
}
```

## MicroPython Script

```
"""
ledcontrol-device-change-httppost-brightness.py
20230409 rwbl

Pico W RESTful web server listening to set the state of an LED via Domoticz Dimmer
Switch.
The states are on, off, toggle, blink, pulse, brightness.

Commands set via HTTP GET request with HTTP response JSON object.
This script handles setting the led using the library picozero.
The picozero LED class functions can be set via the POST request.
Reference: https://picozero.readthedocs.io/en/latest/recipes.html#leds

:commands
LED ON:
curl -v -H "Content-Type: application/json" -d
"{\"led\":\"red\",\"cmd\":\"on\",\"value\":0}" http://picow-ip {"status": "OK",
"title": {"led": "red", "value": 0, "cmd": "on"}, "message": "on"}

LED OFF:
curl -v -H "Content-Type: application/json" -d
"{\"led\":\"red\",\"cmd\":\"off\",\"value\":0}" http:// picow-ip {"status": "OK",
"title": {"led": "red", "value": 0, "cmd": "off"}, "message": "off"}

:note
When using curl ensure to escape the " to \" in the JSON object.

:log
ledcontrol-device-change-httppost-brightness v20230409
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Network client connected from client-ip
HTTP Command={'led': 'yellow', 'value': 0, 'cmd': 'on'}
HTTP Response={"status": "OK", "title": {"led": "yellow", "value": 0, "cmd": "on"},
"message": "on"}
Network connection closed
Network client connected from client-ip
HTTP Command={'led': 'yellow', 'value': 0.1, 'cmd': 'brightness'}
HTTP Response={"status": "OK", "title": {"led": "yellow", "value": 0.1, "cmd":
"brightness"}, "message": "brightness"}
Network connection closed
Network client connected from client-ip
HTTP Command={'led': 'yellow', 'value': 0, 'cmd': 'off'}
HTTP Response={"status": "OK", "title": {"led": "yellow", "value": 0, "cmd":
"off"}, "message": "off"}
Network connection closed
"""

# Libraries
from machine import Pin
import json
# Picozero - note: beta version
import picozero
from picozero import LED
# Import network class
from server import Server
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
NAME = 'ledcontrol-device-change-httppost-brightness'
VERSION = 'v20230409'

# Create the 3 LED objects GP2,3,4
led_red = LED(2)
```

```
led_red.off()
led_yellow = LED(3)
led_yellow.off()
led_green = LED(4)
led_green.off()

"""
Control an LED.

:param object led
    LED object created via led = LED(GPIO pin number)

:param string cmd
    LED command, like on, off, toggle, blink, pulse, brightness

:param int|float|string value
    Value for the command, like for brightness a float between 0-1
"""
def set_led(led,cmd,value):
    # Set the command
    if cmd == 'on':
        led.on()
    elif cmd == 'off':
        led.off()
    elif cmd == 'toggle':
        led.toggle()
    elif cmd == 'blink':
        led.blink()
    elif cmd == 'pulse':
        led.pulse()
    elif cmd == 'brightness':
        if 0 <= value <= 1:
            led.brightness = value
        else:
            print(f'[ERROR] Command {cmd} value {value} out of range 0-1.')
    else:
        prinf('[ERROR] Command {cmd} unknown.')

"""
Handle the request containing the command as JSON object.
The LED is turned on or off depending on JSON key state {"state":0 or 1}
The response JSON object is updated.
Reference: https://picozero.readthedocs.io/en/latest/recipes.html#leds

:param string data
    JSON object with key:value pair to set the LED
    led:red|green|yellow, cmd:on|off|blink<pulse|brightness, value:NNN

:return JSON object response
"""
def handle_request(data):
    # Get the JSON key:
    led = data['led'].lower()
    cmd = data['cmd']
    value = data['value']

    if led == 'red':
        set_led(led_red,cmd,value)
    elif led == 'green':
        set_led(led_green,cmd,value)
    elif led == 'yellow':
        set_led(led_yellow,cmd,value)
    else:
        prinf('[ERROR] LED {led} unknown.')

    # Response is OK
    response[config.KEY_STATE] = config.STATE_OK
    response[config.KEY_MESSAGE] = cmd
    return response
```

```
# Main
print(f'{NAME} {VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
# Connect to the network and get the server object
server = network.connect()

while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Create the HTTP response JSON object
        response = {}

        # Parse the post data. In case of error, the status is 0.
        data, status = network.parse_post_request(request)

        # Assign the postdata to the response KEY_TITLE
        response[config.KEY_TITLE] = data

        # If status is 1, then the post response is properly parsed, lets change
the led state.
        if status == 1:
            response = handle_request(data)
        else:
            # Error with unknown command
            response[config.KEY_STATE] = config.STATE_ERR
            response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

        # Send response to the client and close the connection
        network.send_response(cl, response, True)

    except OSError as e:
        network.ledstatus.off()
        cl.close()
        print('[ERROR] Network Connection closed')
```

# Button Control

## Description

This project turns a Domoticz Switch (IDX=16) On/Off by pressing the Pico Breadboard Kit Button K4.
If the Domoticz switch state is On, the Pico Breadboard Kit LED1 is On else Off.
The Domoticz switch device state is changed by using HTTP API/JSON GET or POST request to the Domoticz server.

**Ideas for Use**

- Control a switch On/Off,
- Remote control, like turning a thermostat setpoint On/Off or switch a blind up/down.

## Solutions

The Pico W is built in a Pico Breadboard Kit. A RESTful web server runs on the Pico W.
If the web server network connection is successful, the Pico W onboard LED is On else Off indicating an error.

If the pushbutton K4 (of the Pico Breadboard Kit) is pressed, the Domoticz switch state with IDX NN is set to On or Off via HTTP API/JSON request to Domoticz.

Two solutions worked out:

**HTTP API/JSON GET Request with Parameter switchlight**

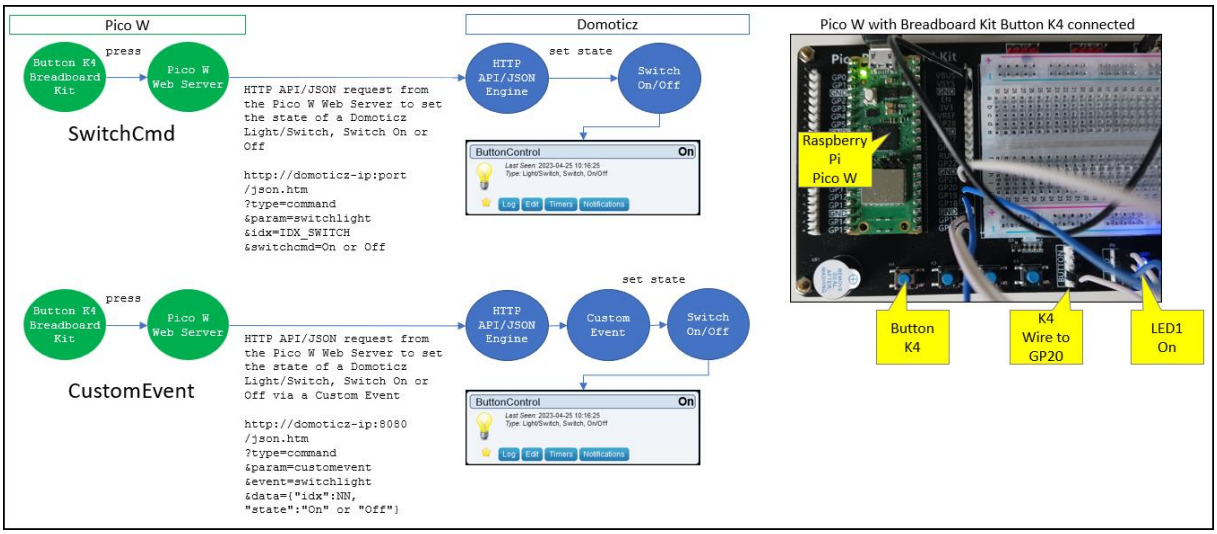
```
http://domoticz-
ip:port/json.htm?type=command&param=switchlight&idx=NN&switchcmd=<On|Off|Toggle|Sto
p>
```

Note: The switch commands On and Off are used.

**HTTP API/JSON GET Request with Parameter customevent**

```
http://domoticz-
ip:8080/json.htm?type=command&param=customevent&event=switchlight&data={"idx":NN,"s
tate":"On" or "Off"}
```

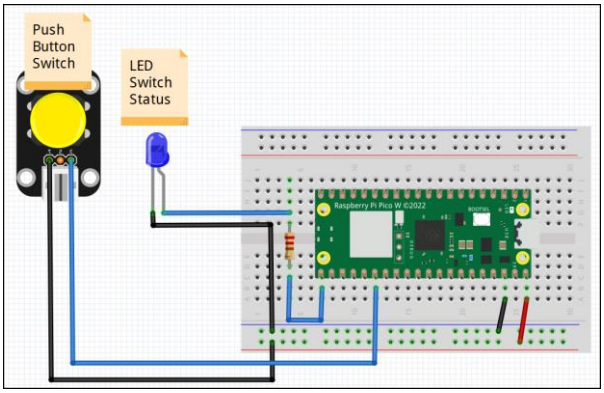
# Block Diagram




# Wiring

| **Pico Breadboard Kit** | **Pico W** |
|---|---|
| Button K4 | GP20 (Pin #26), GND (Pin #38) |
| LED RED | GP16 (Pin #21), GND (Pin #3) |

# Circuit Diagram




# Domoticz Setup

## Devices

Create a virtual sensor, hardware dummy, named “ButtonControl” from sensor type Switch/Light.

After creating the device, the Domoticz devices list shows the entry:

```
Idx=16, Hardware=VirtualSensors, ID=00014060, Unit=1, Name=ButtonControl,
Type=Light/Switch, SubType=Switch, Data=On
```

# Domoticz Device Update

## MicroPython Script

```
"""
File:   buttoncontrol.py
Date:   20230418
Author: Robert W.B. Linn

:description
Turn Domoticz Switch (IDX=16) On/Off by pressing Pico Breadboard Kit Button K4.
If the switch state is On, the Pico Breadboard Kit LED1 is On else Off.
The Domoticz switch device state is changed by using HTTP API/JSON GET request to
the Domoticz server.
http://domoticz-ip:port/json.htm?type=command&param=switchlight&idx=16&switchcmd=On
or Off

:notes
This script handles button press using the library picozero.
Configuration is stored in config.py - Ensure to upload to the picow.
The button K4 of the Pico Breadboard Kit is used.
Press the button short set the switch state On or Off.
When keeping the button down the switch state with change On/Off constantly.
Instead of using the global switch_status could use the state of LED1 to determine
if the light is on or off.

:log
ButtonControl v20230410
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Button K4 pressed - setstatus=On
Send GET request url=http://domoticz-
ip:port/json.htm?type=command&param=switchlight&idx=16&switchcmd=On
Send GET request status=OK

:wiring
Pico Breadboard Kit Button = Pico W
K4 = GP20 (Pin #26) # Pushbutton
LED1 = GP16 (Pin #21)      # LED switch state
The Pico W onboard LED is also used     to indicate the network connection state.
"""

from machine import Pin, Timer
from utime import sleep
# picozero - BETA version
from picozero import Button
# Network class to communicate with Domoticz or other clients
from server import Server
# Configuration (must be uploaded to the picow)
import config

# Constants
NAME = 'ButtonControl'
VERSION = 'v20230410'

# Breadboard Kit LED1 object using config.py settings
led1 = Pin(config.PIN_LED1, Pin.OUT)
led1.value(0)

# LED object indicating the state of the K4 button On/Off
ledbuttonstatus = Pin(config.PIN_LED1, Pin.OUT)
ledbuttonstatus.value(0)

# Button K4 pin
BUTTON_PINNR = 20
# Button K4 object
```

```
btn = Button(BUTTON_PINNR)
# Status of the switch On or Off
switch_status = "Off"

# Domoticz IDX of the switch device
IDX_SWITCH = 16

# URL Domoticz
# Note the idx of the domoticz device ( see GUI > Setup > Devices)
# http://domoticz-
ip:port/json.htm?type=command&param=switchlight&idx=99&switchcmd=<On|Off|Toggle|Sto
p>
# OK: {"status": "OK","title": "SwitchLight"}
# ERROR: {"message" : "Error sending switch command, check device/hardware
(idx=nnn) !","status" : "ERROR","title" : "SwitchLight"}
URL_DOM_SWITCH =
"http://"+config.DOMOTICZ_IP+"/json.htm?type=command&param=switchlight&idx=" +
str(IDX_SWITCH) + "&switchcmd="

"""
Handle Button Pressed.
The global switch_status is used but could also use the LED1 state.
"""
def set_domoticz_switch():
    global switch_status
    if switch_status == "Off":
        switch_status = "On"
        ledbuttonstatus.value(1)
    else:
        switch_status = "Off"
        ledbuttonstatus.value(0)
    print(f'Button K4 pressed - setstatus={switch_status}')
    # Submit HTTP API/JSON request to Domoticz to set switch status
    status = network.send_get_request(URL_DOM_SWITCH + switch_status)

"""
Main
"""
print(f'{NAME} {VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
# Connect to the network and get the server object
server = network.connect()

"""
Press the button short set the switch state On or Off.
When keeping the button down the switch state with change On/Off constantly.
"""
while True:
    # Check if the button is pressed
    if btn.is_pressed:
        # If the button is not released then set the switch state
        if btn.is_released == False:
            set_domoticz_switch()
```

## Automation Script

```
--[[
File:   buttoncontrol.dzvents
Date:   20230418
Author: Robert W.B. Linn

Handle switch device (idx=16, name=Pico W LED1 Control) state change On or Off
triggered by the Pico W web server submitting HTTP API/JSON requests.
The HTTP request submitted from the Pico W to switch On/Off:
http://domoticz-ip:port/json.htm?type=command&param=switchlight&idx=16&switchcmd=On
or Off

Log Example
2023-04-10 13:13:21.310 VirtualSensors: Light/Switch (ButtonControl)
2023-04-10 13:13:21.304 Status: User: admin initiated a switch command
(16/ButtonControl/Off)
2023-04-10 13:13:21.415 Status: dzVents: Info: Handling events for:
"ButtonControl", value: "Off"
2023-04-10 13:13:21.415 Status: dzVents: Info: LOG_BUTTONCONTROL: ------ Start
internal script: buttoncontrol: Device: "ButtonControl (VirtualSensors)", Index: 16
2023-04-10 13:13:21.415 Status: dzVents: Info: LOG_BUTTONCONTROL: Device State
Change: device=ButtonControl, state=Off
2023-04-10 13:13:21.415 Status: dzVents: Info: LOG_BUTTONCONTROL: ------ Finished
buttoncontrol
]]--

-- Domoticz IDX of the switch which state is set by the Pico W web server
local IDX_SWITCH  = 16

local LOG_MARKER = "LOG_BUTTONCONTROL"

return {
 on = {
     devices = {IDX_SWITCH}
    },
 logging = {
     level = domoticz.LOG_INFO, marker = LOG_MARKER
    },
 execute = function(domoticz, device)
  domoticz.log(string.format('Device State Change: device=%s, state=%s',
device.name, device.state), domoticz.LOG_INFO)
 end
}
```

# Domoticz Custom Event

## MicroPython Script

```
"""
File:   buttoncontrol_customevent.py
Date:   20230418
Author: Robert W.B. Linn

:description
Turn Domoticz Switch (IDX=16) On/Off by pressing Pico Breadboard Kit Button K4.
If the switch state is On, the Pico Breadboard Kit LED1 is On else Off.
The Domoticz switch device state is changed by using HTTP API/JSON POST request to
the Domoticz server.
Example:
http://domoticz-
ip:8080/json.htm?type=command&param=customevent&event=switchlight&data={"idx":16,"s
tate":"On"}
http://domoticz-
ip:8080/json.htm?type=command&param=customevent&event=switchlight&data={"idx":16,"s
tate":"Off"}

:notes
This script handles button press using the library picozero.
Configuration is stored in config.py - Ensure to upload to the picow.
The button K4 of the Pico Breadboard Kit is used.
Press the button short set the switch state On or Off.
When keeping the button down the switch state with change On/Off constantly.

:log
ButtonControl Custom Event v20230410
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Button K4 pressed - setstatus=On
Send POST request url=http://domoticz-
ip:port/json.htm?type=command&param=customevent&event=switchlight&data=,
postdata={'idx': 16, 'state': 'On'}
Send POST request status=OK
Button K4 pressed - setstatus=Off
Send POST request url=http://domoticz-
ip:port/json.htm?type=command&param=customevent&event=switchlight&data=,
postdata={'idx': 16, 'state': 'Off'}
Send POST request status=OK

:wiring
Pico Breadboard Kit Button = Pico W
K4 = GP20 (Pin #26) # Pushbutton
LED1 = GP16 (Pin #21)      # LED switch state
The Pico W onboard LED is also used to indicate the network connection state.
"""

# Convert the Domoticz HTTP API/JSON response
import json
from machine import Pin, Timer
from utime import sleep
# picozero - BETA version - beginner-friendly library to help you use common
electronics components with the Raspberry Pi Pico. Thanks to Raspberry Pi
Foundation.
from picozero import Button
# Network class to communicate with Domoticz or other clients
from server import Server
# Configuration (must be uploaded to the picow)
import config

# Constants
NAME = 'ButtonControl Custom Event'
```

```
VERSION = 'v20230410'

# Create the led object indicating the state of the K4 button On/Off
ledbuttonstatus = Pin(config.PIN_LED1, Pin.OUT)
ledbuttonstatus.value(0)

# Button K4 pin
BUTTON_PINNR = 20
# Button K4 object
btn = Button(BUTTON_PINNR)
switch_status = "Off"

## Domoticz IDX of the Switch device
IDX_SWITCH = 16

# URL Domoticz
# Note the idx of the domoticz device ( see GUI > Setup > Devices)
# http://domoticz-
ip:port/json.htm?type=command&param=customevent&event=MyEvent&data=MyData
URL_DOM_SWITCH = "http://"+ config.DOMOTICZ_IP
+"/json.htm?type=command&param=customevent&event=switchlight&data="

# Handle Key Pressed
def set_domoticz_switch():
    global switch_status
    if switch_status == "Off":
        switch_status = "On"
        ledbuttonstatus.value(1)
    else:
        switch_status = "Off"
        ledbuttonstatus.value(0)
    print(f'Button K4 pressed - setstatus={switch_status}')
    # Post data
    postdata = {}
    postdata['idx'] = 16
    postdata['state'] = switch_status
    # Submit domoticz
    status = network.send_post_request(URL_DOM_SWITCH, postdata)

# Main
print(f'{NAME} {VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
# Connect to the network and get the server object
server = network.connect()

"""
Press the button short set the switch state On or Off.
When keeping the button down the switch state with change On/Off constantly.
"""
while True:
    # Check if the button is pressed
    if btn.is_pressed:
        # If the button is not released then set the switch state
        if btn.is_released == False:
            set_domoticz_switch()
```

## Automation Script

```
--[[
File:   buttoncontrol_customevent.dzvents
Date:   20230418
Author: Robert W.B. Linn

Handle switch device (idx=16, name=Pico W LED1 Control) state change On or Off
triggered by the Pico W web server submitting HTTP API/JSON requests.
The HTTP request  submitted from the Pico W to switch On/Off:
http://domoticz-
ip:port/json.htm?type=command&param=customevent&event=switchlight&data={"idx":16,"s
tate":"On"} or {"idx":16,"state":"Off"}
]]--

-- Custom event name as used by the Pico W web server HTTP API/JSON POST request
local CUSTOM_EVENT_NAME = 'switchlight'

return {
 on = {
  customEvents = { CUSTOM_EVENT_NAME }
 },
 data = {},
 logging = {},
 execute = function(domoticz, triggeredItem)
  if (triggeredItem.isCustomEvent) then
      -- domoticz.log(triggeredItem.data)
           -- Check the custom event name in case there are more custom events
      if (triggeredItem.trigger == CUSTOM_EVENT_NAME) then
               -- Get the JSON object from the triggered item
       local data = triggeredItem.json
               -- Log to check idx and state to be set
       domoticz.log(string.format('idx=%d,state=%s', data.idx, data.state))
       -- Set the light state
       domoticz.devices(data.idx).setState(data.state)
      end
  end
 end
}
```

# Button Control MQTT Autodiscover

## Description

This project auto creates a Domoticz Push On Button device which is controlled via Pico W Button.

## Solution

This project is based up the project Button Control.

Instead communicating via HTTP, the Domoticz hardware controller MQTT Auto Discovery Client Gateway with LAN interface (MQTT Autodiscover) is used.

The Pico W connects to the network and to the Domoticz Client Gateways for MQTT & MQTT Auto Discovery.
The config & state messages published by the Pico W are handled by the Domoticz hardware controller MQTT Auto Discovery Client Gateway with LAN interface.

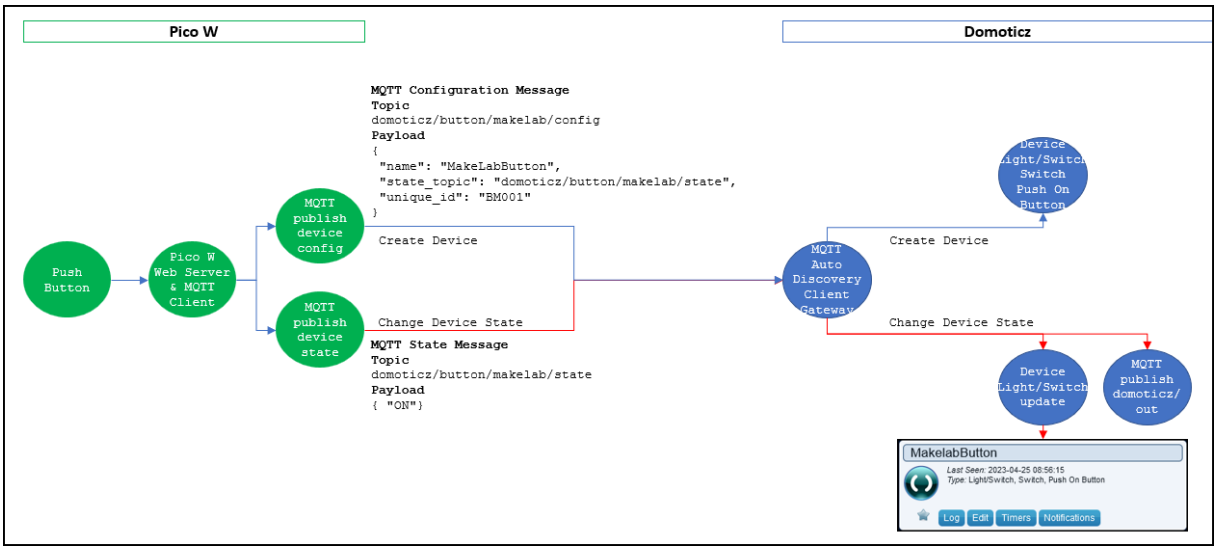
**Step 1** – Publish MQTT Config message to create the Domoticz Push On Button device.
If these devices are already in place, Domoticz takes no action.

The Domoticz devices list with the device auto created assigned to the MQTT Autodiscover hardware (named MQTTADGateway):

| | Idx | Hardware | ID | Unit | Name | Type | SubType | |
|---|---|---|---|---|---|---|---|---|
| | 41 | MQTTADGateway | BM001 | 1 | MakelabButton | Light/Switch | Switch | On |

**Step 2** – Publish MQTT State message after pressing the button.
The MQTT messages published are picked up by Domoticz and the device state is updated.
See block diagram.

# Block Diagram



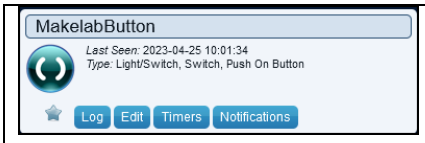

# MQTT Autodiscover Topics & Payload

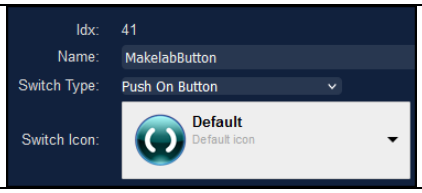
| **CONFIGURATION** | |
|---|---|
| **Light/Switch Device** | |
| Configuration Topic | domoticz/button/makelab/config |
| Configuration Payload | {<br>"name": "MakeLabButton",<br>"state_topic": "domoticz/button/makelab/state",<br>"unique_id": "BM001"<br>} |
| | |
| **STATE** | |
| Topic | domoticz/button/makelab/state |
| Payload | {<br>"ON"<br>} |

# Domoticz

## Devices

The device widget created via MQTT Autodiscover. Note the switch type is “Push On Button”.




## Debug Log

Domoticz Debug Log showing the MQTT Autodiscover gateway receiving the message with the payload.
The message is published using Domoticz topic “domoticz/out”.

```
2023-04-25 10:01:34.149 Debug: MQTTADGateway: topic: domoticz/button/makelab/state,
message: {"ON"}
2023-04-25 10:01:34.280 Debug: MQTTADGateway: topic: domoticz/out, message: {
2023-04-25 10:01:34.280 "Battery" : 255,
2023-04-25 10:01:34.280 "LastUpdate" : "2023-04-25 10:01:34",
2023-04-25 10:01:34.280 "RSSI" : 12,
2023-04-25 10:01:34.280 "description" : "",
2023-04-25 10:01:34.280 "dtype" : "Light/Switch",
2023-04-25 10:01:34.280 "hwid" : "6",
2023-04-25 10:01:34.280 "id" : "BM001",
2023-04-25 10:01:34.280 "idx" : 41,
2023-04-25 10:01:34.280 "name" : "MakelabButton",
2023-04-25 10:01:34.280 "nvalue" : 1,
2023-04-25 10:01:34.280 "stype" : "Switch",
2023-04-25 10:01:34.280 "svalue1" : "0",
2023-04-25 10:01:34.280 "switchType" : "Push On Button",
2023-04-25 10:01:34.280 "unit" : 1
2023-04-25 10:01:34.280 }
```

## Automation Script

There is no automation script used, but … if want to capture device changes, this is a simple example.

```
--[[
File:   buttoncontrol_mqtt_ad.dzvents
Date:   20230425
Author: Robert W.B. Linn

:description
Listen to device changes which has been created using MQTT Autodiscover.

:log
INFO: Log after Domoticz restart
2023-04-25 08:53:18.791 Debug: MQTTADGateway: topic:
domoticz/button/makelab/config, message: {"name": "MakelabButton", "state_topic":
"domoticz/button/makelab/state", "unique_id": "BM001"}
2023-04-25 08:53:18.892 Debug: MQTTADGateway: topic: domoticz/status, message:
online
```

```
INFO: Log after pressing button K4
2023-04-25 08:53:27.610 MQTTADGateway: Light/Switch/Switch (MakelabButton)
2023-04-25 08:53:27.710 Status: dzVents: Info: Handling events for:
"MakelabButton", value: "On"
2023-04-25 08:53:27.710 Status: dzVents: Info: BUTTONCONTROL_MQTT_AD: ------ Start
internal script: buttoncontrol_mqtt_ad: Device: "MakelabButton (MQTTADGateway)",
Index: 41
2023-04-25 08:53:27.710 Status: dzVents: Info: BUTTONCONTROL_MQTT_AD: Device
changed: name=MakelabButton, unique_id=BM001, state=On
2023-04-25 08:53:27.710 Status: dzVents: Info: BUTTONCONTROL_MQTT_AD: ------
Finished buttoncontrol_mqtt_ad
]]--

-- IDX of the device
local IDX_BUTTON = 41

return {
 on = { devices = { IDX_BUTTON } },
 logging = { level = domoticz.LOG_INFO, marker = 'BUTTONCONTROL_MQTT_AD', },
 execute = function(domoticz, device)
  domoticz.log(string.format('Device changed: name=%s, unique_id=%s, state=%s',
      device.name, device.deviceId, device.state))
 end
}
```

# Web Server

## Libraries

The MicroPython script uses the MicroPython library Lightweight MQTT client for MicroPython “umqtt.simple”.

**Credits**
Thanks for developing & sharing the MicroPython library [micropython-umqqt.simple].

## MicroPython Script

```
"""
File:   buttoncontrol_mqtt_ad.py
Date:   20230425
Author: Robert W.B. Linn

:description
Test MQTT auto discovery with Domoticz and the Pico W running as web server.
Domoticz Hardware: MQTT Auto Discovery Client Gateway with LAN Interface,
Name=MQTTADGateway
Domoticz Device created via MQTT auto discovery by publishing config topic (see
domoticz log below).
The autodiscover component is button and the Domoticz device is type=Light/Switch.
Note that this script does not use the library server.py but has simple code to
connect to the network.

:external libraries
umqtt.simple

MQTT Remove Retained message for creating the button:
mosquitto_sub -h localhost --remove-retained -t 'domoticz/button/makelab/config' -W
1
Restart Domoticz:
sudo service domoticz.sh restart

:thonny log
buttoncontrol_mqtt_ad v20230425
Network connecting...
Network connected: picow-ip
MQTT Broker connecting...
MQTT Broker connected: 192.168.1.179
MQTT State published: topic=domoticz/button/makelab/config, payload={"name":
"MakeLabButton", "state_topic": "domoticz/button/makelab/state", "unique_id":
"BM001"}
MQTT published: topic=domoticz/button/makelab/state, payload={"ON"}

:wiring
Button = Pico W
Button K4 = GP20 (Pin #26)
GND (-) = GND (Pin #28)
"""

# Imports
import network
import time
from machine import Pin
# picozero - BETA version (Installed via Thonny manage packages)
from picozero import Button
# mqtt simple (Installed via Thonny manage packages)
from umqtt.simple import MQTTClient
# Configuration (must be uploaded to the pico w)
import config

# Constants
```

```
VERSION = const('buttoncontrol_mqtt_ad v20230425')

"""
BUTTON
"""
# Button K4 pin
BUTTON_PINNR = 20
# Button object
btn = Button(BUTTON_PINNR)

# MQTT publish with encoded topic & payload (buffered objects required).
# The topic is the state_topic as defined in the configuration.
# The payload is the string ON (uppercase).
def button_pressed():
    client.publish(STATE_TOPIC, STATE_PAYLOAD)
    print(f'MQTT published: topic={STATE_TOPIC.decode()},
payload={STATE_PAYLOAD.decode()}')

# Set the function to run if the button is pressed
btn.when_pressed = button_pressed

"""
MQTT
"""
MQTT_BROKER  = const('192.168.1.179')
MQTT_CLIENT_ID= const('picow_button_k4')

# Topic & payload to create the device. The configuration component is button.
BUTTON_TOPIC_CONFIG = const(b'domoticz/button/makelab/config')
BUTTON_PAYLOAD_CONFIG      = const(b'{"name": "MakeLabButton", "state_topic":
"domoticz/button/makelab/state", "unique_id": "BM001"}')

# Topic used to publish the state of the button
# Payload is JSON format (uppercase): {"ON"}
STATE_TOPIC  = const(b'domoticz/button/makelab/state')
STATE_PAYLOAD   = const(b'{"ON"}')

# Connect to the MQTT broker with client id and server address
def mqtt_connect(client_id, mqtt_server):
    client = MQTTClient(client_id, mqtt_server, keepalive=3600)
    client.connect()
    print(f'MQTT Broker connecting...')
    return client

# Reconnect to the MQTT broker
def mqtt_reconnect():
    print('[ERROR] Failed to connect to the MQTT Broker. Reconnecting...')
    time.sleep(5)
    machine.reset()

# Info
print(f'{VERSION}')

# Connect to the network
try:
    print(f'Network connecting...')
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(config.WIFI_SSID, config.WIFI_PASSWORD)
    time.sleep(5)
    if not wlan.isconnected():
        raise RuntimeError('[ERROR] Can not connect to the network.')
    print(f'Network connected: {wlan.ifconfig()[0]}')
except Exception as e:
    raise RuntimeError('[ERROR] Can not connect to the network.')

# Connect to MQTT
# Publish mqtt auto discovery topic & payload for the switch device (button)
try:
```

```
    # Connect to MQTT broker
    client = mqtt_connect(MQTT_CLIENT_ID, MQTT_BROKER)
    print(f'MQTT Broker connected: {MQTT_BROKER}')

    # Publish config topics to auto create the device in domoticz
    client.publish(BUTTON_TOPIC_CONFIG, BUTTON_PAYLOAD_CONFIG)
    print(f'MQTT State published: topic={BUTTON_TOPIC_CONFIG.decode()},
payload={BUTTON_PAYLOAD_CONFIG.decode()}')

except OSError as e:
    mqtt_reconnect()

# Main
while True:
    pass
```

# DHT22 Temperature & Humidity

## Description

This project reads in regular intervals the DHT22 sensor temperature & humidity, connected to the Pico W, and sent the data to a Domoticz Temperature + Humidity device.

**Ideas for Use**

- Mini weather station,
- Room temperature & humidity control.

# Solutions

The Pico W has a DHT22 sensor connected and runs as a web server.
The web server reads in regular intervals the temperature & humidity.

Two solutions worked out:

**Domoticz Device Update**
The DHT22 sensor data is sent to Domoticz via HTTP API/JSON GET request with parameter:

- udevice – used for the command,
- svalue - contains the temperature, humidity, humidity status,
- idx – the Domoticz temperature humidity device to be updated.

```
http://domoticz-
ip:port/json.htm?type=command&param=udevice&idx=15&nvalue=0&svalue=16;64;0
```

**Domoticz Custom Event**
The DHT22 sensor data is sent to Domoticz via HTTP API/JSON POST request with parameter:

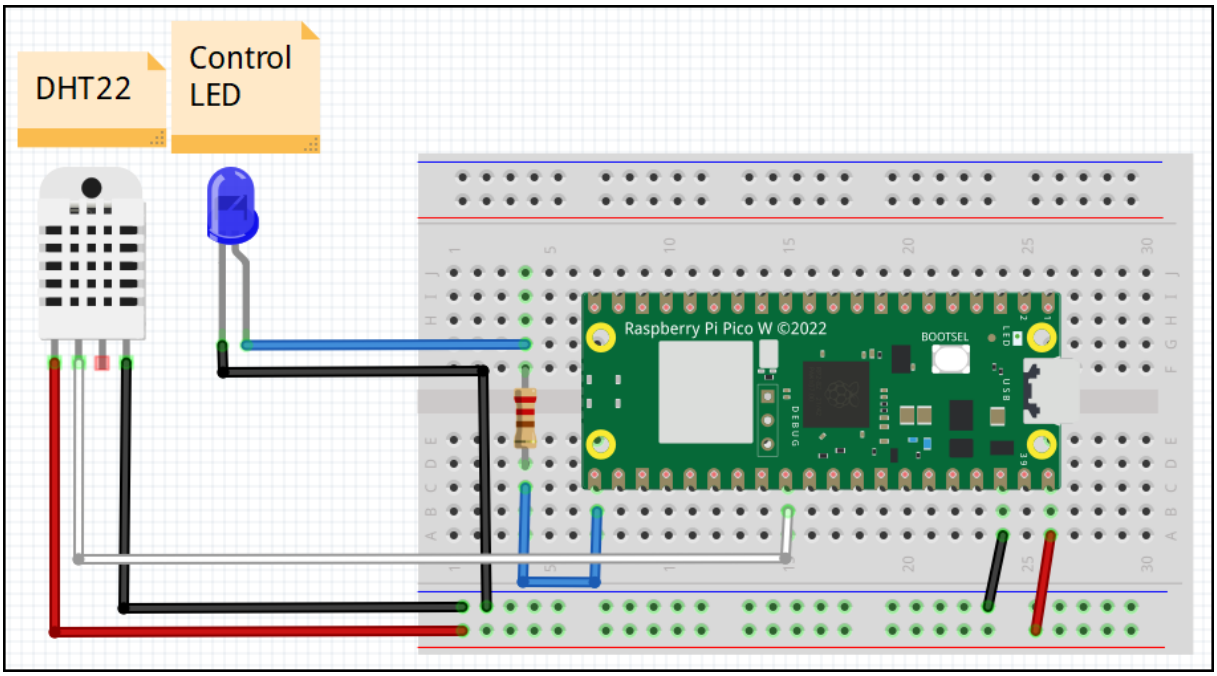
- customevent – used for the command,
- event - the custom event name,
- data - contains JSON object temperature (t), humidity (h), and humidity status (s).

```
http://domoticz-
ip:port/json.htm?type=command&param=customevent&event=DHT22&data={"h": 56, "t": 16,
"s": 0}
```

# Wiring

| **DHT22** | **Pico W** |
|---|---|
| VCC (+) | VBUS (Pin #40) |
| OUT | GP22 (Pin #29) |
| GND (-) | GND (Pin #38) |
| | |
| **LED (blue)** | **Pico W** |
| + (Anode) | 16 (Pin #21) |
| GND (Cathode) | GND (Pin #38) |

# Circuit Diagram

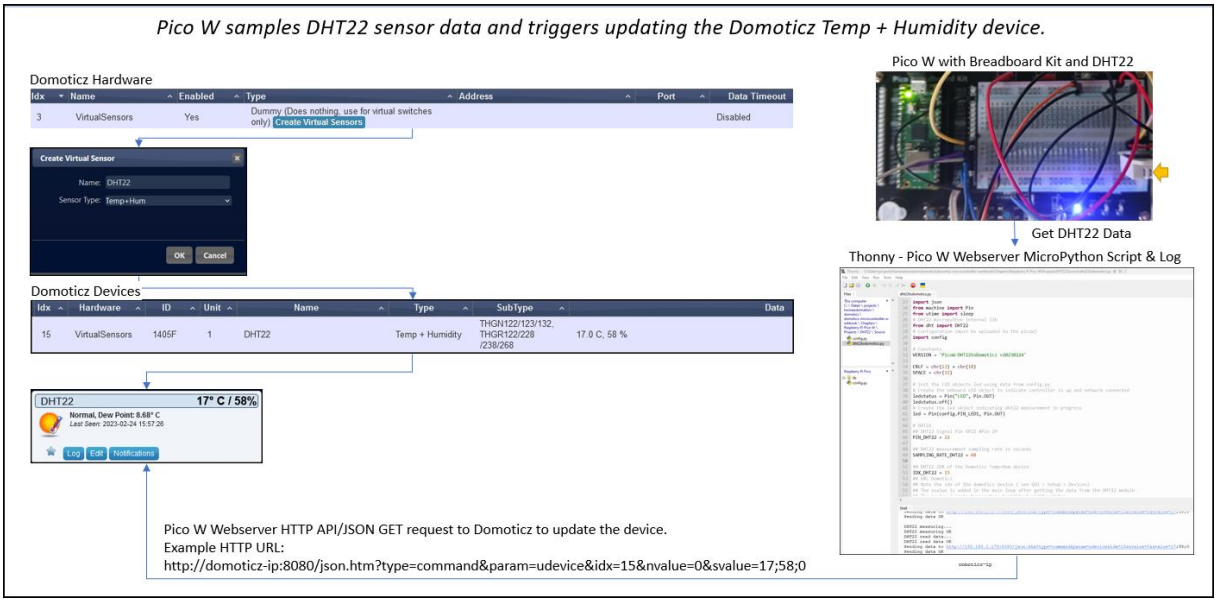


# Domoticz Device Update

The picture shows the flow of actions:

1. Create the Domoticz Temp+Hum device from the hardware “Dummy” (Create Virtual Sensors) ,
2. Run the Pico W web server and request every 60 seconds the DHT22 data,
3. After receiving data, the Pico W web server submits the data to Domoticz via HTTP API/JSON GET request with the command “udevice” for the device “idx” and the data as “svalue”,
4. Domoticz handles the incoming HTTP request and updates the device.

## Block Diagram



## MicroPython Script

```
"""
File:   dht22.py
Date:   20230318
Author: Robert W.B. Linn

Read in regular intervals the DHT22 temperature and humidity and update the svalue
(i.e., 16;55;1) of a Domoticz device named DHT22.
The Domoticz device is updated using HTTP API/JSON request to the Domoticz server.

:notes
Pico Breadboard Kit is used to wire up the DHT22.
Pico Breadboard Kit LED1 is used as status LED when requesting DHT22 data and
updating domoticz.
Configuration stored in config.py, ensure to upload to the picow.
DHT22 measures every 60 seconds.

:log
DHT22 v20230311
Sampling Rate: 60s.
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
DHT22 t=19, h=43, hs=0, svalue=19;43;0
Send GET request url=http://domoticz-
ip:port/json.htm?type=command&param=udevice&idx=15&nvalue=0&svalue=19;43;0
Send GET request status=OK

:wiring
DHT22 = Pico W
VCC (+) = VBUS (Pin #40)
OUT = GP22 (Pin #29)
GND (-) = GND (Pin #28)
"""

# Imports
from machine import Pin
from utime import sleep
# DHT22 micropython internal lib
from dht import DHT22
# Call server from server.py (must be uploaded to the picow)
from server import Server
# Configuration (must be uploaded to the picow)
import config

# Constants
VERSION = 'DHT22 v20230311'

# Create the led object indicating dht22 measurement in progress
led1 = Pin(config.PIN_LED1, Pin.OUT)
led1.value(0)

# DHT22 Signal Pin GP22 #Pin 29
PIN_DHT22 = 22
# DHT22 measurement sampling rate in seconds
SAMPLING_RATE_DHT22 = 60
# DHT22 IDX of the Domoticz Temp+Hum device
IDX_DHT22 = 15
# URL Domoticz
# Note the idx of the domoticz device ( see GUI > Setup > Devices)
# The svalue is added in the main loop after getting the data from the DHT22
module.
# The svalue format: temperature,humidity,humidity status
URL_DOM_DHT22 = "http://"+ config.DOMOTICZ_IP
+"/json.htm?type=command&param=udevice&idx=" + str(IDX_DHT22) + "&nvalue=0&svalue="

# Create the dht22 sensor object
```

```
dht22_sensor = DHT22(Pin(Pin_DHT22, Pin.IN, Pin.PULL_UP))

"""
Set the humidity status level used for Domoticz HUM_STAT value.

:param int hum
    0: NORMAL, 1: COMFORTABLE, 2: DRY, 3: WET

:return int level
    Humidity level 0 - 3
"""
def set_humidity_status(hum, temp):
    level = 9
    # 2 = Dry
    if hum <= 30:
        level = 2
    # 3 = Wet
    elif hum >= 70:
        level = 3
    # 1 = Comfortable
    elif hum >= 35 and hum <= 65 and temp >=22 and temp <= 26:
        level = 1
    # 0 = Normal
    else:
        level = 0
    return level

"""
DHT22 measurement with roundes values for temperature and humidity.
During measurement, LED1 of the Pico Breadboard is on.

:return string svalue
    svalue with temperature (°C), humidity (0-100%) and humidity_status (0-3)

:example
    16;55;1
"""
def get_dht22_data():
    led1.value(1)
    sleep(1)
    # print(f'DHT22 measuring...')
    dht22_sensor.measure()
    # print(f'DHT22 measuring OK')

    # print(f'DHT22 read data...')
    # Assign the data (rounded)
    temperature     = round(dht22_sensor.temperature())
    humidity        = round(dht22_sensor.humidity())
    humidity_status = set_humidity_status(humidity, temperature)

    # Set the svalue, i.e. svalue=TEMP;HUM;HUM_STAT
    svalue = str(temperature) + ';' + str(humidity) + ';' + str(humidity_status)
    led1.value(0)
    print(f'DHT22 t={temperature}, h={humidity}, hs={humidity_status},
svalue={svalue}')
    return svalue

# Info
print(f'{VERSION}')
print(f'Sampling Rate: {SAMPLING_RATE_DHT22}s.')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()

# Main
# Measure DHT22 every NN seconds (see constant SAMPLING_DELAY)
```

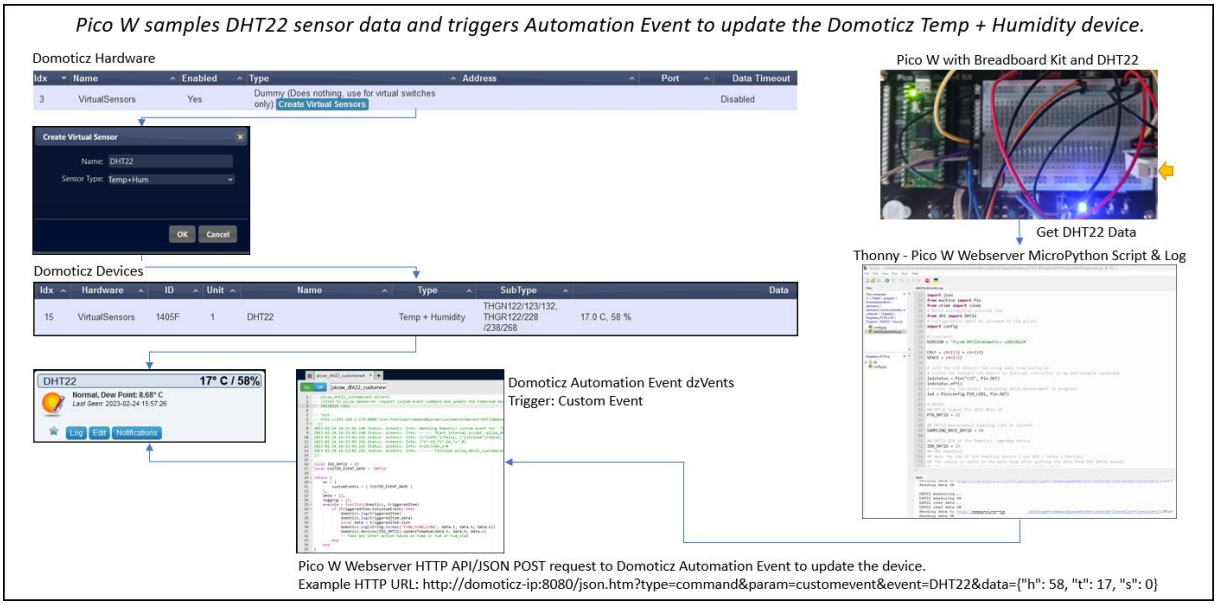
```
while True:

    # Measure DHT22 temperature & humidity & humidity status
    # Submit Domoticz HTTP API/JSON GET request to update the device
    network.send_get_request(URL_DOM_DHT22 + get_dht22_data())

    # Delay till next sample
    sleep(SAMPLING_RATE_DHT22)
```

# Domoticz Custom Event

The picture shows the flow of actions:

1. Create the Domoticz Temp+Hum device from the hardware “Dummy” (Create Virtual Sensors) ,
2. Run the Pico W web server and request every 60 seconds the DHT22 data,
3. After receiving data, the Pico W web server submits the data to Domoticz via HTTP API/JSON POST request with the command “customevent” and data as “JSON object”,
4. Domoticz handles the incoming HTTP request and updates the device.

## Block Diagram




## MicroPython Script

```
"""
File:    dht22_customevent.py
Date:    20230318
Author: Robert W.B. Linn

Read in regular intervals the DHT22 temperature and humidity and update the
Domoticz device named DHT22.
The Domoticz device is updated using HTTP API/JSON POST request Custom Event to the
Domoticz server.
The data is a JSON object: {"t":temperature NN,"h":humidity 0-100,"s":humidity
status 0-3}

:example
http://domoticz-
ip:8080/json.htm?type=command&param=customevent&event=DHT22&data={"t":19,"h":64,"s"
:0}

:notes
Pico Breadboard Kit is used to wire up the DHT22.
Pico Breadboard Kit LED1 is used as status LED when requesting DHT22 data and
updating domoticz.
Configuration stored in config.py, ensure to upload to the picow.
DHT22 measures every 60 seconds.
```

```
:log
DHT22_CustomEvent v20230311
Sampling Rate: 60s.
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
DHT22 t=19, h=43, hs=0, data={'h': 43, 't': 19, 's': 0}
Send POST request url=http://domoticz-
ip:port/json.htm?type=command&param=customevent&event=DHT22&data=, postdata={'h':
43, 't': 19, 's': 0}
Send POST request status=OK

:wiring
DHT22 = Pico W
VCC (+) = VBUS (Pin #40)
OUT = GP22 (Pin #29)
GND (-) = GND (Pin #28)
"""

# Imports
from machine import Pin
from utime import sleep
# Convert the Domoticz HTTP API/JSON response
import json
# DHT22 micropython internal lib
from dht import DHT22
# Call server from server.py (must be uploaded to the picow)
from server import Server
# Configuration (must be uploaded to the picow)
import config

# Constants
VERSION = 'DHT22_CustomEvent v20230311'

# Create the led object indicating dht22 measurement in progress
led1 = Pin(config.PIN_LED1, Pin.OUT)
led1.value(0)

# DHT22 Signal Pin GP22 #Pin 29
PIN_DHT22 = 22
# DHT22 measurement sampling rate in seconds
SAMPLING_RATE_DHT22 = 60
# DHT22 IDX of the Domoticz Temp+Hum device
IDX_DHT22 = 15
# URL Domoticz
# Note the idx of the domoticz device ( see GUI > Setup > Devices)
# The data is a JSON object, i.e. {"t":19,"h":64,"s":0}
URL_DOM_DHT22 = "http://domoticz-
ip:port/json.htm?type=command&param=customevent&event=DHT22&data="

# Create the dht22 sensor object
dht22_sensor = DHT22(Pin(Pin_DHT22, Pin.IN, Pin.PULL_UP))

"""
Set the humidity status level used for Domoticz HUM_STAT value.

:param int hum
    0: NORMAL, 1: COMFORTABLE, 2: DRY, 3: WET

:return int level
    Humidity level 0 - 3
"""
def set_humidity_status(hum, temp):
    level = 9
    # 2 = Dry
    if hum <= 30:
        level = 2
    # 3 = Wet
```

```
    elif hum >= 70:
        level = 3
    # 1 = Comfortable
    elif hum >= 35 and hum <= 65 and temp >=22 and temp <= 26:
        level = 1
    # 0 = Normal
    else:
        level = 0
    return level

"""
DHT22 measurement with roundes values for temperature and humidity.
During measurement, LED1 of the Pico Breadboard is on.

:return string data
    JSON object with key:value pairs t=temperature (°C), h=humidity (0-100%),
s=humidity_status (0-3)

:example
    {'h': 48, 't': 18, 's': 0}
"""
def get_dht22_data():
    led1.value(1)
    sleep(1)
    # print(f'DHT22 measuring...')
    dht22_sensor.measure()
    # print(f'DHT22 measuring OK')

    # print(f'DHT22 read data...')
    # Assign the data (rounded)
    temperature     = round(dht22_sensor.temperature())
    humidity        = round(dht22_sensor.humidity())
    humidity_status = set_humidity_status(humidity, temperature)

    # Set the data JSON object: {"t":NN,"h":NN,"s":N}
    data = {}
    data['t'] = temperature
    data['h'] = humidity
    data['s'] = humidity_status
    led1.value(0)
    print(f'DHT22 t={temperature}, h={humidity}, hs={humidity_status},
data={data}')
    return data

# Info
print(f'{VERSION}')
print(f'Sampling Rate: {SAMPLING_RATE_DHT22}s.')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()

# Main
# Measure DHT22 every NN seconds (see constant SAMPLING_DELAY)
while True:

    # Measure DHT22 temperature & humidity & humidity status
    # Submit Domoticz HTTP API/JSON POST request to update the device
    network.send_post_request(URL_DOM_DHT22, get_dht22_data())

    # Delay till next sample
    sleep(SAMPLING_RATE_DHT22)
```

## Automation Script

```
--[[
File:   dht22_customevent.dzvents
Date:   20230225
Author: Robert W.B. Linn

Listen to Pico W web server request custom event command and update the temp+hum
device temp,hum,hum_stat.

Test
http://domoticz-
ip:port/json.htm?type=command&param=customevent&event=DHT22&data={"t":19,"h":64,"s"
:0}
Log
2023-02-24 16:33:03.190 Status: dzVents: Info: Handling Domoticz custom event for:
"DHT22"
2023-02-24 16:33:03.190 Status: dzVents: Info: ------ Start internal script:
picow_dht22_customevent: Custom event: "DHT22"
2023-02-24 16:33:03.191 Status: dzVents: Info: {["isXML"]=false,
["isSystem"]=false, ["customEvent"]="DHT22", ["data"]="{"t":19,"h":64,"s":0}",
["isShellCommandResponse"]=false, ["type"]="customEvent", ["isHTTPResponse"]=false,
["json"]={["h"]=64, ["t"]=19, ["s"]=0}, ["isCustomEvent"]=true, ["message"]="",
["isGroup"]=false, ["isHardware"]=false, ["status"]="info", ["isDevice"]=false,
["trigger"]="DHT22", ["isSecurity"]=false, ["baseType"]="custom",
["dump"]=function, ["isTimer"]=false, ["isScene"]=false, ["isVariable"]=false,
["isJSON"]=true}
2023-02-24 16:33:03.191 Status: dzVents: Info: {"t":19,"h":64,"s":0}
2023-02-24 16:33:03.192 Status: dzVents: Info: t=19,h=64,s=0
2023-02-24 16:33:03.192 Status: dzVents: Info: ------ Finished
picow_dht22_customevent
]]--

local IDX_DHT22 = 15
-- Custom event name as used by the Pico W web server HTTP API/JSON POST request
local CUSTOM_EVENT_NAME = 'DHT22'

return {
 on = {
  customEvents = { CUSTOM_EVENT_NAME }
 },
 data = {},
 logging = {},
 execute = function(domoticz, triggeredItem)
  if (triggeredItem.isCustomEvent) then
   domoticz.log(triggeredItem)
            -- Check the custom event name in case there are more custom events
      if (triggeredItem.trigger == CUSTOM_EVENT_NAME) then
       domoticz.log(triggeredItem.data)
       local data = triggeredItem.json
       domoticz.log(string.format('t=%d,h=%d,s=%d', data.t, data.h, data.s))
       domoticz.devices(IDX_DHT22).updateTempHum(data.t, data.h, data.s)
       -- Take any other action based on temp or hum or hum_stat
      end
  end
 end
}
```

# DHT22 Temperature & Humidity MQTT Autodiscover

## Description

This project auto creates two Domoticz devices for Temperature & Humidity and regular updates the devices state.

## Solution

This project is based up the project DHT22 Temperature & Humidity and uses this Home Assistant Example (thanks for sharing).

Instead communicating via HTTP, the Domoticz hardware controller MQTT Auto Discovery Client Gateway with LAN interface (MQTT Autodiscover) is used.

The Pico W connects to the network and to the Domoticz Client Gateways for MQTT & MQTT Auto Discovery.
The config & state messages published by the Pico W are handled by the Domoticz hardware controller MQTT Auto Discovery Client Gateway with LAN interface.

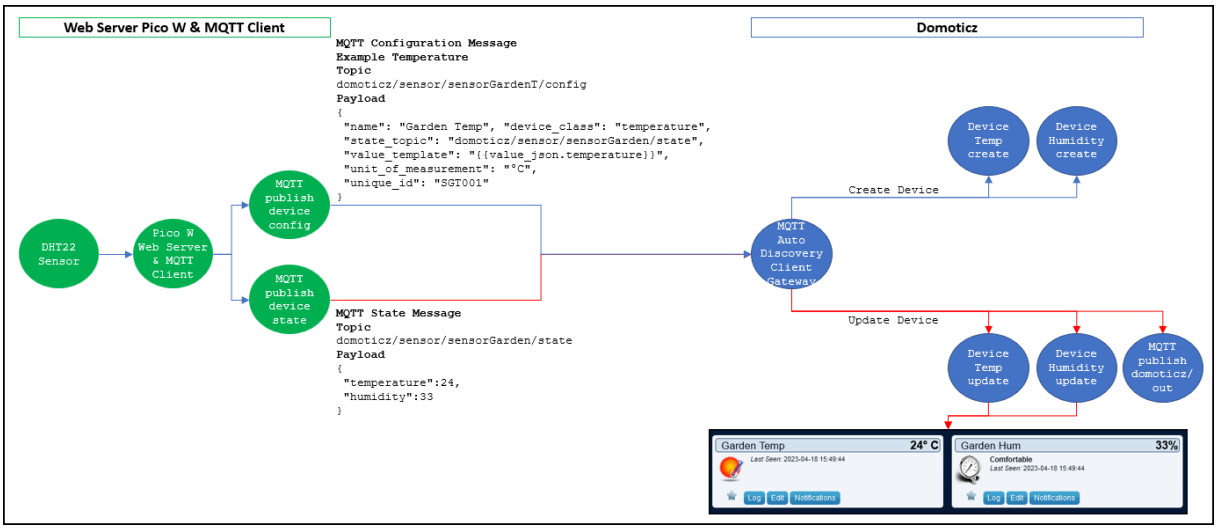
**Step 1** – Publish MQTT Config messages to create the two Domoticz devices for Temperature & Humidity.
If these devices are already in place, Domoticz takes no action.

The Domoticz devices list with the two devices auto created for the MQTT Autodiscover hardware (named MQTTADGateway):

| | Idx | Hardware | ID | Unit | Name | Type | SubType | |
|---|---|---|---|---|---|---|---|---|
| | 38 | MQTTADGateway | SGT001 | 1 | Garden Temp | Temp | THR128/138, THC138 | 19.0 C |
| | 37 | MQTTADGateway | SGH001 | 1 | Garden Hum | Humidity | LaCrosse WS2300 | Humidity 39 % |

**Step 2** – Publish MQTT State message after reading the DHT22 temperature and humidity.
The MQTT messages published are picked up by Domoticz and the value of the two devices are updated.
See block diagram showing the two device widgets Garden Temp & Garden Hum.

# Block Diagram




# MQTT Autodiscover Topics & Payload

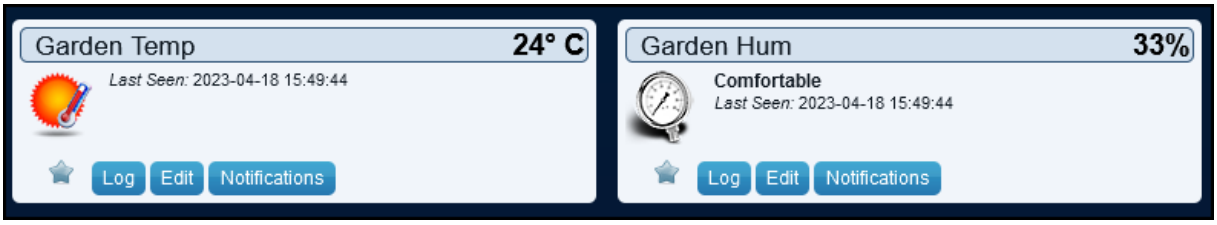
| | |
|---|---|
| **CONFIGURATION** | |
| **Temperature Device** | |
| Configuration Topic | domoticz/sensor/sensorGardenT/config |
| Configuration Payload | {<br>"name": "Garden Temp",<br>"device_class": "temperature",<br>"state_topic": "domoticz/sensor/sensorGarden/state",<br>"value_template": "{{value_json.temperature}}",<br>"unit_of_measurement": "°C",<br>"unique_id": "SGT001"<br>} |
| **Humidity Device** | |
| Configuration Topic | domoticz/sensor/sensorGardenH/config |
| Configuration Payload | {<br>"name": "Garden Hum",<br>"device_class": "humidity",<br>"state_topic": "domoticz/sensor/sensorGarden/state",<br>"value_template": "{{value_json.humidity}}",<br>"unit_of_measurement": "%",<br>"unique_id": "SGH001"<br>} |
| | |
| **STATE** | |
| Topic | domoticz/sensor/sensorGarden/state |
| Payload<br>Temperature &<br>Humidity | {<br>"temperature":{T},<br>"humidity":{H}<br>} |
| Payload<br>Temperature | {<br>"temperature":{T},<br>} |

| | |
|---|---|
| Payload<br>Humidity | {<br>"humidity":{H},<br>} |
| | **Note**<br>{T} and {H} are placeholders for the values. |

# Domoticz

## Devices

The two devices widget created via MQTT Autodiscover:




## Debug Log

Domoticz Debug Log showing the MQTT Autodiscover gateway receiving the message with the payload containing the temperature & humidity keys with values.
The message is split into two messages for the temperature & humidity for the Domoticz topic “domoticz/out”.

```
2023-04-18 15:51:15.090 Debug: MQTTADGateway: topic:
domoticz/sensor/sensorGarden/state, message: {"temperature":24, "humidity":33}
2023-04-18 15:51:15.220 Debug: MQTTADGateway: topic: domoticz/out, message: {
2023-04-18 15:51:15.220 "Battery" : 255,
2023-04-18 15:51:15.220 "LastUpdate" : "2023-04-18 15:51:15",
2023-04-18 15:51:15.220 "RSSI" : 12,
2023-04-18 15:51:15.220 "description" : "",
2023-04-18 15:51:15.220 "dtype" : "Humidity",
2023-04-18 15:51:15.220 "hwid" : "6",
2023-04-18 15:51:15.220 "id" : "SGH001",
2023-04-18 15:51:15.220 "idx" : 37,
2023-04-18 15:51:15.220 "name" : "Garden Hum",
2023-04-18 15:51:15.220 "nvalue" : 33,
2023-04-18 15:51:15.220 "stype" : "LaCrosse WS2300",
2023-04-18 15:51:15.220 "svalue1" : "1",
2023-04-18 15:51:15.220 "unit" : 1
2023-04-18 15:51:15.220 }
2023-04-18 15:51:15.220
2023-04-18 15:51:15.321 Debug: MQTTADGateway: topic: domoticz/out, message: {
2023-04-18 15:51:15.321 "Battery" : 255,
2023-04-18 15:51:15.321 "LastUpdate" : "2023-04-18 15:51:15",
2023-04-18 15:51:15.321 "RSSI" : 12,
2023-04-18 15:51:15.321 "description" : "",
2023-04-18 15:51:15.321 "dtype" : "Temp",
2023-04-18 15:51:15.321 "hwid" : "6",
2023-04-18 15:51:15.321 "id" : "SGT001",
2023-04-18 15:51:15.321 "idx" : 38,
2023-04-18 15:51:15.321 "name" : "Garden Temp",
2023-04-18 15:51:15.321 "nvalue" : 0,
2023-04-18 15:51:15.321 "stype" : "THR128/138, THC138",
2023-04-18 15:51:15.321 "svalue1" : "24.00",
2023-04-18 15:51:15.321 "unit" : 1
2023-04-18 15:51:15.321 }
```

```
2023-04-18 15:51:15.321
```

## Automation Script

There is no automation script used, but … if want to capture device changes, this is a simple example.

```
--[[
File:   dht22_mqtt_ad.dzvents
Date:   20230424
Author: Robert W.B. Linn

:description
Listen to two device changes which has been created using MQTT Autodiscover.

:log
2023-04-24 10:53:30.121 MQTTADGateway: Humidity/LaCrosse WS2300 (Garden Hum)
2023-04-24 10:53:30.136 MQTTADGateway: Temp/THR128/138, THC138 (Garden Temp)
2023-04-24 10:53:30.215 Status: dzVents: Info: Handling events for: "Garden Hum",
value: "51"
2023-04-24 10:53:30.215 Status: dzVents: Info: DHT22_MQTT_AD: ------ Start internal
script: dht22_mqtt_ad: Device: "Garden Hum (MQTTADGateway)", Index: 37
2023-04-24 10:53:30.215 Status: dzVents: Info: DHT22_MQTT_AD: Device changed:
name=Garden Hum, unique_id=SGH001, state=51
2023-04-24 10:53:30.215 Status: dzVents: Info: DHT22_MQTT_AD: ------ Finished
dht22_mqtt_ad
2023-04-24 10:53:30.268 Status: dzVents: Info: Handling events for: "Garden Temp",
value: "18.00"
2023-04-24 10:53:30.268 Status: dzVents: Info: DHT22_MQTT_AD: ------ Start internal
script: dht22_mqtt_ad: Device: "Garden Temp (MQTTADGateway)", Index: 38
2023-04-24 10:53:30.268 Status: dzVents: Info: DHT22_MQTT_AD: Device changed:
name=Garden Temp, unique_id=SGT001, state=18.00
2023-04-24 10:53:30.268 Status: dzVents: Info: DHT22_MQTT_AD: ------ Finished
dht22_mqtt_ad
2023-04-24 10:53:30.117 Debug: MQTTADGateway: topic:
domoticz/sensor/sensorGarden/state, message: {"temperature":18, "humidity":51}
]]--

-- IDX of the devices Temperature & Humidity
local IDX_DHT22_T = 38
local IDX_DHT22_H = 37

return {
 on = { devices = { IDX_DHT22_T, IDX_DHT22_H } },
 logging = { level = domoticz.LOG_INFO, marker = 'DHT22_MQTT_AD', },
 execute = function(domoticz, device)
  domoticz.log(string.format('Device changed: name=%s, unique_id=%s, state=%s',
      device.name, device.deviceId, device.state))
     -- Device changed: name=Garden Temp, unique_id=SGT001, state=21.00
 end
}
```

# Web Server

## Libraries

The MicroPython script uses the MicroPython library Lightweight MQTT client for MicroPython “umqtt.simple”.

**Credits**
Thanks for developing & sharing the MicroPython library micropython-umqqt.simple.

## MicroPython Script

```
"""
File:   dht22_mqtt_ad.py
Date:   20230424
Author: Robert W.B. Linn

:description
Test MQTT auto discovery with Domoticz and the Pico W running as web server.
Domoticz Hardware: MQTT Auto Discovery Client Gateway with LAN Interface,
Name=MQTTADGateway
Domoticz Devices created via MQTT auto discovery by publishing config topic (see
domoticz log below).
The two devices are from type=Temp and type=Humidity.
Note that this script does not use the library server.py but has simple code to
connect to the network.

:external libraries
umqtt.simple

MQTT Remove Retained messages:
mosquitto_sub -h localhost --remove-retained -t '#' -W 1

:domoticz log
NOTE:
Initially the two devices for temperature & humidity are not in place.
These are created first time by the MQTT topics
"domoticz/sensor/sensorGardenT/config" and "domoticz/sensor/sensorGardenH/config".
The state, i.e. the two device value updates is set by the topic
"domoticz/sensor/sensorGarden/state".
See below log for payload examples for the topics.

2023-04-16 14:30:33.183 Debug: : MQTT PublishSchema 1 (1), Retain 0
2023-04-16 14:30:33.183 Debug: : MQTT PublishSchema 0 (0), Retain 0
2023-04-16 14:30:36.702 Debug: MQTTADGateway: topic: domoticz/status, message:
online
2023-04-16 14:30:52.422 Debug: MQTTADGateway: topic:
domoticz/sensor/sensorGardenT/config, message: {"name": "Garden Temp",
"device_class": "temperature", "state_topic": "domoticz/sensor/sensorGarden/state",
"value_template": "{{value_json.temperature}}", "unit_of_measurement": "°C",
"unique_id": "SGT001"}
2023-04-16 14:30:52.523 Debug: MQTTADGateway: topic:
domoticz/sensor/sensorGardenH/config, message: {"name": "Garden Hum",
"device_class": "humidity", "state_topic": "domoticz/sensor/sensorGarden/state",
"value_template": "{{value_json.humidity}}", "unit_of_measurement": "%",
"unique_id": "SGH001"}
2023-04-16 14:30:53.704 Debug: MQTTADGateway: topic:
domoticz/sensor/sensorGarden/state, message: {"temperature":21, "humidity":45}
2023-04-16 14:31:04.907 Debug: MQTTADGateway: topic:
domoticz/sensor/sensorGarden/state, message: {"temperature":21, "humidity":45}

:thonny log
dht22_mqtt_ad v20230420
Sampling Rate: 10s.
Network connecting...
```

```
Network connected: picow-ip
MQTT Broker connecting...
MQTT Broker connected: domoticz-ip
MQTT Autodiscover published: topic=domoticz/sensor/sensorGardenT/config,
payload={"name": "Garden Temp", "device_class": "temperature", "state_topic":
"domoticz/sensor/sensorGarden/state", "value_template":
"{{value_json.temperature}}", "unit_of_measurement": "°C", "unique_id": "SGT001"}
MQTT Autodiscover published: topic=domoticz/sensor/sensorGardenH/config,
payload={"name": "Garden Hum", "device_class": "humidity", "state_topic":
"domoticz/sensor/sensorGarden/state", "value_template": "{{value_json.humidity}}",
"unit_of_measurement": "%", "unique_id": "SGH001"}
MQTT published: topic=domoticz/sensor/sensorGarden/state,
payload={"temperature":21, "humidity":37}
MQTT published: topic=domoticz/sensor/sensorGarden/state,
payload={"temperature":21, "humidity":37}

:wiring
DHT22 = Pico W
VCC (+) = VBUS (Pin #40)
OUT = GP22 (Pin #29)
GND (-) = GND (Pin #28)
"""

# Imports
import network
import time
from machine import Pin
from utime import sleep, sleep_ms
# Installed via Thonny manage packages
from umqtt.simple import MQTTClient
# DHT22 micropython internal lib
from dht import DHT22
# Call server from server.py (must be uploaded to the picow)
from server import Server
# Configuration (must be uploaded to the pico w)
import config
import sys

# Constants
VERSION = const('dht22_mqtt_ad v20230420')

"""
DHT22
"""
# DHT22 Signal GP22 (Pin #29)
PIN_DHT22 = const(22)
# DHT22 measurement sampling rate in seconds
SAMPLING_RATE = const(60)

# Create the DHT22 sensor object
dht22_sensor = DHT22(Pin(PIN_DHT22, Pin.IN, Pin.PULL_UP))

# DHT22 measurement with roundes values for temperature and humidity.
# return int temperature, int humidity
def get_dht22_data():
    # print(f'DHT22 measuring...')
    dht22_sensor.measure()
    # print(f'DHT22 measuring OK')

    # print(f'DHT22 read data...')
    # Assign the data (rounded)
    temperature     = round(dht22_sensor.temperature())
    humidity        = round(dht22_sensor.humidity())

    # Create the payload for the STATE_TOPIC (see below)
    payload = STATE_PAYLOAD
    payload = payload.replace('{T}', str(temperature))
    payload = payload.replace('{H}', str(humidity))
```

```
    # print(f'DHT22 t={temperature}, h={humidity}')
    return payload.encode()

"""
MQTT
"""
MQTT_BROKER  = const('192.168.1.179')
MQTT_CLIENT_ID = const('picow_dht22')

# Topics & payload to create the devices. The configuration component is sensor.
TEMP_TOPIC_CONFIG= const(b'domoticz/sensor/sensorGardenT/config')
TEMP_PAYLOAD_CONFIG = const(b'{"name": "Garden Temp", "device_class": "temperature",
"state_topic": "domoticz/sensor/sensorGarden/state", "value_template":
"{{value_json.temperature}}", "unit_of_measurement": "°C", "unique_id": "SGT001"}')
HUM_TOPIC_CONFIG = const(b'domoticz/sensor/sensorGardenH/config')
HUM_PAYLOAD_CONFIG= const(b'{"name": "Garden Hum", "device_class": "humidity",
"state_topic": "domoticz/sensor/sensorGarden/state", "value_template":
"{{value_json.humidity}}", "unit_of_measurement": "%", "unique_id": "SGH001"}')

# Topic used to publish the state of the two sensors
# Payload is JSON format: {"temperature":21, "humidity":45}
STATE_TOPIC  = const(b'domoticz/sensor/sensorGarden/state')
STATE_PAYLOAD   = '{"temperature":{T}, "humidity":{H}}'

# Connect to the MQTT broker with client id and server address
def mqtt_connect(client_id, mqtt_server):
    client = MQTTClient(client_id, mqtt_server, keepalive=3600)
    client.connect()
    print(f'MQTT Broker connecting...')
    return client

# Reconnect to the MQTT broker
def mqtt_reconnect():
    print('[ERROR] Failed to connect to the MQTT Broker. Reconnecting...')
    time.sleep(5)
    machine.reset()

# Info
print(f'{VERSION}')
print(f'Sampling Rate: {SAMPLING_RATE}s.')

# Connect to the network
print(f'Network connecting...')
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(config.WIFI_SSID, config.WIFI_PASSWORD)
time.sleep(5)
if not wlan.isconnected():
    raise RuntimeError('[ERROR] Can not connect to the network.')
print(f'Network connected: {wlan.ifconfig()[0]}')

# Connect to MQTT
# Publish mqtt auto discovery topic & payload for the temperature & humidity device
try:
    # Connect to MQTT broker
    client = mqtt_connect(MQTT_CLIENT_ID, MQTT_BROKER)
    print(f'MQTT Broker connected: {MQTT_BROKER}')

    # Publish config topics to auto create the two devices in domoticz
    client.publish(TEMP_TOPIC_CONFIG, TEMP_PAYLOAD_CONFIG)
    print(f'MQTT Autodiscover published: topic={TEMP_TOPIC_CONFIG.decode()},
payload={TEMP_PAYLOAD_CONFIG.decode()}')

    client.publish(HUM_TOPIC_CONFIG, HUM_PAYLOAD_CONFIG)
    print(f'MQTT Autodiscover published: topic={HUM_TOPIC_CONFIG.decode()},
payload={HUM_PAYLOAD_CONFIG.decode()}')

except OSError as e:
    mqtt_reconnect()
```

```
# Main
while True:
    # Read the sensor data to get the payload {"temperature":19, "humidity":39}
    payload = get_dht22_data()

    # MQTT publish with encoded topic & payload (buffered objects required)
    client.publish(STATE_TOPIC, payload)
    print(f'MQTT published: topic={STATE_TOPIC.decode()},
payload={payload.decode()}')

    # Delay till next sample
    sleep(SAMPLING_RATE)
```

# LCD 20x4 I2C LED Control

## Description

This project switches, via Domoticz, LED1 of the Pico Breadboard Kit On/Off and display the LED state on an LCD 20x4 I2C display (LCD2004) connected to the Pico W.

This project shows how to set text on the LCD2004. It will be used by other projects.

**Ideas for Use**

- Domoticz Mini Information display,
- Domoticz Motherboard Monitor,
- Weather Station.

**Screenshot LED1 On displayed on the LCD2004.**


## Solution

The Pico W has an LCD 20x4 I2C display connected and runs as a web server.
A Domoticz Switch state change triggers an Domoticz Automation Event dzVents.
This event listens to the switch device state changes and submits an HTTP POST request to the Pico W web server.

The POST data received by the Pico W is a JSON object:

```
{"state": "on" or "off"}
```

The LED1 is set according to the given state on or off.
The LCD2004 displays “LED1: On OK” on line 3 (row index 2, col index 0).

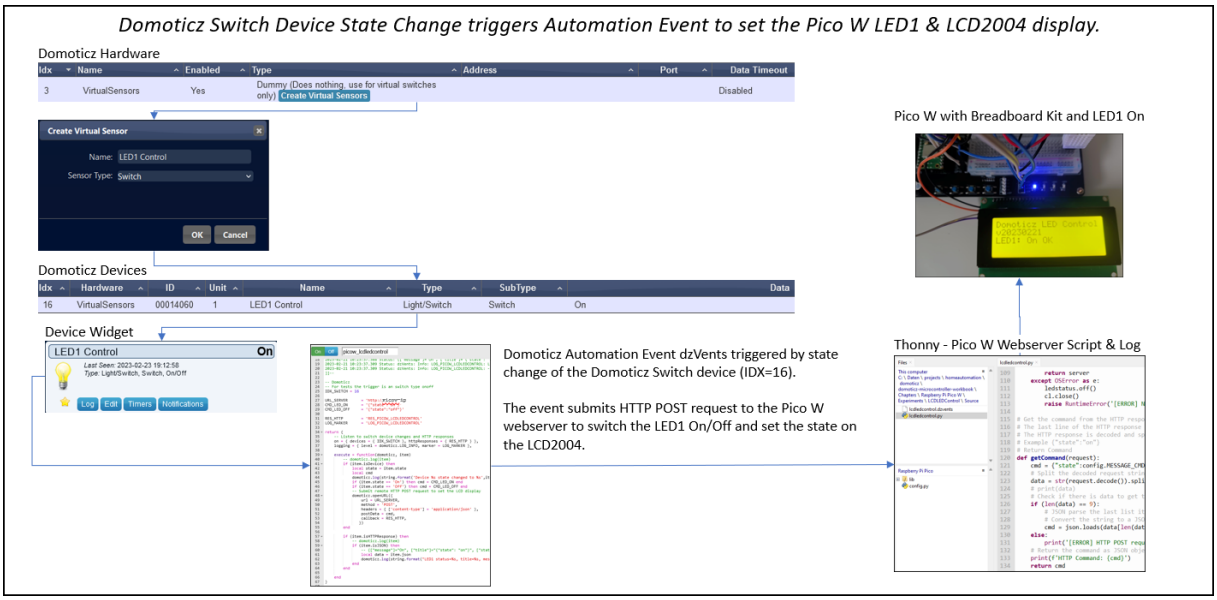
The Pico W web server sends an HTTP response back to Domoticz (as the client).

The HTTP response received by Domoticz, is a JSON object with key:value pairs parsed by the Domoticz event.

```
{["message"]="On", ["title"]="{"state": "on"}", ["status"]="OK"}
```

- message - LED1 state On or Off,
- title - Command received from the Domoticz Automation Event,
- status - OK, means the command has been executed successfully.

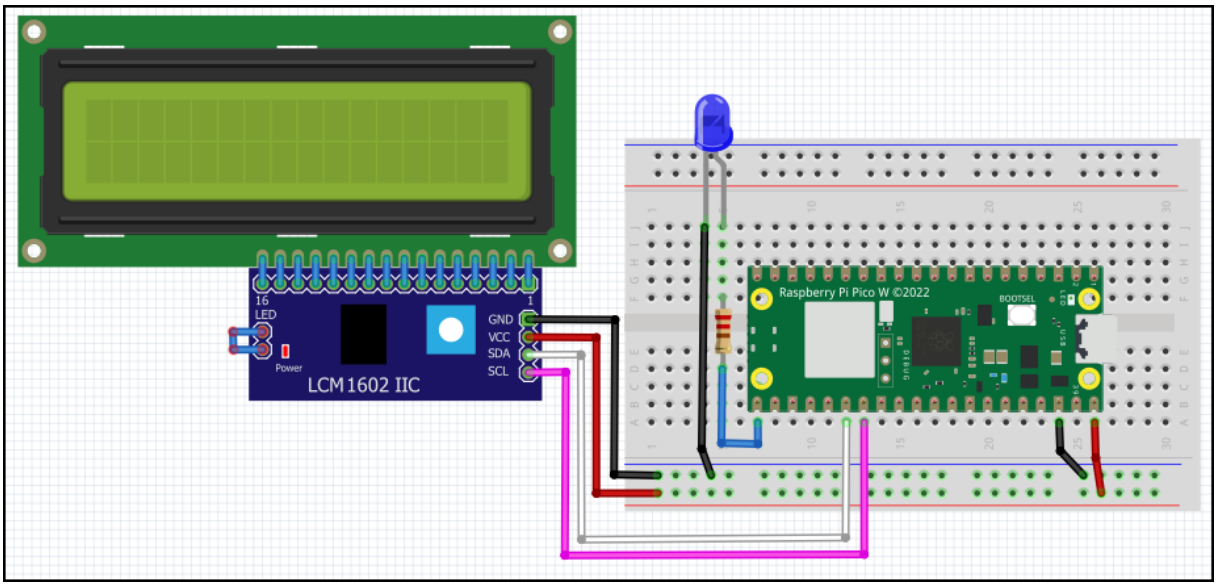
# Block Diagram



# Wiring

| **LCD 2004 I2C** | **Pico W** |
|---|---|
| VCC | VBUS (5V) |
| SDA | GP20 (Pin #26) |
| SCL | GP21 (Pin #27) |
| GND | GND (Pin #38) |
| I2C Address | 0x27 |
| | |
| **LED (blue)** | **Pico W** |
| + (Anode) | GP16 (Pin #21) |
| GND (Cathode) | GND (Pin #38) |

# Circuit Diagram




*Note*

The circuit shows an LCD 16x2 I2C instead the used LCD 20x4 I2C.
This solution can also be used for an LCD 16x2 I2C - need to change parameter in the Pico W web server script:

```
# LCD1602 Constants
LCD_I2C_ADDRESS = 0x27
LCD_ROWS = 2
LCD_COLS = 16
```

*Note*

The I2C address must be checked via the scan function. See MicroPython source code below.

# Domoticz Setup

## Devices

Create a virtual sensor, hardware dummy, named LED1 Control from sensor type Switch/Light.

After creating the device, the Domoticz devices list shows the entry:

```
Idx=16, Hardware=VirtualSensors, ID=00014060, Unit=1, Name=LED1 Control,
Type=Light/Switch, SubType=Switch, Data=On
```

## Automation Script

The event listens to the Domoticz switch device state change On or Off.
Depending on the switch state, the HTTP POST JSON object is defined.
The HTTP POST request is sent to the Pico W web server.
The HTTP response from the Pico W web server is logged in the Domoticz log.

```
--[[
File:   lcdledcontrol.dzvents
Date:   20230221
Author: Robert W.B. Linn

Switch the Pico W LED1 (of the Breadboard Kit) and display on LCD2004.
The Pico W runs a RESTful web server.
The HTTP POST request to the Pico W web server is a JSON object: {"state":"on" or
"off"}.
The Pico W web server HTTP response is a JSON object with key:value pairs:
{["message"]="On", ["title"]="{"state": "on"}", ["status"]="OK"}

Domoticz Log
2023-02-21 10:23:36.299 Status: dzVents: Info: Handling events for: "Pico W-
Control", value: "On"
2023-02-21 10:23:36.300 Status: dzVents: Info: LOG_PICOW_LCDLEDCONTROL: ------
Start internal script: picow_lcdledcontrol: Device: "Pico W-Control
(VirtualSensors)", Index: 16
2023-02-21 10:23:36.300 Status: dzVents: Info: LOG_PICOW_LCDLEDCONTROL: Device Pico
W-Control state changed to On
2023-02-21 10:23:36.300 Status: dzVents: Info: LOG_PICOW_LCDLEDCONTROL: ------
Finished picow_lcdledcontrol
2023-02-21 10:23:36.300 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
2023-02-21 10:23:37.308 Status: dzVents: Info: Handling httpResponse-events for:
"RES_PICOW_LCDLEDCONTROL"
2023-02-21 10:23:37.308 Status: dzVents: Info: LOG_PICOW_LCDLEDCONTROL: ------
Start internal script: picow_lcdledcontrol: HTTPResponse: "RES_PICOW_LCDLEDCONTROL"
2023-02-21 10:23:37.308 Status: {["message"]="On", ["title"]="{"state": "on"}",
["status"]="OK"}
2023-02-21 10:23:37.309 Status: dzVents: Info: LOG_PICOW_LCDLEDCONTROL: LED1
status=OK, title={"state": "on"}, message=On
2023-02-21 10:23:37.309 Status: dzVents: Info: LOG_PICOW_LCDLEDCONTROL: ------
Finished picow_lcdledcontrol
]]--

-- Domoticz
-- For tests the trigger is an switch type onoff
local IDX_SWITCH = 16

local URL_SERVER      = 'http://picow-ip'
local CMD_LED_ON      = '{"state":"on"}'
local CMD_LED_OFF     = '{"state":"off"}'
```

```
local RES_HTTP        = 'RES_PICOW_LCDLEDCONTROL'
local LOG_MARKER      = 'LOG_PICOW_LCDLEDCONTROL'

return {
    -- Listen to switch device changes and HTTP responses
 on = { devices = { IDX_SWITCH }, httpResponses = { RES_HTTP } },
 logging = { level = domoticz.LOG_INFO, marker = LOG_MARKER },

 execute = function(domoticz, item)
       -- domoticz.log(item)
  if (item.isDevice) then
           local state = item.state
           local cmd
      domoticz.log(string.format('Device %s state changed to %s',item.name,state),
domoticz.LOG_INFO)
           if (item.state == 'On') then cmd = CMD_LED_ON end
           if (item.state == 'Off') then cmd = CMD_LED_OFF end
           -- Submit remote HTTP POST request to set the LCD display
           domoticz.openURL({
               url = URL_SERVER,
               method = 'POST',
               headers = { ['content-type'] = 'application/json' },
               postData = cmd,
               callback = RES_HTTP,
              })
       end

       if (item.isHTTPResponse) then
           -- domoticz.log(item)
           if (item.isJSON) then
               -- {["message"]="On", ["title"]="{"state": "on"}", ["status"]="OK"}
               local data = item.json
               domoticz.log(string.format("LED1 status=%s, title=%s, message=%s",
data.status, data.title, data.message))
           end
       end

 end
}
```

# Web Server

## Libraries

The MicroPython script uses the two external libraries *lcd_api.py* and *machine_i2c_lcd.py* to control the LCD2004.
Enhanced is the library *lcd_api.py* with additional functions, like setting text at position col:row and more.
The two libraries are stored on the Pico W in folder lib (see sub chapter Additional Libraries).

**Credits**
Thanks for developing & sharing the libraries lcd_api and machine_i2c_lcd.

## MicroPython Script

```
"""
File: lcdledcontrol.py
Date: 20230318
Author: Robert W.B. Linn

Pico W RESTful web server listening for data from Domoticz event.
The incoming data is from a HTTP POST request with JSON object to switch LED1 on or
off.
LED1 is attached on the Pico Breadboard kit.

:log
Example turning LED1 on from Domoticz:
HTTP Command: {'state': 'on'}
HTTP Response: {"status": "OK", "title": {"state": "on"}, "message": "On"}

:wiring
LCD2004 = Pico W
VCC = VBUS (5V) (red)
SDA = GP20 (Pin #26) (white)
SCL = GP21 (Pin #27) (Pink)
GND = GND (black)
"""

# Libraries
from time import sleep
from machine import Pin
import json
# Call server from server.py (must be uploaded to the picow)
from server import Server
# LCD Display libs stored in Pico W folder lib
# lcd_api.py, machine_i2c_lcd.py
from machine import I2C, Pin
from machine_i2c_lcd import I2cLcd
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
NAME = 'Domoticz LED Control'
VERSION = 'v20230311'

CRLF = chr(13) + chr(10)
SPACE = chr(32)

# Create the LED1 (blue) object using config.py settings
led1 = Pin(config.PIN_LED1, Pin.OUT)
led1.off()

# LCD2004 Constants
LCD_I2C_ADDRESS = 0x27
```

```
LCD_PIN_SDA = 20
LCD_PIN_SCL = 21
LCD_ROWS = 4
LCD_COLS = 20

def set_lcd(address, pinsda, pinscl, rows, cols):
    """
    Create the LCD object by init the lcd with i2c.

    :param hex address
        Address of the LCD I2C. Default 0x27

    :param int pinsda
        SDA pin

    :param int pinscl
        SCL pin

    :param int rows
        Number of rows 20 or 16

    :param int cols
        Number of cols 4 or 2

    :return
        LCD object

    :example
    set_lcd(LCD_I2C_ADDRESS, LCD_PIN_SDA, LCD_PIN_SCL, LCD_ROWS, LCD_COLS)
    """
    try:
        # I2C object
        i2c = I2C(0, sda=Pin(Pinsda), scl=Pin(Pinscl), freq=100000)
        # Init LCD object using I2C with address & rows (4, index 0-3) & cols (20,
index 0-19)
        lcd = I2cLcd(i2c, address, rows, cols)
        print("LCD init. Address: " + str(i2c.scan()))
        return lcd
    except OSError as e:
        raise RuntimeError('[ERROR] LCD init.')

def set_lcd_welcome(row1, row2):
    """
    LCD Initial text on row 1 & 2
    """
    lcd.putstr(row1 + "\n" + row2)
    sleep(.3)

def handle_request(cmd, status):
    """
    Handle the LCD command defined as JSON object.

    :param JSON object
        JSON object with key:value pair {"state":"on" or "off"}

    :status
        If status is 1 set the display else unknown command

    :return JSON object response
    """
    # Assign the command to the response title
    response[config.KEY_TITLE] = cmd

    if status == 1:
        # Select the command and set the lcd text
        if cmd['state'] == 'on':
            led1.on()
            response[config.KEY_MESSAGE] = config.MESSAGE_ON
            response[config.KEY_STATE] = config.STATE_OK
```

```
        elif cmd['state'] == 'off':
            led1.off()
            response[config.KEY_MESSAGE] = config.MESSAGE_OFF
            response[config.KEY_STATE] = config.STATE_OK
        else:
            response[config.KEY_STATE] = config.STATE_ERR
            response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN
    else:
            response[config.KEY_STATE] = config.STATE_ERR
            response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

    # LCD display set
    # Clear row 2 (the row range for LCD2004 is 0 to 3)
    lcd.clrrow(2)
    sleep(.3)

    # Write the keys message and state as string at col 0, row 2
    lcd.putstrat(0, 2, 'LED1: ' + response[config.KEY_MESSAGE] + ' ' +
response[config.KEY_STATE])
    sleep(.3)

    return response

# Main
# Listen for incoming connections from the Domoticz Automation Event dzVents
print(f'{NAME} {VERSION}')

# Create the LCD object
lcd = set_lcd(LCD_I2C_ADDRESS, LCD_PIN_SDA, LCD_PIN_SCL, LCD_ROWS, LCD_COLS)

# Set the LCD display welcome text
set_lcd_welcome(NAME, VERSION)

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()

"""
Main Loop
"""
while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Get the cmd as JSON object from the POST request
        # {"state":"on" or "off"}
        cmd, status = network.parse_post_request(request)

        # Create the HTTP response JSON object
        response = {}

        # Handle the command to update the LCD text.
        # Set the response
        response = handle_request(cmd, status)

        # Send the response to Domoticz and close the connection (wait for new)
        network.send_response(cl, response, True)

    except OSError as e:
        ledstatus.off()
        cl.close()
        print('[ERROR] Network Connection closed')
```

# LCD 20x4 I2C Motherboard Info

## Description

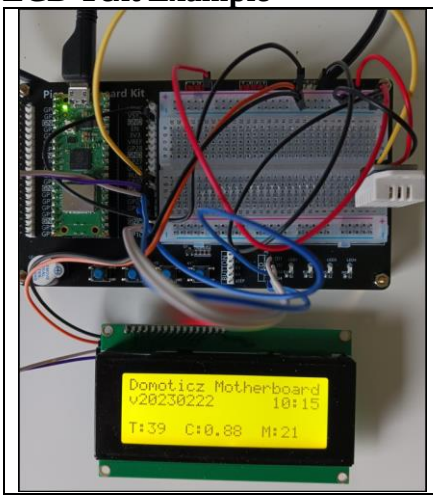
This project displays and updates in regular intervals selective Domoticz data from the hardware motherboard sensors on an LCD 20x4 I2C display (LCD2004) connected to the Pico W.

## Solution

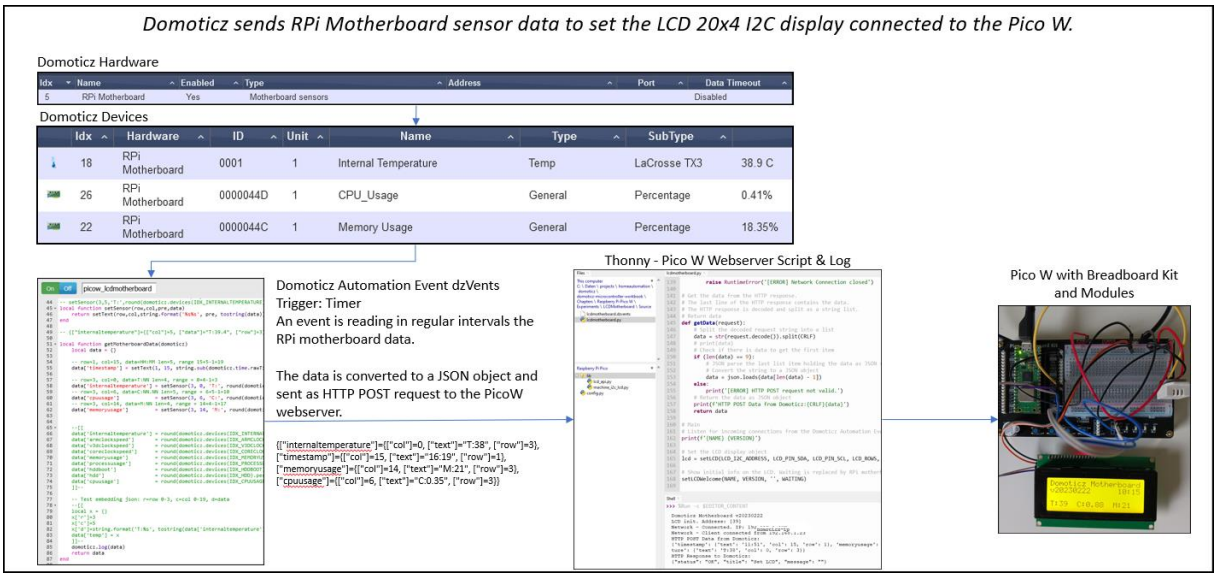
The LCD2004 setup is as previous described in the project LCD 20x4 I2C LED Control [Domoticz to Pico W].

Domoticz uses an Automation Event dzVents, triggered by a switch or timer, to send the selective Domoticz hardware motherboard sensors data via HTTP POST request to the Pico W web server. The data is a JSON array with items per LCD line.

**LCD Text Example**

| | |
|---|---|
|  | The motherboard sensors<br>T=internaltemperature<br>C=cpuusage<br>M=memoryusage |

# Block Diagram




# Wiring

See project LCD 20x4 I2C LED Control [Domoticz to Pico W].

# Circuit Diagram

See project LCD 20x4 I2C LED Control [Domoticz to Pico W].

# Domoticz Setup

## Devices

The hardware Motherboard sensors is added, and the devices Internal Temperature, CPU Usage and Memory Usage are added.
The other devices from the hardware Motherboard sensors are not used for this project.

After creating the device(s), the Domoticz devices list shows the entries:

```
IDX=26, Hardware=RPi Motherboard, ID=0000044D, Unit=1, Name=CPU Usage,
Type=General, SubType=Percentage, Data=0.43%
IDX=22, Hardware=RPi Motherboard, ID=0000044C, Unit=1, Name=Memory Usage,
Type=General, SubType=Percentage, Data=22.93%
IDX=18, Hardware=RPi Motherboard, ID=0001, Unit=1, Name=Internal Temperature,
Type=Temp, SubType=LaCrosse TX3, Data=38.4 C
```

## Automation Script

```
--[[
File:   lcdmotherboard.dzvents
Date:   20230221
Author  Robert W.B. Linn

Switch the Pico W LED1 (of the Breadboard Kit) and display on LCD2004.
The Pico W runs a RESTful web server.
The HTTP POST request to the Pico W web server is a JSON object: {"state":"on" or
"off"}.
The Pico W web server HTTP response is a JSON object with key:value pairs:
{["message"]="On", ["title"]="{"state": "on"}", ["status"]="OK"}

Log
2023-03-06 10:45:00.310 Status: dzVents: Info: LOG_PICOW_LCDMOTHERBOARD: ------
Start internal script: picow_lcdmotherboard:, trigger: "every minute"
2023-03-06 10:45:00.334 Status: dzVents: Info: LOG_PICOW_LCDMOTHERBOARD:
{["memoryusage"]={["row"]=3, ["text"]="M:20", ["col"]=14}, ["cpuusage"]={["row"]=3,
["text"]="C:0.37", ["col"]=6}, ["timestamp"]={["row"]=1, ["text"]="10:45",
["col"]=15}, ["internaltemperature"]={["row"]=3, ["text"]="T:44", ["col"]=0}}
2023-03-06 10:45:00.335 Status: dzVents: Info: LOG_PICOW_LCDMOTHERBOARD: ------
Finished picow_lcdmotherboard
2023-03-06 10:45:00.335 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
]]--

-- Domoticz
-- For tests the trigger is an switch type onoff
local IDX_SWITCH = 16
--- IDX of the motherboard sensors (devices)
local IDX_INTERNALTEMPERATURE = 18      -- temperature
local IDX_ARMCLOCKSPEED = 19            -- sensorValue
local IDX_V3DCLOCKSPEED = 20            -- sensorValue
local IDX_CORECLOCKSPEED = 21           -- sensorValue
local IDX_MEMORYUSAGE = 22              -- percentage
local IDX_PROCESSUSAGE = 23                     -- sensorValue
local IDX_HDDBOOT = 24                          -- percentage
local IDX_HDD = 25              -- percentage
local IDX_CPUUSAGE = 26                         -- percentage

-- Round a number with digital places
local function round(num, numDecimalPlaces)
  return tonumber(string.format("%." .. (numDecimalPlaces or 0) .. "f", num))
end

-- Create table with keys col, row and text to be displayed on the LCD.
-- For a LCd2004 row 0-3, col 0-19, text length max 20 characters
-- setText(3,0,'Hello World')
local function setText(row,col,text)
    local x = {}
    x['row']=row
    x['col']=col
    x['text']=text
 return x
end

-- Create table with sensor data for LCD display
-- setSensor(3,5,'T:',round(domoticz.devices(IDX_INTERNALTEMPERATURE).temperature,
2))
local function setSensor(row,col,pre,data)
    return setText(row,col,string.format('%s%s', pre, tostring(data)))
end

-- {["internaltemperature"]={["col"]=5, ["data"]="T:39.4", ["row"]=3},
["cpuusage"]=0.68, ["coreclockspeed"]=500, ["v3dclockspeed"]=250,
["processusage"]=45.06, ["armclockspeed"]=600, ["hdd"]=38.19, ["memoryusage"]=20.9,
["hddboot"]=19.68}
```

```
local function getMotherboardData(domoticz)
    local data = {}

    -- row=1, col=15, data=HH:MM len=5, range 15+5-1=19
 data['timestamp'] = setText(1, 15, string.sub(domoticz.time.rawTime, 1, 5))

    -- row=3, col=0, data=T:NN len=4, range = 0+4-1=3
    data['internaltemperature'] = setSensor(3, 0, 'T:',
round(domoticz.devices(IDX_INTERNALTEMPERATURE).temperature, 0))
    -- row=3, col=6, data=C:NN.NN len=5, range = 6+5-1=10
 data['cpuusage']            = setSensor(3, 6, 'C:',
round(domoticz.devices(IDX_CPUUSAGE).percentage, 2))
    -- row=3, col=14, data=M:NN len=4, range = 14+4-1=17
 data['memoryusage']         = setSensor(3, 14, 'M:',
round(domoticz.devices(IDX_MEMORYUSAGE).percentage, 0))


    --[[
    data['internaltemperature'] =
round(domoticz.devices(IDX_INTERNALTEMPERATURE).temperature, 2)
 data['armclockspeed']       =
round(domoticz.devices(IDX_ARMCLOCKSPEED).sensorValue, 0)
 data['v3dclockspeed']       =
round(domoticz.devices(IDX_V3DCLOCKSPEED).sensorValue, 0)
 data['coreclockspeed']      =
round(domoticz.devices(IDX_CORECLOCKSPEED).sensorValue, 0)
 data['memoryusage']         = round(domoticz.devices(IDX_MEMORYUSAGE).percentage,
2)
 data['processusage']        =
round(domoticz.devices(IDX_PROCESSUSAGE).sensorValue, 2)
 data['hddboot']             = round(domoticz.devices(IDX_HDDBOOT).percentage, 2)
 data['hdd']                 = round(domoticz.devices(IDX_HDD).percentage, 2)
 data['cpuusage']            = round(domoticz.devices(IDX_CPUUSAGE).percentage, 2)
    ]]--

 -- Test embedding json: r=row 0-3, c=col 0-19, d=data
    --[[
    local x = {}
    x['r']=3
    x['c']=5
    x['d']=string.format('T:%s', tostring(data['internaltemperature']))
 data['temp'] = x
    ]]--
 domoticz.log(data)
 return data
end

-- URL of the Pico W web server
local URL_SERVER      = 'http://picow-ip'

local PROJECT         = 'PICOW_LCDMOTHERBOARD'
local RES_HTTP        = 'RES_' .. PROJECT
local LOG_MARKER      = 'LOG_' .. PROJECT

local TIMER_RULE = 'every minute'

return {
    -- Listen to switch device changes and HTTP responses
 on = { devices = { IDX_SWITCH }, timer = { TIMER_RULE }, httpResponses = {
RES_HTTP } },
 logging = { level = domoticz.LOG_INFO, marker = LOG_MARKER },

 execute = function(domoticz, item)
        -- domoticz.log(item)
        if (item.isTimer) then
            domoticz.openURL({
                url = URL_SERVER,
                method = 'POST',
```

```
                headers = { ['content-type'] = 'application/json' },
                postData = getMotherboardData(domoticz),
                callback = RES_HTTP,
               })
        end

   if (item.isDevice) then
       domoticz.log(string.format('Device %s state changed to %s', item.name,
item.state), domoticz.LOG_INFO)
            if (item.state == 'On') then
                -- Submit remote HTTP POST request to set the LCD display
                domoticz.openURL({
                    url = URL_SERVER,
                    method = 'POST',
                    headers = { ['content-type'] = 'application/json' },
                    postData = getMotherboardData(domoticz),
                    callback = RES_HTTP,
                   })
            end
        end

        -- Handle HTTP response: OK is item statusCode 200 and item.ok true
        -- Else error like statusCode 7, item.ok false
        if (item.isHTTPResponse) then
            -- domoticz.log(string.format("%d %s", item.statusCode, item.ok))
            if (item.statusCode == 200) then
                if (item.isJSON) then
                    -- {["message"]="On", ["title"]="{"state": "on"}",
["status"]="OK"}
                    local data = item.json
                    domoticz.log(data)
                    -- domoticz.log(string.format("LED1 status=%s, title=%s,
message=%s", data.status, data.title, data.message))
                end
            else
                -- Error like 7 false; ERROR 7:Couldn't connect to server
                domoticz.log(string.format("ERROR %d:%s", item.statusCode,
item.statusText))
            end
        end
 end
}
```

# Web Server

## Libraries

See project LCD 20x4 I2C LED Control [Domoticz to Pico W].

## MicroPython Script

```
"""
File: lcdmotherboard.py
Date: 20230318
Author: Robert W.B. Linn

On an LCD2004 connected to the Pico W, display text and selective RPi motherboard
sensor data received from Domoticz.
The Pico W runs a RESTful web server handling incoming data from a Domoticz
Automation event dzVents.
The incoming data is received from a HTTP POST request with JSON object to set the
text.
The JSON object contains for each of the displayed text, the col, row and text.
{'sensor': {'text': 'TEXT', 'col': NN, 'row': N}, ...}
This enables to set the LCD display layout from the Domoticz event.

:log
Domoticz Motherboard v20230311
LCD init. Address: [39]
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Network client connected from client-ip
HTTP Command={'timestamp': {'text': '16:00', 'col': 15, 'row': 1}, 'memoryusage':
{'text': 'M:20', 'col': 14, 'row': 3}, 'cpuusage': {'text': 'C:0.3', 'col': 6,
'row': 3}, 'internaltemperature': {'text': 'T:41', 'col': 0, 'row': 3}}
HTTP Response={"status": "OK", "title": {"timestamp": {"text": "16:00", "col": 15,
"row": 1}, "memoryusage": {"text": "M:20", "col": 14, "row": 3}, "cpuusage":
{"text": "C:0.3", "col": 6, "row": 3}, "internaltemperature": {"text": "T:41",
"col": 0, "row": 3}}, "message": ""}
Network connection closed

:wiring
LCD2004 = Pico W
VCC = VBUS (5V) (red)
SDA = GP20 (Pin #26) (white)
SCL = GP21 (Pin #27) (Pink)
GND = GND (black)
"""

# Libraries
import time
from time import sleep
from machine import Pin
import json
# Call server from server.py (must be uploaded to the picow)
from server import Server
# LCD Display libs stored in Pico W folder lib
# lcd_api.py, machine_i2c_lcd.py
from machine import I2C, Pin
from machine_i2c_lcd import I2cLcd
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
## Name (row 0), Version (row 1), Waiting (row 3) are displayed on the LCD
NAME = 'Domoticz Motherboard'
VERSION = 'v20230311'
WAITING = 'Waiting for data...'
```

```
## Title used for the HTTP JSON response to Domoticz key title
TITLE = 'Set LCD'

# Create the LED1 (blue) object using config.py settings
led1 = Pin(config.PIN_LED1, Pin.OUT)
led1.off()

# LCD2004 Constants
LCD_I2C_ADDRESS = 0x27
LCD_PIN_SDA = 20
LCD_PIN_SCL = 21
LCD_ROWS = 4
LCD_COLS = 20

"""
Create the LCD object by init the lcd with i2c.

:param hex address
    Address of the LCD I2C. Default 0x27

:param int pinsda
    SDA pin

:param int pinscl
    SCL pin

:param int rows
    Number of rows 20 or 16

:param int cols
    Number of cols 4 or 2

:return
    LCD object

:example
init_lcd(LCD_I2C_ADDRESS, LCD_PIN_SDA, LCD_PIN_SCL, LCD_ROWS, LCD_COLS)
"""
def init_lcd(address, pinsda, pinscl, rows, cols):
    try:
        # I2C object
        i2c = I2C(0, sda=Pin(Pinsda), scl=Pin(Pinscl), freq=100000)
        # Init LCD object using I2C with address & rows (4, index 0-3) & cols (20,
index 0-19)
        lcd = I2cLcd(i2c, address, rows, cols)
        print("LCD init. Address: " + str(i2c.scan()))
        return lcd
    except OSError as e:
        raise RuntimeError('[ERROR] LCD init.')

"""
LCD Initial text on row 1 & 2
"""
def set_lcd_welcome(row1, row2, row3, row4):
    lcd.putstr(row1 + "\n" + row2 + "\n" + row3 + "\n" + row4)
    sleep(.3)

"""
Set the sensor text at col, row

:param string Sensor
    String defining the RPi motherboard sensor, i.e. internaltemperature

:example
set_lcd_sensor_text('internaltemperature')
"""
def set_lcd_sensor_text(data, sensor):
    # Get the sensor data
    col = data[sensor]['col']
```

```
    row = data[sensor]['row']
    text = data[sensor]['text']
    # Clear the sensor data text at col, row
    lcd.clrtext(col, row, len(text))
    sleep(.1)
    # Write the sensor text at col, row
    lcd.putstrat(col, row, text)
    sleep(.1)

"""
Handle the LCD command defined as JSON object.
The command defines for every sensor data the text and the LCD start position
col/row.
{'timestamp': {'text': '15:53', 'col': 15, 'row': 1}, 'memoryusage': {'text':
'M:20', 'col': 14, 'row': 3}, 'cpuusage': {'text': 'C:0.39', 'col': 6, 'row': 3},
'internaltemperature': {'text': 'T:39', 'col': 0, 'row': 3}}

:param JSON object
    JSON object with key:value pair {"state":"on" or "off"}

:status
    If status is 1 set the display else unknown command

:return JSON object response
"""
def handle_request(cmd, status):
    # Assign the command to the response title
    response[config.KEY_TITLE] = cmd

    # If the status is 1 (OK) then set the lcd display with the sensor data.
    if status == 1:
        # Clear rows first (the row range for LCD2004 is 0 to 3)
        # Clear row 2 = NOT USED
        lcd.clrrow(2)
        sleep(.1)

        # Row 3 is used to display the RPi motherboard sensor data
        lcd.clrrow(3)
        sleep(.1)

        # Set the sensor data (subset only) on row 3
        set_lcd_sensor_text(cmd, 'timestamp')
        set_lcd_sensor_text(cmd, 'internaltemperature')
        set_lcd_sensor_text(cmd, 'cpuusage')
        set_lcd_sensor_text(cmd, 'memoryusage')

        # Set the response
        response[config.KEY_STATE] = config.STATE_OK
        response[config.KEY_MESSAGE] = config.MESSAGE_EMPTY
    else:
        response[config.KEY_STATE] = config.STATE_ERR
        response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

    # Return the response which is send to Domoticz
    return response

# Main
# Listen for incoming connections from the Domoticz Automation Event dzVents
print(f'{NAME} {VERSION}')

# Create the LCD display object
lcd = init_lcd(LCD_I2C_ADDRESS, LCD_PIN_SDA, LCD_PIN_SCL, LCD_ROWS, LCD_COLS)

# Show initial info on the LCD. Waiting is replaced by RPi motherboard sensor data
set_lcd_welcome(NAME, VERSION, '', WAITING)

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)
```

```
# Connect to the network and get the server object
server = network.connect()

"""
Main Loop
"""
while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Get the cmd to set the LCD text as JSON object from the POST request
        cmd, status = network.parse_post_request(request)

        # Create the HTTP response JSON object
        response = {}

        # Handle the command to update the LCD text.
        # Set the response
        response = handle_request(cmd, status)

        # Send the response to Domoticz and close the connection (wait for new)
        network.send_response(cl, response, True)

    except OSError as e:
        ledstatus.off()
        cl.close()
        print('[ERROR] Network - Connection closed')
```

# LCD 20x4 I2C Text Input

## Description

This project enables to enter text in a Domoticz text device with input field and display the text on an LCD 20x4 I2C display.

**Ideas for Use**

- Message or Information display with text set by user,
- Instead, LCD use a larger display or 7-segment display.

## Solution

This project is

- based upon the project LCD 20x4 I2C LED Control,
- inspired by this solution shared in the Domoticz Forum (thanks).

A Domoticz text device is enhanced by an input field with a button (Set).
If the button is pressed, Domoticz sends an HTTP POST request to the Pico W web server.
The POST request data (JSON format) contains the text to be displayed.
The display lines are separated by comma:

```
"text": "Line1,Line2,Line3,Line4"
```

The Pico W webserver parses the post data, performs checks, and display the text as lines on the LCD display.

*Notes*

- The LCD 20x4 has 20 columns and 4 rows,
- Although the Domoticz database has VARCHAR(200) for the field svalue specified, SQLite3 will not enforce that length specified

A Domoticz Automation event sets the text device input field and sends the HTTP POST request.

# Block Diagram


# Domoticz

## Device

Create a virtual sensor, hardware dummy, named “LCD Text Input” from sensor type Text.

After creating the device, the Domoticz devices list shows the entry:

```
Idx=36, Hardware=VirtualSensors, ID=00082036, Unit=1, Name=LCD Text Input,
Type=General, SubType=Text
```

## Automation Script

```
--[[
File: lcdtextinput.dzvents
Date: 20230425
Author: Robert W.B. Linn
Based on this solution:
https://www.domoticz.com/forum/viewtopic.php?p=293175#p293175 (Thanks)

Create a Domoticz GUI input widget for text. The widget uses a text device.
The text is sent to the Pico W web server and sets the text of up-to 4 lines (LCD
rows 0-3).

As the event is running on the Domoticz system, there is no need to add an IP
address to the HTML form POST action.
See below variable widgetinputhtml.
]]--

local URL_SERVER      = 'http://picow-ip'

-- IDX of the switch which enables to init the textinput device
local IDX_SWITCH = 16

-- IDX of the text device which is functioning as an input device
local IDX_TEXT_DEVICE = 36
local INPUT_LABEL = "Text:"
-- Optional store the text in a user var (NOT USED)
local IDX_UV = 1
```

```
local PROJECT       = 'LCDWIDGETINPUT'
local RES_HTTP      = 'RES_' .. PROJECT
local LOG_MARKER    = 'LOG_' .. PROJECT

-- Create the HTML code for the text device containing input field.
-- This is a form submitting a POST request on the localhost.
-- The input field has type text. The label is commented out. The input field has
default with 200px.
-- The POST data contains all parameter as defined in the text device Domoticz HTTP
API/JSON documentation.
local widgetinputhtml = [[<!-- Widget Input for Text -->
 <iframe name="dummyframe" id="dummyframe" style="display: none">
 </iframe>
 <form method="POST" action="/json.htm" target="dummyframe">
  <input type="hidden" id="" name="type" value="command">
  <input type="hidden" id="" name="param" value="udevice">
  <input type="hidden" id="" name="idx" value="{IDX}"
  <input type="hidden" id="" name="nvalue" value="0">
  <!-- <label for="fname">{LABEL}</label><br> -->
  <input type="text" id="fname" name="svalue" value="{VALUE}" style="width: 200px;">
     <input type="image" class="btnsmall" alt="Set">
</form>]]

-- Set the content of the text device.
-- The content is HTML code (see previous var widgetinputhtml) with the value.
-- Ensure to remove the newline characters.
local function setTextDevice(domoticz, value)
    local text = widgetinputhtml
    text = string.gsub(text, "{IDX}", IDX_TEXT_DEVICE)
    text = string.gsub(text, "{LABEL}", string.format("%s", INPUT_LABEL))
    text = string.gsub(text, "{VALUE}", string.format("%s", value))
    text = string.gsub(text, "\n", "")
    domoticz.log(text)
    -- Silent update of the text device to avoid triggering device change.
    domoticz.devices(IDX_TEXT_DEVICE).updateText(text).silent()
end

local function updateLCD(domoticz, text)
    local postdata = {}
    postdata['text'] = text
    domoticz.openURL({
        url = URL_SERVER, method = 'POST',
        headers = { ['content-type'] = 'application/json' },
        postData = postdata, callback = RES_HTTP,
       })
end

return {
 on = {
  devices = { IDX_SWITCH, IDX_TEXT_DEVICE },
  httpResponses = { RES_HTTP }
 },
 logging = {
  level = domoticz.LOG_DEBUG, marker = LOG_MARKER,
 },
 -- Handle device changes
 execute = function(domoticz, item)
        -- Handle switch which inits the text of the text device with html
        if item.isDevice then
            device = item
            if device.idx == IDX_SWITCH then
                setTextDevice(domoticz, '')
            end

            -- Handle change text device triggered by set button of the HTML form.
            -- The text device content is the inputted value without HTML code.
            if device.idx == IDX_TEXT_DEVICE then
```

```
                -- Get the state of the input device which is the text put in
(without HTML code)
                -- Device TextInputDevice changed, state=Line1,Line2,Line3,Line4
             domoticz.log(string.format('Device change: name=%s, state=%s',
device.name, device.state))

                -- Update text device with new value embedded in the html code
                setTextDevice(domoticz, device.state)

                -- Update LCD display via HTTP POST request to Pico W web server
                updateLCD(domoticz, device.state)
            end
        end

        if (item.isHTTPResponse) then
            -- domoticz.log(item)
            if (item.isJSON) then
                local data = item.json
                domoticz.log(string.format("status=%s, title=%s, message=%s",
data.status, data.title, data.message))
            end
        end
 end
}
```

# Web Server

## Libraries

See project LCD 20x4 I2C LED Control.

## MicroPython Script

```
"""
File: lcdtextinput.py
Date: 20230425
Author: Robert W.B. Linn
"""

# Libraries
from time import sleep
from machine import Pin
import json
# Call server from server.py (must be uploaded to the picow)
from server import Server
# LCD Display libs stored in PicoW folder lib
# lcd_api.py, machine_i2c_lcd.py
from machine import I2C, Pin
from machine_i2c_lcd import I2cLcd
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
NAME = 'LCD Text Input'
VERSION = 'v20230425'

CRLF = chr(13) + chr(10)
SPACE = chr(32)

# LCD2004 Constants
LCD_I2C_ADDRESS = 0x27
LCD_PIN_SDA = 20
LCD_PIN_SCL = 21
LCD_ROWS = 4
LCD_COLS = 20
```

```
def set_lcd(address, pinsda, pinscl, rows, cols):
    """
    Create the LCD object by init the lcd with i2c.

    :param hex address
        Address of the LCD I2C. Default 0x27

    :param int pinsda
        SDA pin

    :param int pinscl
        SCL pin

    :param int rows
        Number of rows 20 or 16

    :param int cols
        Number of cols 4 or 2

    :return
        LCD object

    :example
    set_lcd(LCD_I2C_ADDRESS, LCD_PIN_SDA, LCD_PIN_SCL, LCD_ROWS, LCD_COLS)
    """
    try:
        # I2C object
        i2c = I2C(0, sda=Pin(pinsda), scl=Pin(pinscl), freq=100000)
        # Init LCD object using I2C with address & rows (4, index 0-3) & cols (20,
index 0-19)
        lcd = I2cLcd(i2c, address, rows, cols)
        print("LCD init. Address: " + str(i2c.scan()))
        return lcd
    except OSError as e:
        raise RuntimeError('[ERROR] LCD init.')

def set_lcd_welcome(row1, row2):
    """
    LCD Initial text on row 1 & 2
    """
    lcd.putstr(row1 + "\n" + row2)
    sleep(.3)

def handle_request(cmd, status):
    """
    Handle the LCD command defined as JSON object.

    :param JSON object
        JSON object with key:value pair {"text":"widget_input"}

    :status
        If status is 1 set the display else unknown command

    :return JSON object response
    """
    # print(lcd.num_lines, lcd.num_columns)

    # Assign the command to the response title
    response[config.KEY_TITLE] = cmd
    # Reset the message
    response[config.KEY_MESSAGE] = ''
    # Rest response
    response[config.KEY_STATE] = config.STATE_OK

    # If the status is 1 set the LCD text else error
    if status == 1:
        # Clear the LCD first
        lcd.clear()
```

```
        # Write the text input from the widget starting at col 0, row 0
        # Get the text
        text = cmd['text']
        # Split the text by , into up-to 4 lines 0-3
        lines = text.split(",")
        linenr = 0
        # Loop over the lines and write the line at col 0, row rownr
        # Check if the number of lines and number of columns are in range
        for line in lines:
            if 0 <= linenr <= lcd.num_lines - 1:
                if len(line) > lcd.num_columns:
                    response[config.KEY_MESSAGE] = f'[WARNING] Line {linenr}
exceeds max length {str(lcd.num_columns)} ({str(len(line))}).'
                    response[config.KEY_STATE] = config.STATE_WARNING
                    print(response[config.KEY_MESSAGE])
                    line = line[0:lcd.num_columns]
                lcd.putstrat(0, linenr, line)
                linenr = linenr + 1
            else:
                response[config.KEY_MESSAGE] = f'[WARNING] Line {linenr} ("{line}")
out of range 0-{lcd.num_lines - 1}.'
                response[config.KEY_STATE] = config.STATE_WARNING
                print(response[config.KEY_MESSAGE])
        # sleep(.3)
    else:
        response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN
        response[config.KEY_STATE] = config.STATE_ERR

    return response

# Main
# Listen for incoming connections from the Domoticz Automation Event dzVents
print(f'{NAME} {VERSION}')

# Create the LCD object
lcd = set_lcd(LCD_I2C_ADDRESS, LCD_PIN_SDA, LCD_PIN_SCL, LCD_ROWS, LCD_COLS)

# Set the LCD display welcome text
set_lcd_welcome(NAME, VERSION)

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()

"""
Main Loop
"""
while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Get the cmd as JSON object from the POST request
        # {"state":"on" or "off"}
        cmd, status = network.parse_post_request(request)

        # Create the HTTP response JSON object
        response = {}

        # Handle the command to update the LCD text.
        # Set the response
        response = handle_request(cmd, status)

        # Send the response to Domoticz and close the connection (wait for new)
        network.send_response(cl, response, True)
```

```
    except OSError as e:
        ledstatus.off()
        cl.close()
        print('[ERROR] Network Connection closed')
```

# Enhancements

Few enhancement ideas.

Use a user variable or persistent data (dzVents) to store the text (only) input.

Use text input device to send an emergency message. This handled by the Automation script.

Use number input to set an 7-segment LED display.

# TM1637 4-digit 7-segment LED Display

## Description

This project displays and updates in regular intervals selective Domoticz sensor data on a 4-digit 7-segment LED display connected to a Pico W.

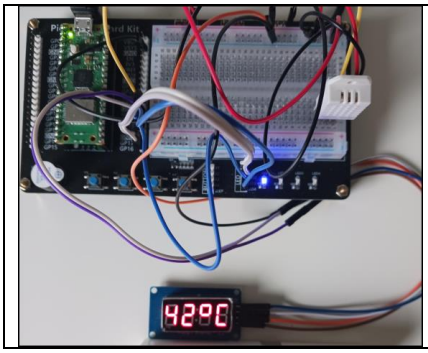
## Solution

The Pico W web server listens to incoming HTTP POST requests.
The POST data is a JSON object with key:value pair(s) to set the TM1637 display:

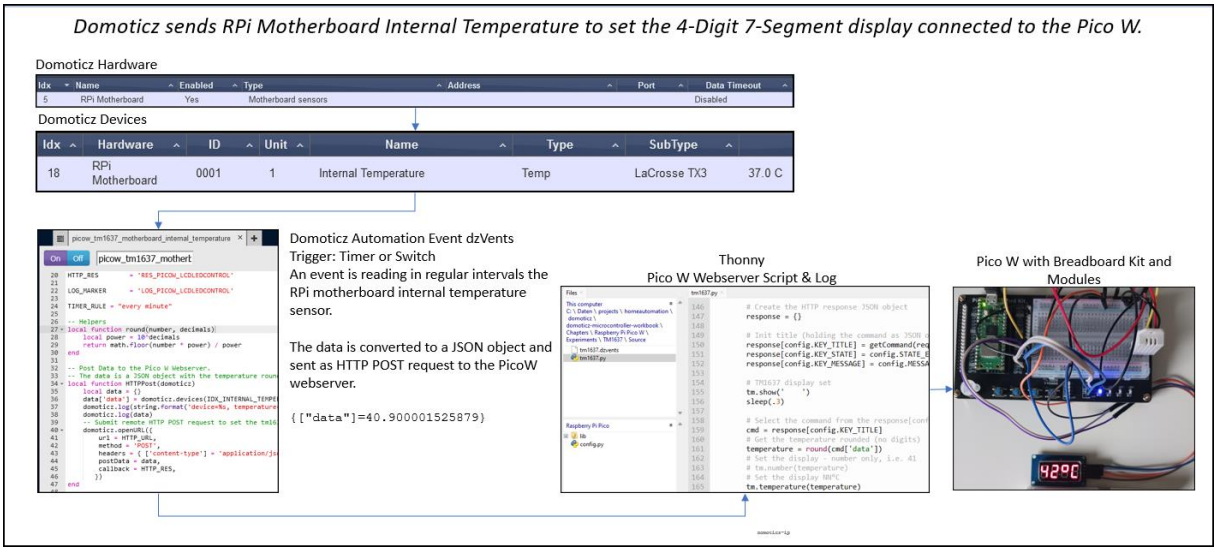
```
{"data":NNNN}
```

Domoticz uses an Automation Event dzVents, triggered by a switch or timer, to send data to the Pico W web server.

To get started using the TM1637, the display shows the Raspberry Pi Motherboard Internal Temperature.



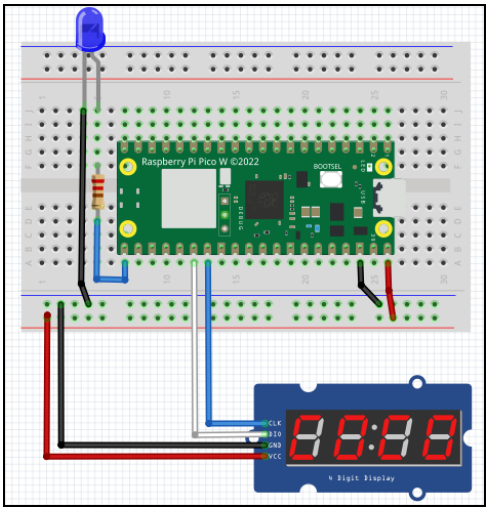

## Block Diagram




## Wiring

| **TM1637 I2C** | **Pico W** |
|---|---|
| VCC | VBUS (5V) |
| SDA | GP20 (Pin #26) |
| SCL | GP21 (Pin #27) |
| GND | GND (Pin #38) |
| | |
| **LED (blue)** | **Pico W** |
| + (Anode) | GP16 (Pin #21) |
| GND (Cathode) | GND (Pin #38) |

# Circuit Diagram




# Domoticz Setup

## Devices

The hardware “Motherboard sensors” is added and selective devices, like the Internal Temperature are added.
The other devices from the hardware Motherboard sensors are not used for this project.

After creating the device, the Domoticz devices list shows the entries:

```
…
IDX=18, Hardware=RPi Motherboard, ID=0001, Unit=1, Name=Internal Temperature,
Type=Temp, SubType=LaCrosse TX3, Data=38.4 C
…
```

Note: Only the Internal Temperature device is shown here

## Automation Script

```
--[[
File:   tm1637_motherboard_internal_temperature.dzvents
Date:   20230304
Author: Robert W.B. Linn

Display the motherboard interna temperature on a TM1637 connected to the Pico W.
The Pico W runs a RESTful web server.
The HTTP POST request to the Pico W web server is a JSON object: {"data":NN}.
The NN is the temperature with 0 digit (integer).
The Pico W web server HTTP response is a JSON object with key:value pairs:
{["message"]="On", ["title"]="{"data": "NN.N"}", ["status"]="OK"}
]]--

-- Domoticz
local IDX_INTERNAL_TEMPERATURE = 18
local IDX_SWITCH = 16

local HTTP_URL        = 'http://picow-ip'
-- HTTP_POST_DATA  = '{"data":"on"}'
local HTTP_RES        = 'RES_PICOW_LCDLEDCONTROL'
local LOG_MARKER      = 'LOG_PICOW_LCDLEDCONTROL'
local TIMER_RULE = "every minute"

-- Helpers
local function round(number, decimals)
    local power = 10^decimals
    return math.floor(number * power) / power
end

-- Post Data to the Pico W web server.
-- The data is a JSON object with the temperature rounded: {"data":NN}
local function HTTPPost(domoticz)
    local data = {}
    data['data'] = domoticz.devices(IDX_INTERNAL_TEMPERATURE).temperature
    domoticz.log(string.format('device=%s, temperature=%.1f',
                domoticz.devices(IDX_INTERNAL_TEMPERATURE).name,
                domoticz.devices(IDX_INTERNAL_TEMPERATURE).temperature),
domoticz.LOG_INFO)
    -- Submit remote HTTP POST request to set the tm1637
    domoticz.openURL({
        url = HTTP_URL, method = 'POST',
        headers = { ['content-type'] = 'application/json' },
        postData = data, callback = HTTP_RES,
       })
end

return {
 on = {
     devices = { IDX_SWITCH, IDX_INTERNAL_TEMPERATURE },
     timer = { TIMER_RULE },
     httpResponses = { HTTP_RES }
    },
 logging = { level = domoticz.LOG_INFO, marker = LOG_MARKER },
 execute = function(domoticz, item)
  if (item.isTimer) then
    HTTPPost(domoticz)
   end
  if (item.isDevice) then
    HTTPPost(domoticz)
   end
   if (item.isHTTPResponse) then
     if (item.isJSON) then
       local data = item.json

domoticz.log(string.format("status=%s,title=%s,message=%s",data.status,data.title,
data.message))
```

```
      end
    end
  end
}
```

# Web Server

## Libraries

The MicroPython script uses the external MicroPython TM1637 library to control the TM1637.
The library is stored on the Pico W in folder lib (see sub chapter Additional Libraries).

**Credit**
Thanks for developing & sharing the MicroPython TM1637 library.

## MicroPython Script

```
"""
File:tm1637.py
Date:20230318
Author: Robert W.B. Linn

:description
Pico W RESTful web server listening for data from Domoticz event.
The incoming data is from a HTTP POST request with JSON object.
The JSON object has key:value pais: {"data":NNNN}
LED1 is attached on the Pico Breadboard kit.

:log
Domoticz TM1637 v20230304
TM1637 init.
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Network client connected from client-ip
HTTP Command={'data': 40.90001}
HTTP Response={"title": {"data": 40.90001}, "message": "41°C", "status": "OK"}
Network connection closed

:wiring
TM1637 = Pico W
VCC = VBUS (5V) (red)
DIO = GP20 (Pin #26) (white)
CLK = GP21 (Pin #27) (Pink)
GND = GND (black)
"""

# Libraries
import time
from time import sleep
from machine import Pin
# Call server from server.py (must be uploaded to the picow)
from server import Server
# TM1637 lib stored in Pico W folder lib
import tm1637
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
NAME = 'Domoticz TM1637'
VERSION = 'v20230304'

# Create the LED1 (blue) object using config.py settings
led1 = Pin(config.PIN_LED1, Pin.OUT)
led1.off()

# LCD2004 Constants
TM1637_I2C_ADDRESS = 0x27
```

```
TM1637_PIN_DIO = 20
TM1637_PIN_CLK = 21

# Create the TM1637 object by init the TM1637 with i2c.
# Example: setTM1637(TM1637_I2C_ADDRESS, TM1637_PIN_DIO, TM1637_PIN_CLK)
# Return - TM1637 object
def init_tm1637(address, pindio, pinclk):
    try:
        # Init TM1637 object
        tm = tm1637.TM1637(clk=Pin(Pinclk), dio=Pin(Pindio))
        print("TM1637 init.")
        return tm
    except OSError as e:
        raise RuntimeError('[ERROR] TM1637 init.')

# TM1637 set display from the JSON object.
# cmd - Command from the key response[config.KEY_TITLE]
# Example: cmd['data']=41.3001
# Return - Temperature rounded
def set_tm1637(cmd, status):
    # Assign the command to the response title
    response[config.KEY_TITLE] = cmd

    # If the status is 1 (OK) then set the tm1637 with data.
    if status == 1:
        # Clear the display
        tm.show('    ')
        sleep(.3)

        # Get the temperature rounded (no digits) from the JSON key 'data'
        temperature = round(cmd['data'])

        # Set the display NN°C
        tm.temperature(temperature)

        # Not Used = Just to show
        # Set the display - number only, i.e. 41
        # tm.number(temperature)

        # Set the response
        # Convert the KEY_TITLE from JSON object to a string
        response[config.KEY_MESSAGE] = str(temperature) + "°C"
        response[config.KEY_STATE] = config.STATE_OK
    else:
        response[config.KEY_STATE] = config.STATE_ERR
        response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

    return response

# Main
# Listen for incoming connections from the Domoticz Automation Event dzVents
print(f'{NAME} {VERSION}')

# Set the TM1637 display
tm = init_tm1637(TM1637_I2C_ADDRESS, TM1637_PIN_DIO, TM1637_PIN_CLK)
tm.show('1958')
#tm.show('    ')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()

"""
Main Loop
"""
while True:
    try:
```

```
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Get the cmd to set the LCD text as JSON object from the POST request
        cmd, status = network.parse_post_request(request)

        # Create the HTTP response JSON object
        response = {}

        # Set the display with data from the command
        response = set_tm1637(cmd, status)

        # Send the response to Domoticz and close the connection (wait for new)
        network.send_response(cl, response, True)

    except OSError as e:
        ledstatus.off()
        cl.close()
        print('[ERROR] Network Connection closed')
```

# Servo Motor

## Description

This project enables to control, via Domoticz, a servo motor connected to the Pico W.

**Ideas for Use**

- Use as a switch.

## Solution

The Pico W is built in a Pico Breadboard Kit.
The servo, type Tower Pro Micro Servo 9g SG90, is connected to the Pico W.

A RESTful web server runs on the Pico W.
If the web server network connection is successful, the Pico W onboard LED is ON else OFF indicating an error.
The web server listens to incoming client connections, which are HTTP POST requests.
The POST data is a JSON object with the key:value pair to set the angle (position) of the servo:

```
{"angle":NNN}
```

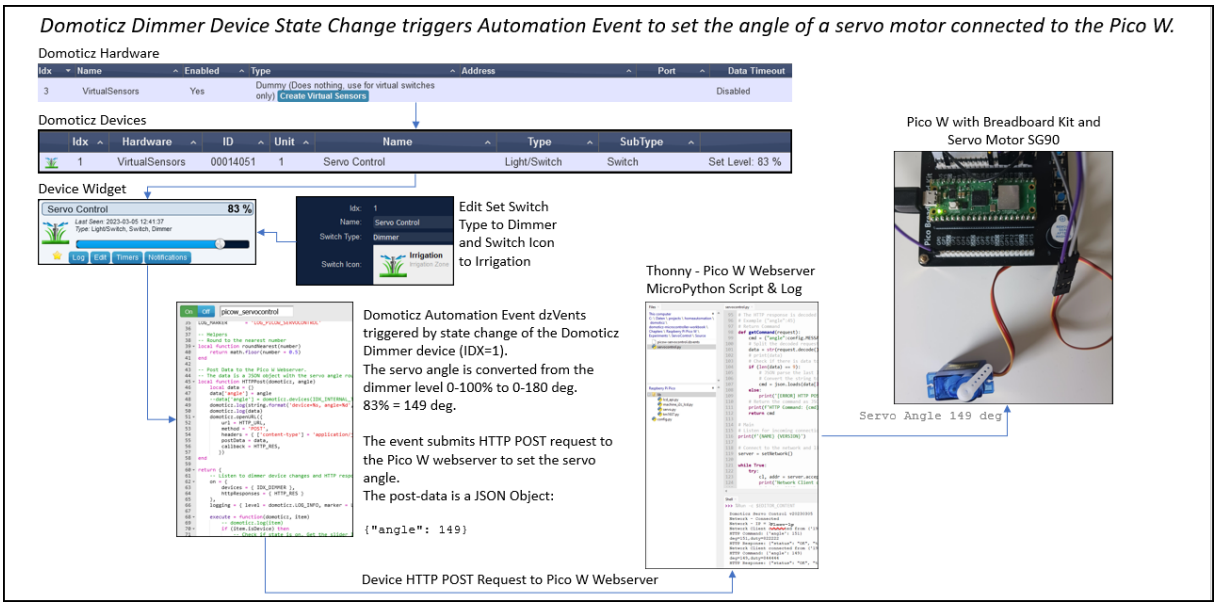
The angle is between 0-180 degrees.

A Domoticz switch device type dimmer sets the position of the servo motor (0-100% is converted to 0-180 degrees).
If the level value changes, a Domoticz Automation Event dzVents submits the HTTP POST request to the Pico W web server.
The web server sets the servo angle.
If the dimmer is set to off, the angle is set to 0. If set to on, the last level is set as shown by the dimmer slider.

**Flow**

1. Create the Domoticz Switch device from the hardware “Dummy” (Create Virtual Sensors) ,
2. Edit the Switch device widget and set the type to Dimmer.
   The Switch icon is also changed (as a test),
3. Run the Pico W web server (Thonny),
4. In Domoticz change the dimmer level of the switch device,
5. The automation event is triggered and converts the dimmer level 0-100% to a servo motor angle 0-180 deg. The angle is used as post-data for the HTTP POST request to the Pico W web server,
6. After receiving data, the Pico W web server set the servo motor angle and submits an HTTP POST request back to Domoticz with the status. The LED1 is on whilst setting the servo motor angle.

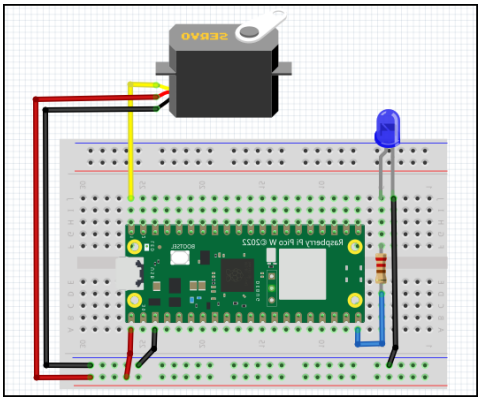
## Block Diagram




# Wiring

| **Servo Motor**<br>**Tower Pro Micro Servo 9g SG90** | **Pico W** |
|---|---|
| VCC | VBUS (5V) |
| Signal | GP0 (Pin #1) |
| GND | GND (Pin #38) |
| | |
| **LED (blue)** | **Pico W** |
| + (Anode) | GP16 (Pin #21) |
| GND (Cathode) | GND (Pin #38) |

# Circuit Diagram



# Domoticz Setup

## Device

From the Domoticz hardware Dummy, a switch device is added, and its type changed to dimmer.

After creating the device, the Domoticz devices list shows the entry:

```
IDX=1, Hardware=VirtualSensors, ID=00014051, Unit=1, Name=Servo Control,
Type=Light/Switch, SubType=Switch, Date=Set Level: 28 %
```

## Automation Script

```
--[[
File:   servocontrol.dzvents
Date:   20230305
Author: Robert W.B. Linn

Display the motherboard_internal_temperature on a TM1637 connected to the Pico W.
The Pico W runs a RESTful web server.
The HTTP POST request to the Pico W web server is a JSON object: {"angle":NNN}.
The NN is the angle of the servo to set. The angle is converted from the dimmer
range 0-100% to 0-180 deg.
The Pico W web server HTTP response is a JSON object with key:value pairs:
{"status": "OK", "title": "{\"angle\": 133}", "message": "133"}
]]--

-- Domoticz
local IDX_SWITCH = 16
local IDX_DIMMER = 1

local HTTP_URL        = 'http://picow-ip'
-- HTTP_POST_DATA  = '{"angle":90}'
local HTTP_RES        = 'RES_PICOW_SERVOCONTROL'
local LOG_MARKER      = 'LOG_PICOW_SERVOCONTROL'

-- Round to the nearest number
local function roundNearest(number)
    return math.floor(number + 0.5)
end

-- Post Data to the Pico W web server.
-- The data is a JSON object with the servo angle rounded: {"angle":NNN}
local function HTTPPost(domoticz, angle)
    local data = {}
    data['angle'] = angle
    --data['angle'] = domoticz.devices(IDX_INTERNAL_TEMPERATURE).temperature
    domoticz.log(string.format('device=%s,
angle=%d',domoticz.devices(IDX_SWITCH).name, angle), domoticz.LOG_INFO)
    domoticz.log(data)
    domoticz.openURL({
        url = HTTP_URL,
        method = 'POST',
        headers = { ['content-type'] = 'application/json' },
        postData = data,
        callback = HTTP_RES,
       })
end

return {
    -- Listen to dimmer device changes and HTTP responses
 on = {
     devices = { IDX_DIMMER },
     httpResponses = { HTTP_RES }
```

```
    },
 logging = { level = domoticz.LOG_INFO, marker = LOG_MARKER },

 execute = function(domoticz, item)
       -- domoticz.log(item)
  if (item.isDevice) then
           -- Check if state is on. Get the slider level to set the angle.
           if (item.state == 'On') then
          -- Get the level 0 - 100 %
          level = tonumber(item.levelVal)
          -- Convert the level to angle 0 - 180 deg
          angle = roundNearest((180 / 100) * level)
           end
           -- Check if the state is Off (the off button pressed)
           if (item.state == 'Off') then angle = 0 end

           -- Log
      domoticz.log(string.format("device=%s, state=%s, levelVal=%d, angle=%d",
item.name, item.state, item.levelVal, angle))

      -- Set the servo angle
           HTTPPost(domoticz, angle)
       end

       if (item.isHTTPResponse) then
           -- domoticz.log(item)
           if (item.isJSON) then
               -- {"status": "OK", "title": "{\"angle\": 133}", "message": "133"}
               local data = item.json
               domoticz.log(string.format("status=%s, title=%s, message=%s",
data.status, data.title, data.message))
           end
       end

 end
}
```

Domoticz Log

```
2023-03-05 14:10:59.616 VirtualSensors: Light/Switch (Servo Control)
2023-03-05 14:10:59.610 Status: User: admin initiated a switch command (1/Servo
Control/Set Level)
2023-03-05 14:10:59.705 Status: dzVents: Info: Handling events for: "Servo
Control", value: "On"
2023-03-05 14:10:59.705 Status: dzVents: Info: LOG_PICOW_SERVOCONTROL: ------ Start
internal script: picow_servocontrol: Device: "Servo Control (VirtualSensors)",
Index: 1
2023-03-05 14:10:59.705 Status: dzVents: Info: LOG_PICOW_SERVOCONTROL: device=Servo
Control, state=On, levelVal=45, angle=81
2023-03-05 14:10:59.706 Status: dzVents: Info: LOG_PICOW_SERVOCONTROL: device=Pico
W LED1 Control, angle=81
2023-03-05 14:10:59.706 Status: dzVents: Info: LOG_PICOW_SERVOCONTROL:
{["angle"]=81}
2023-03-05 14:10:59.706 Status: dzVents: Info: LOG_PICOW_SERVOCONTROL: ------
Finished picow_servocontrol
2023-03-05 14:10:59.707 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
2023-03-05 14:11:00.079 Status: dzVents: Info: Handling httpResponse-events for:
"RES_PICOW_SERVOCONTROL"
2023-03-05 14:11:00.079 Status: dzVents: Info: LOG_PICOW_SERVOCONTROL: ------ Start
internal script: picow_servocontrol: HTTPResponse: "RES_PICOW_SERVOCONTROL"
2023-03-05 14:11:00.079 Status: dzVents: Info: LOG_PICOW_SERVOCONTROL: status=OK,
title={"angle": 81}, message=81
2023-03-05 14:11:00.079 Status: dzVents: Info: LOG_PICOW_SERVOCONTROL: ------
Finished picow_servocontrol
```

# Web Server

## Libraries

The MicroPython script uses the library *servo.py* to set the angle of the servo motor.
The library is stored on the Pico W in folder lib (see sub chapter Additional Libraries).

```
"""
File: servo.py
Date: 20230305
Author: Robert W.B. Linn

Class to control a servo motor.
Tested with a Tower Pro Micro Servo 9 g SG90.
Min and max duty default values are obtained via try out.
Min angle is 0, max angle is 180.
Servo signal pin is default Pico Pin GP0 (Pin #1).
"""

# Imports
import machine
from machine import Pin, PWM

# Servo Class
class Servo:
    """
    Init the servo with defaults.

    :parameter long MIN_DUTY

    :parameter long MAX_DUTY

    :parameter int pin

    :parameter int frequency
    """
    def __init__(self, MIN_DUTY=500000, MAX_DUTY=2500000, pin=0, frequency=50):
        self.pwm = machine.PWM(machine.Pin(Pin))
        self.pwm.freq(frequency)
        self.MIN_DUTY = MIN_DUTY
        self.MAX_DUTY = MAX_DUTY

    """
    Set the servo angle between 0 - 180 degrees.

    :parameter int angle
        Set the angle of the servo between 0 - 180 degrees.

    :return flat duty_ns
    """
    def setAngle(self, angle):
        if angle < 0:
            angle = 0
        elif angle > 180:
            angle = 180
        duty_ns = int(self.MAX_DUTY - angle * (self.MAX_DUTY-self.MIN_DUTY)/180)
        # print(duty_ns)
        self.pwm.duty_ns(duty_ns)
        return duty_ns
```

## MicroPython Script

```
"""
File: servocontrol.py
Date: 20230318
Author: Robert W.B. Linn

Pico W RESTful web server listening for data from Domoticz event.
The incoming data is from a HTTP POST request with JSON object to set the position
(angle) of a servo motor 0-180 degrees.

:log
Domoticz Servo Control v20230311
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Network client connected from client-ip
HTTP Command={'angle': 180}
Servo position deg=180,duty=500000
HTTP Response={"status": "OK", "title": {"angle": 180}, "message": "180"}
Network connection closed

:wiring
Servo = Pico W
VCC = VBUS (5V) (red)
Signal = GP0 (Pin #1) (yellow)
GND = GND (black)
"""

# Libraries
import time
from time import sleep
from machine import Pin
import json
# Call server from server.py (must be uploaded to the picow)
from server import Server
# Servo lib stored in Pico W folder lib
from servo import Servo
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
NAME = 'Domoticz Servo Control'
VERSION = 'v20230311'

# Create the LED1 (blue) object using config.py settings
led1 = Pin(2, Pin.OUT)
led1.off()

# Create the servo object with the default (GP0 (Pin #1)
servo = Servo()

# Set the servo pos and log
def set_servo_position(pos):
    duty = servo.set_angle(pos)
    print("Servo position deg=" + str(pos) + ",duty=" + str(duty))

"""
Handle the resquest to set the servo pos between 0-180 degrees.
"""
def handle_request(cmd, status):
    # Assign the command to the response title
    response[config.KEY_TITLE] = cmd

    # If the status is 1 (OK) then set the lcd display with the sensor data.
    if status == 1:
        led1.on()
```

```
        # Get the angle of the servo to set
        angle = cmd['angle']

        # Set the servo pos
        set_servo_position(angle)

        # Set the response
        response[config.KEY_STATE] = config.STATE_OK
        response[config.KEY_MESSAGE] = str(angle)

        led1.off()
    else:
        response[config.KEY_STATE] = config.STATE_ERR
        response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

    # Return the response which is send to Domoticz
    return response

# Main
print(f'{NAME} {VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()

"""
Main Loop
"""
while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Get the cmd to set the LCD text as JSON object from the POST request
        cmd, status = network.parse_post_request(request)

        # Create the HTTP response JSON object
        response = {}

        # Handle the command to set the servo pos
        # Set the response
        response = handle_request(cmd, status)

        # Send the response to Domoticz and close the connection (wait for new)
        network.send_response(cl, response, True)

    except OSError as e:
        ledstatus.off()
        cl.close()
        print('[ERROR] Network Connection closed')
```

## Thonny Log

```
Domoticz Servo Control v20230305
Network - Connected
Network - IP = picow-ip
Network Client connected from ('client-ip', 52702)
HTTP Command: {'angle': 151}
deg=151,duty=822222
HTTP Response: {"status": "OK", "title": "{\"angle\": 151}", "message": "151"}
Network Client connected from ('client-ip', 46286)
HTTP Command: {'angle': 149}
deg=149,duty=844444
HTTP Response: {"status": "OK", "title": "{\"angle\": 149}", "message": "149"}
```

# Enhancement Ideas

Some ideas whilst exploring setting the servo motor. These depend of course on solution.

- Domoticz selector switch with angle positions, like 0, 45, 90, 135 and 180.
- Show the angle on a 4-digit 7-segment display TM1637.
- Use a linear servo to control a throttle or switch.

# RFID Reader

## Description

This project reads the UID of an RFID card and sends the data to Domoticz text device.

**Ideas for Use**

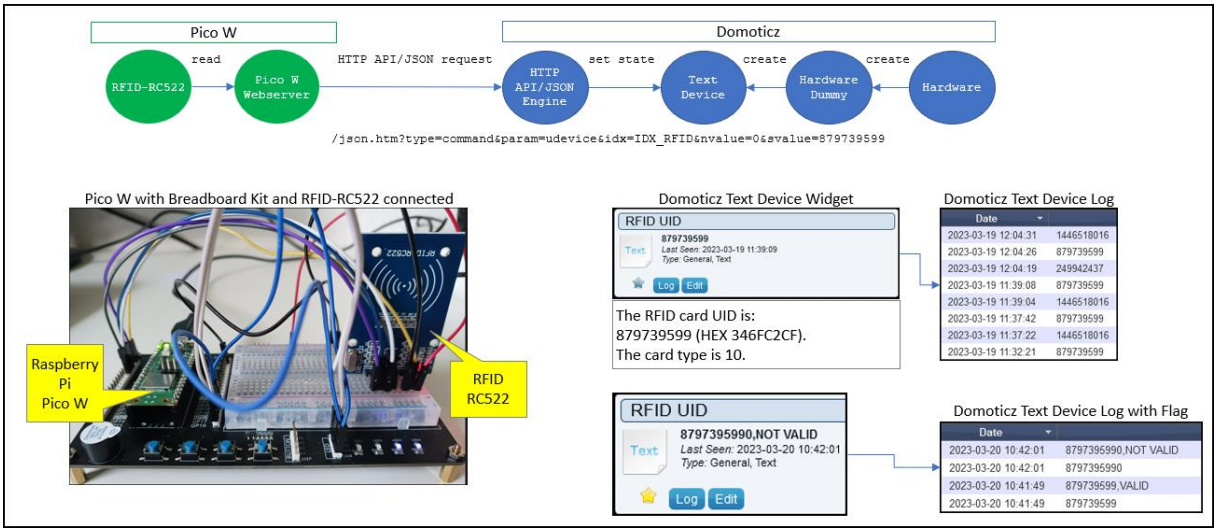
- Access control,
- Open garage door.

## Solution

The Pico W web server listens to card readings of the RFID-RC522 sensor connected.
An Domoticz text device is updated with the card UID as decimal value using HTTP API/JSON request to the Domoticz server.
The log of the Domoticz text device lists all the cards read.
The time between the card readings must be greater threshold (default 2 seconds) to avoid multiple readings and updates of the Domoticz text device.

For a later version, its planned to send the RFID card data, beside the RFID card UID. This could be done via custom event.

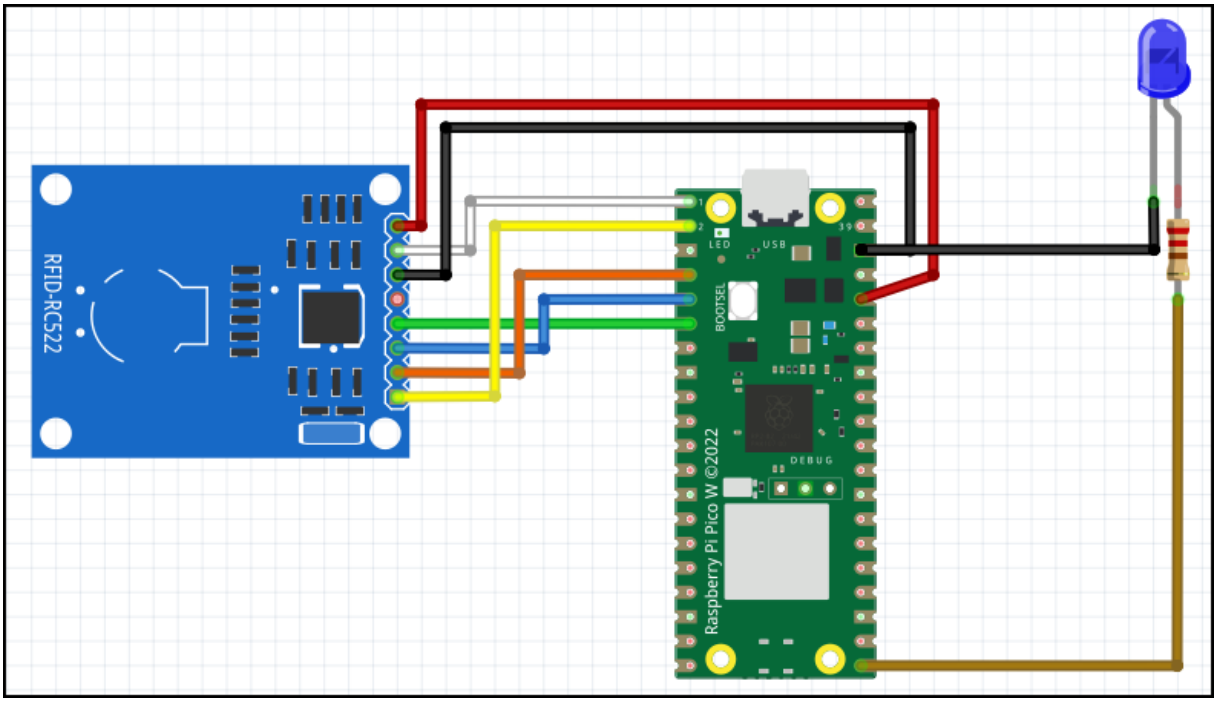
## Block Diagram



# Wiring

| **RFID-RC522** | **Pico W** |
|---|---|
| VCC | 3V3 (Pin #36) |
| RST | GP0 (Pin #1) |
| GND | GND (Pin #38) |
| IRQ | Not connected |
| MISO | GP4 (Pin #6) |
| MOSI | GP3 (Pin #5) |
| SCK | GP2 (Pin #4) |
| SDA | GP1 (Pin #2) |
| | |
| **LED (blue)** | **Pico W** |
| + (Anode) | GP16 (Pin #21) |
| GND (Cathode) | GND (Pin #38) |

# Circuit Diagram



# Domoticz Setup

## Device

A text device is used to store the card UID as a decimal value i.e., 879739599.
From the Domoticz hardware Dummy, add a text device:
Create a virtual sensor with Name: RFID UID, Sensor Type: Text

After creating the device, the Domoticz devices list shows the entry:

```
Idx=27, Hardware=VirtualSensors, ID=00082027, Unit=1, Name=RFID UID, Type=General,
SubType=Text, Data=Hello World
```

## Device Update

The device is updated via HTTP API/JSON request for a Text sensor. This is handled by the Pico W web server.

```
http://domoticz-
ip:port/json.htm?type=command&param=udevice&idx=IDX&nvalue=0&svalue=TEXT
```

# Web Server

## Libraries

The MicroPython script uses the external library PiPicoRFID to read the card.
The library is stored on the Pico W in folder lib (see sub chapter Additional Libraries).

**Credit**
Thanks for developing & sharing the PiPicoRFID library.

## MicroPython Script

```
"""
File: rfid.py
Date: 20230319
Author: Robert W.B. Linn

:description
Read the UID of an RFID card and send to Domoticz text device.
The Domoticz text device is updated using HTTP API/JSON request to the Domoticz
server.
The log of the Domoticz text device lists all the cards read.
The time between the card readings must be greater threshold to avoid multiple
readings and updates of the Domoticz text device.
The threshold in seconds is defined as constant MIN_DELTA_TIME = 2.

:notes
Pico Breadboard Kit is used to wire up the RFID.
Pico Breadboard Kit LED1 is used as status LED when requesting RFID data and
updating domoticz.
Configuration stored in config.py, ensure to upload to the picow.
For a later version, its planned to send the RFID card data, beside the RFID card
UID. This could be done via custom event.

:credits
The MFRC522 library PiPicoRFID from Saket Upadhyay (https://github.com/Saket-
Upadhyay/PiPicoRFID).
```

```
The base code is from https://github.com/wendlers/micropython-mfrc522.

:log
RFID v20230319
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Init RFID Module=rp2
CARD DETECTED: tag_type=10, uid_hex=346FC2CF, uid_dec=879739599
Send GET request url=http://domoticz-
ip:8080/json.htm?type=command&param=udevice&idx=27&nvalue=0&svalue=879739599
Send GET request status=OK
CARD DETECTED: tag_type=10, uid_hex=56381D00, uid_dec=1446518016
Send GET request url=http://domoticz-
ip:8080/json.htm?type=command&param=udevice&idx=27&nvalue=0&svalue=1446518016
Send GET request status=OK

:wiring
RFID-RC522 Module = Pico W
VCC = 3.3V
RST = GP0
GND = GND
IRQ = Not connected
MISO = GP4
MOSI = GP3
SCK = GP2
SDA = GP1
"""

"""
Imports
"""
from machine import Pin
from utime import sleep
import time
# RFID
import mfrc522
from os import uname
# Call server from server.py (must be uploaded to the picow)
from server import Server
# Configuration (must be uploaded to the picow)
import config

# Constants
VERSION = 'RFID v20230319'

# Create the led object indicating RFID read in progress
led1 = Pin(config.PIN_LED1, Pin.OUT)
led1.value(0)

"""
Domoticz
"""
# IDX text device
IDX_RFID = 27
# URL to update the text device
# Note the idx of the domoticz device ( see GUI > Setup > Devices)
# The svalue is added in the main loop after getting the data from the RFID.
URL_DOM = "http://"+ config.DOMOTICZ_IP
+"/json.htm?type=command&param=udevice&idx=" + str(IDX_RFID) + "&nvalue=0&svalue="

"""
RFID
"""
# Flag to read the data
# Not used as only the card uid is used
READ_DATA = False

"""
```

```
RFID Timer
"""
first_time_reading = True
# Delta time between readings in seconds
MIN_DELTA_TIME = 2
# Start time in seconds
start_time = time.time()
# Read_time
read_time = start_time
# Previous UID NOT USED
uid_previous = -1

"""
RFID object init

:return object mfrc522
"""
def init_rfid():
    print(f'Init RFID Module={str(uname()[0])}')
    return mfrc522.MFRC522(sck=2, miso=4, mosi=3, cs=1, rst=0)
    # print(f'Place card before reader. READ ARRD: 0x08')

"""
Read the RFID data (optional)

:param object rdr
    Card reader object

:return string hexstr
    Card data as hex string

:example
read_data(rdr)
RAW DATA: ['0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0',
'0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0']
['0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0', '0x0',
'0x0', '0x0', '0x0', '0x0', '0x0', '0x0']
"""
def read_data(rdr):
    hexstr = []
    # Get the card data (optional)
    if rdr.select_tag(raw_uid) == rdr.OK:

        key = [0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF]
        auth = rdr.auth(rdr.AUTHENT1A, 8, key, raw_uid)
        if rdr.auth(rdr.AUTHENT1A, 8, key, raw_uid) == rdr.OK:
            data = rdr.read(8)
            # print(data) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
            if data is not None:
                for i in data:
                    hexstr.append(hex(i))
            print("RAW DATA: " + str(hexstr))
            rdr.stop_crypto1()
        else:
            print("AUTH ERR")
    else:
        print("Failed to select tag")

    return hexstr

"""
Main
"""
print(f'{VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
```

```
server = network.connect()

# Init the RFID module
rdr = init_rfid()

while True:

    # Wait for RFID card to read.
    # If RFID card found, send the card UID as decimal to Domoticz text device
    # Read the rfid card data
    (stat, tag_type) = rdr.request(rdr.REQIDL)
    # print(stat)

    # If the reader status is OK, then get the card type and uid
    if stat == rdr.OK:
        # LED1 indicator on
        led1.value(1)

        # Get the reading status and uid as raw data
        (stat, raw_uid) = rdr.anticoll()

        # If the status is OK, lets send the UID to Domoticz
        if stat == rdr.OK:
            # Get the tag type, i.e. 10
            tagtype = f'{tag_type:0X}'

            # Get the card uid as hex (each byte 2 hex size in uppercase) and dec,
i.e. 56381D00 = 1446518016
            # The dec value is send to domoticz
            uid_hex =
f'{raw_uid[0]:0>2X}{raw_uid[1]:0>2X}{raw_uid[2]:0>2X}{raw_uid[3]:0>2X}'
            uid_dec = int(uid_hex, 16)
            print(f'CARD DETECTED: tag_type={tagtype}, uid_hex={uid_hex},
uid_dec={uid_dec}')

            # Read_time and check the delta time in seconds between the readings
            # This to avoid multiple text device updates within (milli)seconds
            read_time = time.time()
            delta_time = read_time - start_time
            # print(f'{start_time}, {read_time}, {delta_time}')

            # Update domoticz if the delta time exceeded or if first time reading
            if delta_time > MIN_DELTA_TIME or first_time_reading:
                first_time_reading = False
                uid_previous = uid_dec
                start_time = read_time

                # Submit Domoticz HTTP API/JSON GET request to update the device
                # The uid card id decimal value is submitted to the Domoticz text
device
                network.send_get_request(URL_DOM + str(uid_dec))

                # Read data is not used and to be developed further like handling
auth error
                if READ_DATA:
                    data = read_data(rdr)
                    # network.send_get_request(URL_DOM + data)

        # LED1 indicator off
        led1.value(0)
```

### Thonny Log

```
RFID v20230319
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Init RFID Module=rp2
```

```
CARD DETECTED: tag_type=10, uid_hex=346FC2CF, uid_dec=879739599
Send GET request url=http://domoticz-
ip:port:port/json.htm?type=command&param=udevice&idx=27&nvalue=0&svalue=879739599
Send GET request status=OK
```

# Enhancement Ideas

Some ideas whilst exploring the card reader.

## Device Change

Use an Automation Event dzVents to handle changes of the text device, i.e., the card or token UID received from the Pico W.
The card or token UID can be checked to trigger any action (see next enhancement idea).
The action could be to set a switch or send a notification.

## Automation Script

```
--[[
File:   rfid_device_change.dzvents
Date:   20230320
Author: Robert W.B. Linn

:description
Handle the rfid text device state change triggered by the Pico W web server
submitting HTTP API/JSOM requests.

2023-03-19 11:37:33.558 Status: dzVents: Info: LOG_PICOW_RFID: ------ Start
internal script: picow_rfid: Device: "RFID UID (VirtualSensors)", Index: 27
2023-03-19 11:37:33.558 Status: dzVents: Info: LOG_PICOW_RFID: Device RFID UID:
statechange=1446518016
2023-03-19 11:37:33.558 Status: dzVents: Info: LOG_PICOW_RFID: Card UID 1446518016
is WRONG.
2023-03-19 11:37:33.558 Status: dzVents: Info: LOG_PICOW_RFID: ------ Finished
picow_rfid
2023-03-19 11:37:42.775 Status: dzVents: Info: Handling events for: "RFID UID",
value: "879739599"
2023-03-19 11:37:42.775 Status: dzVents: Info: LOG_PICOW_RFID: ------ Start
internal script: picow_rfid: Device: "RFID UID (VirtualSensors)", Index: 27
2023-03-19 11:37:42.775 Status: dzVents: Info: LOG_PICOW_RFID: Device RFID UID:
statechange=879739599
2023-03-19 11:37:42.775 Status: dzVents: Info: LOG_PICOW_RFID: Card UID 879739599
is OK.
2023-03-19 11:37:42.775 Status: dzVents: Info: LOG_PICOW_RFID: ------ Finished
picow_rfid
]]--

-- Domoticz IDX of the text device state is set by the Pico W web server
IDX_DEVICE  = 27

UID_CARD_A = '879739599'

LOG_MARKER = "LOG_PICOW_RFID"

return {
 on = { devices = { IDX_DEVICE } },
 logging = { level = domoticz.LOG_INFO, marker = LOG_MARKER },
 execute = function(domoticz, device)
  domoticz.log(string.format('Device %s: statechange=%s', device.name,
device.state), domoticz.LOG_INFO)
   local uid = device.state
   if uid == UID_CARD_A then
       domoticz.log(string.format('Card UID %d is OK.', uid))
         else
       domoticz.log(string.format('Card UID %d is WRONG.', uid))
   end
 end
}
```

## Check & Additional Log

If the text of the device has not changed, no new log entry is added for the device.
This means if the same card or token UID is read, but not logged.
To ensure a new log entry is added every time a card is read, a flag VALID or NOT VALID could be added to the device text.

This solution provides therefor an additional log entry and a check of the UID is valid or not.

| Date | Data |
|---|---|
| 2023-03-20 10:42:01 | 8797395990,NOT VALID |
| 2023-03-20 10:42:01 | 8797395990 |
| 2023-03-20 10:41:49 | 879739599,VALID |
| 2023-03-20 10:41:49 | 879739599 |

## Automation Script

```
--[[
File:   rfid_uid_check.dzvents
Date:   20230320
Author: Robert W.B. Linn

:description
Handle the rfid text device state change triggered by the Pico W web server
submitting HTTP API/JSOM requests.

:log
2023-03-20 10:41:49.155 Status: dzVents: Info: Handling events for: "RFID UID",
value: "879739599"
2023-03-20 10:41:49.155 Status: dzVents: Info: LOG_PICOW_RFID: ------ Start
internal script: picow_rfid: Device: "RFID UID (VirtualSensors)", Index: 27
2023-03-20 10:41:49.155 Status: dzVents: Info: LOG_PICOW_RFID: Device RFID UID:
statechange=879739599
2023-03-20 10:41:49.155 Status: dzVents: Info: LOG_PICOW_RFID: Card UID 879739599
is VALID.
2023-03-20 10:41:49.156 Status: dzVents: Info: LOG_PICOW_RFID: ------ Finished
picow_rfid
2023-03-20 10:41:49.156 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
2023-03-20 10:41:49.204 Status: [web:8080] Incoming connection from: ::1
2023-03-20 10:41:49.246 Status: dzVents: Info: Handling events for: "RFID UID",
value: "879739599,VALID"
2023-03-20 10:41:49.246 Status: dzVents: Info: LOG_PICOW_RFID: ------ Start
internal script: picow_rfid: Device: "RFID UID (VirtualSensors)", Index: 27
2023-03-20 10:41:49.246 Status: dzVents: Info: LOG_PICOW_RFID: Device RFID UID:
statechange=879739599,VALID
2023-03-20 10:41:49.246 Status: dzVents: Info: LOG_PICOW_RFID: ------ Finished
picow_rfid
2023-03-20 10:42:01.345 Status: dzVents: Info: Handling events for: "RFID UID",
value: "8797395990"
2023-03-20 10:42:01.345 Status: dzVents: Info: LOG_PICOW_RFID: ------ Start
internal script: picow_rfid: Device: "RFID UID (VirtualSensors)", Index: 27
2023-03-20 10:42:01.345 Status: dzVents: Info: LOG_PICOW_RFID: Device RFID UID:
statechange=8797395990
2023-03-20 10:42:01.345 Status: dzVents: Info: LOG_PICOW_RFID: Card UID 8797395990
is NOT VALID.
2023-03-20 10:42:01.345 Status: dzVents: Info: LOG_PICOW_RFID: ------ Finished
picow_rfid
2023-03-20 10:42:01.346 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
2023-03-20 10:42:01.430 Status: dzVents: Info: Handling events for: "RFID UID",
value: "8797395990,NOT VALID"
2023-03-20 10:42:01.431 Status: dzVents: Info: LOG_PICOW_RFID: ------ Start
internal script: picow_rfid: Device: "RFID UID (VirtualSensors)", Index: 27
2023-03-20 10:42:01.431 Status: dzVents: Info: LOG_PICOW_RFID: Device RFID UID:
statechange=8797395990,NOT VALID
2023-03-20 10:42:01.431 Status: dzVents: Info: LOG_PICOW_RFID: ------ Finished
picow_rfid
]]--

-- Domoticz IDX of the text device state is set by the Pico W web server
local IDX_DEVICE  = 27
-- Card UID to verify
local UID_CARD_A = '879739599'
local FLAG_VALID = 'VALID'
local FLAG_NOT_VALID = 'NOT VALID'
-- Log marker
local LOG_MARKER = "LOG_PICOW_RFID"

-- Add a flag VALID or NOT VALID to the device state.
-- If the uid of the device is not changed, no new log entry is set.
-- Note the use of openurl with http api/json get request.
```

```
-- This updates the text of the device immediate, whereas dzvents updateText needs
more time.
local function UpdateDevice(domoticz, device, state)
    -- Define the new svalue with uid,state
    local svalue = string.format('%s,%s', device.text, state)
    -- Define the url for the http api/json get request
    local url =
string.format('http://localhost:8080/json.htm?type=command&param=udevice&idx=%d&nva
lue=0&svalue=%s', device.idx, svalue)
    -- Replace space by %20 in the url
    url = string.gsub(url, ' ', '%%20')
    -- Submit the http get request
    domoticz.openURL(url)
    -- Optional return the new device svalue
    return svalue
end

-- Handle two device state changes:
-- First state change triggered by the Pico W web server with the card UID only.
-- Second state change by this event after adding the flag to the UID to add as
device log entry uid,flag.
-- If the card UID only is used, no new device log entry is added.
-- By using a flag Domoticz notices a text change and adds a log entry.
return {
 on = {
     devices = { IDX_DEVICE }
 },
 logging = {
     level = domoticz.LOG_INFO, marker = LOG_MARKER
    },
 execute = function(domoticz, device)
  domoticz.log(string.format('Device %s: statechange=%s', device.name,
device.state), domoticz.LOG_INFO)
   -- Check if the device state contains the flag uid,flag
   if string.find(device.state, ",") == nil then
       -- Get the UID
       uid = device.state
            -- Check the uid for any action
       if uid == UID_CARD_A then
           domoticz.log(string.format('Card UID %d is %s.', uid, FLAG_VALID))
           UpdateDevice(domoticz, device, FLAG_VALID)
           -- Trigger any action ...
            else
           domoticz.log(string.format('Card UID %s is %s.', uid, FLAG_NOT_VALID))
           UpdateDevice(domoticz, device, FLAG_NOT_VALID)
       end
        end
 end
}
```

# TM1638 LED&KEY

## Description

This project explores how to use the 8-digit TM1638 seven segment 8 LEDs and 8 Push buttons component (TM1638 LED&KEY) with Domoticz.

**Ideas for Use**

This component offers a variety of applications with Domoticz. Just a few to mention are

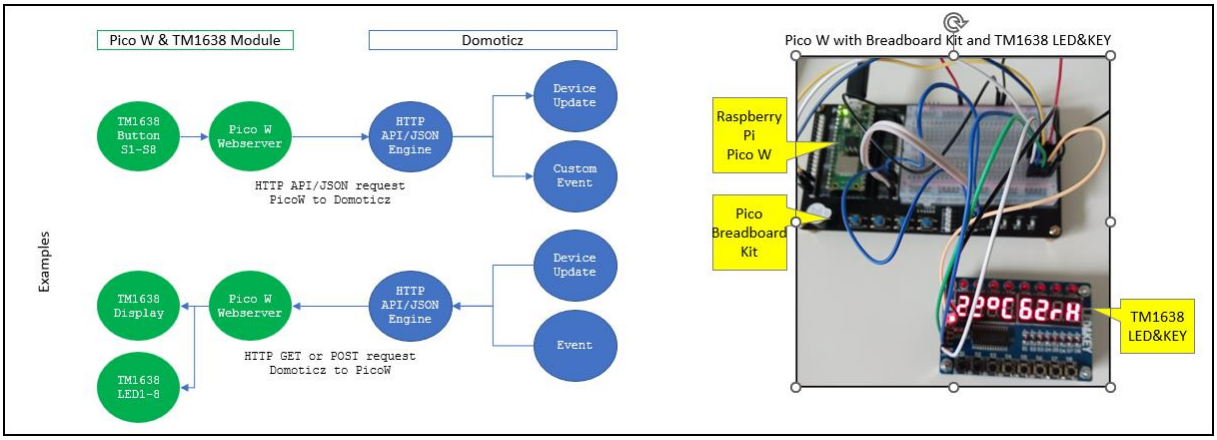
- Devices monitor using up-to 8 LEDs as indicators,
- Push buttons to set the setpoint of up-to 8 thermostats or turn light on/off,
- Segment display to display thermostat setpoint changes or temperatures or server sensor data,
- and many more...

## Solution

A two-way communication can be used, either the Pico W web server

- listens to HTTP GET or POST requests triggered by Domoticz
  (Domoticz to Pico W)
  *or*
- sends HTTP API/JSON requests to Domoticz to update a device or trigger an Automation Event, like a dzVents Custom Event
  (Pico W to Domoticz)

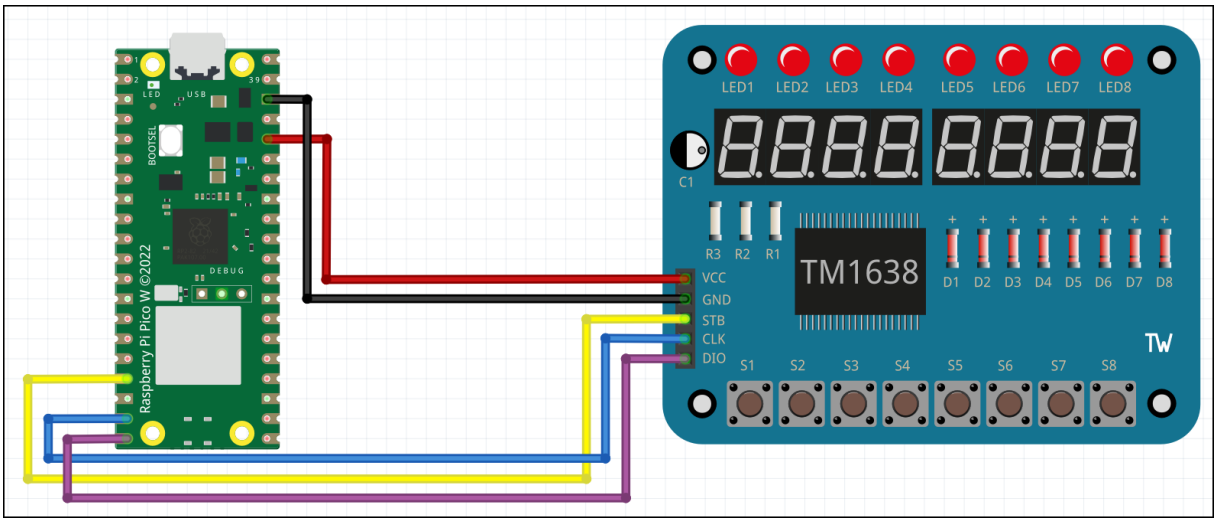
## Block Diagram



# Wiring

| TM1638 LED&KEY Module | Pico W |
|---|---|
| VCC | 3V3 (Pin #36) |
| GND | GND (Pin #38) |
| STB | GP13 (Pin #17) |
| CLK | GP14 (Pin #19) |
| DIO | GP15 (Pin #20) |

# Circuit Diagram




# Domoticz Setup

The Domoticz device(s) or Domoticz Automation Event(s) used, depend on the solution developed.

Several basic projects have been developed which can be a base for solutions using the TM1638 LED&KEY component.

# Web Server

## Libraries

The MicroPython script(s) uses a modified version (by the author of this book) of the external MicroPython TM1638 LED display driver for 8x 7-segment decimal LED components with 8x individual LEDs and 8x push buttons.
The library is stored on the Pico W in folder lib (see sub chapter Additional Libraries).

**Credit**
Thanks for developing & sharing the TM1638 LED driver.

## MicroPython Script

There are several projects developed, each having their own script.
All scripts are using the class Server from server.py (in folder lib on the Pico W) and the configuration script config.py

# Projects

Next some simple projects to explore how to use the TM1638 LED&KEY component.

## LED1-8 set by Domoticz Switch On/Off

Pico W RESTful web server listening to control LED1-8 of the TM1638LEDKEY component via Domoticz Switch.
Commands set via HTTP GET request with HTTP response JSON object.
The command can be in any case as converted to lowercase.
Communication: Domoticz to Pico W.

## MicroPython Script

```
"""
File: tm1638-ledcontrol-get.py
Date: 20230321
Author: Robert W.B. Linn

:description
Pico W RESTful web server listening to control LED1-8 of the TM1638LEDKEY module
via Domoticz Switch.
Commands set via HTTP GET request with HTTP response JSON object.
The command can be in any case as converted to lowercase.

:commands
N = LED number 1 to 8 as displayed on the module.

LED ON
HTTP Request: http://picow-ip/led/N/on
HTTP response: {"status": "OK", "title": "/led/N/on", "message": "On"}

LED OFF
HTTP Request: http://picow-ip/led/1/off
HTTP response: {"status": "OK", "title": "/led/N/off", "message": "Off"}

LED STATE
HTTP Request: http://picow-ip/led/1/state
HTTP response: {"status": "OK", "title": "/led/N/state", "message": "On"}

In case of an error:
HTTP response: {"status": "ERROR", "title": "/led/N/x", "message": "Unknown
command."}

Example using curl to turn LED1 on:
curl -v http://picow-ip/led/1/on

:log
TM1638-LEDControl-GET v20230321
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Network client connected from client-ip
HTTP Command=/led/8/on
HTTP Response={"title": "/led/8/on", "message": "on", "status": "OK"}
Network connection closed
Network client connected from client-ip
```

```
HTTP Command=/led/8/state
HTTP Response={"title": "/led/8/state", "message": "On", "status": "OK"}
Network connection closed
"""

# Libraries
import network
import socket
import time
from machine import Pin
# Server class from server.py
from server import Server
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config
# TM1638: credits to https://github.com/mcauser/micropython-tm1638
import tm1638ex

# Constants
NAME = 'TM1638-LEDControl-GET'
VERSION = 'v20230321'

# URL params to switch LED1-8 on or off or request state
CMD_LED_ON  = 'on'
CMD_LED_OFF  = 'off'
CMD_LED_STATE= 'state'

# Create the LED1 object (as indicator) using config.py settings
led1 = Pin(config.PIN_LED1, Pin.OUT)
led1.value(0)

# Create the tm1638 object with STB = GP13, CLK = GP14, DIO = GP15
tm = tm1638ex.TM1638(stb=Pin(tm1638ex.PIN_STB), clk=Pin(tm1638ex.PIN_CLK),
dio=Pin(tm1638ex.PIN_DIO))
# Turn all LEDs off
tm.leds(tm.STATE_OFF)

"""
Handle the request containing the command.
The LED is turned on/off or the state is requested.
The response JSON object is updated.

:param string cmd
    Command to set the LED1-8 state on/off or get the state.
    The command can be in any case as converted to lowercase.
    Examples: /led/8/on, /led/8/off, /led/8/state

:return JSON object response
    JSON key:value pairs: {"title": <command>, "message": <state>, "status": OK or
ERROR}
    Example: {"title": "/led/8/on", "message": "on", "status": "OK"}
"""
def handle_request(cmd):
    # Convert the command to lowercase
    cmd = cmd.lower()

    # Split the command to get the led number and state
    # /led/N/on split into 4 items: '', led, 1-8, on|off|state
    cmd_params = cmd.split('/')

    # Check if the command length is 4
    if len(cmd_params) != 4:
        response[config.KEY_MESSAGE] = f'LED command unknown ({led_cmd}).'
        response[config.KEY_STATE] = config.STATE_ERR
        # raise ValueError(f'LED command unknown ({led_cmd}).')
        return response

    # Get & check the led pos in range 0-7.
    # Note the message is for the led range 1-8 as on the module LED1-LED8
    led_pos = int(cmd_params[2]) - 1
```

```
    if not tm.LED_POS_MIN <= led_pos <= tm.LED_POS_MAX:
        response[config.KEY_MESSAGE] = f'LED position {led_pos + 1} out of range
{tm.LED_POS_MIN + 1}-{tm.LED_POS_MAX + 1}.'
        response[config.KEY_STATE] = config.STATE_ERR
        # raise ValueError(f'LED Position out of range ({led_pos}).')
        return response

    # Get & check the command on,off,state
    led_cmd = cmd_params[3]
    if not led_cmd in [CMD_LED_ON, CMD_LED_OFF, CMD_LED_STATE]:
        response[config.KEY_MESSAGE] = f'LED command unknown ({led_cmd}).'
        response[config.KEY_STATE] = config.STATE_ERR
        # raise ValueError(f'LED command unknown ({led_cmd}).')
        return response

    # Set the led on or off or get the state
    if led_cmd == CMD_LED_ON:
        tm.led(led_pos, 1)
        response[config.KEY_MESSAGE] = config.MESSAGE_ON
    if led_cmd == CMD_LED_OFF:
        tm.led(led_pos, 0)
        response[config.KEY_MESSAGE] = config.MESSAGE_OFF
    if led_cmd == CMD_LED_STATE:
        if tm.led_value(led_pos) == 1:
            response[config.KEY_MESSAGE] = config.MESSAGE_ON
        else:
            response[config.KEY_MESSAGE] = config.MESSAGE_OFF
    response[config.KEY_STATE] = config.STATE_OK
    return response

"""
Main
"""
print(f'{NAME} {VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
# Connect to the network and get the server object
server = network.connect()

while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Create the HTTP response JSON object
        response = {}

        # Parse the get data. In case of error, the status is 0.
        cmd, status = network.parse_get_request(request)

        # Assign the command to the response KEY_TITLE
        response[config.KEY_TITLE] = cmd

        # If the status is 1, handle the command
        if status == 1:
            response = handle_request(cmd)
        else:
            response[config.KEY_STATE] = config.STATE_ERR
            response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

        # Send response to the client and close the connection
        network.send_response(cl, response, True)

    except OSError as e:
        network.ledstatus.off()
        cl.close()
        print('[ERROR] Network Connection closed')
```

## Key S1-8 press sets Domoticz Alert Device

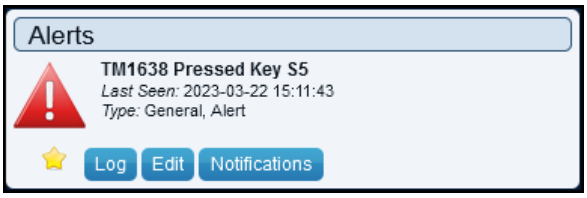
Pico W RESTful web server listening if one of the push buttons S1-S8 of the TM1638LEDKEY component is pressed.
For S1 to S5, an HTTP API/JSON request is sent to Domoticz to set the Level and Text for an Alert Device.
Example: pressing S1 is sets alert level 0 with text "TM1638 Pressed Key S1".
This is a basic example for handling push button press and submit HTTP API/JSON request to Domoticz.
Communication: Pico W to Domoticz.

## MicroPython Script

```
"""
File: tm1638-keys.py
Date: 20230321
Author: Robert W.B. Linn

:description
Pico W RESTful web server listening if one of the push buttons S1-S8 of the
TM1638LEDKEY module is pressed.
For S1 to S5, an HTTP API/JSON request is send to Domoticz to set an the Level and
Text for an Alert Device.
Example: pressing S1 is sets alert level 0 with text "TM1638 Pressed Key S1".
This is a basic example for handling push button press and submit HTTP API/JSON
request to Domoticz.

The log shows pressing keys S1 to S5 with HTTP API/JSON request updating level and
text for the Alert device with idx 7.
:log
tm1638-keyscontrol v20230321
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Send GET request url=http://domoticz-
ip:port/json.htm?type=command&param=udevice&idx=7&nvalue=0&svalue=TM1638 Pressed
Key S1
Send GET request status=OK
Send GET request url=http://domoticz-
ip:port/json.htm?type=command&param=udevice&idx=7&nvalue=1&svalue=TM1638 Pressed
Key S2
Send GET request status=OK
Send GET request url=http://domoticz-
ip:port/json.htm?type=command&param=udevice&idx=7&nvalue=2&svalue=TM1638 Pressed
Key S3
Send GET request status=OK
Send GET request url=http://domoticz-
ip:port/json.htm?type=command&param=udevice&idx=7&nvalue=3&svalue=TM1638 Pressed
Key S4
Send GET request status=OK
Send GET request url=http://domoticz-
ip:port/json.htm?type=command&param=udevice&idx=7&nvalue=4&svalue=TM1638 Pressed
Key S5
Send GET request status=OK
"""

# Libraries
import network
import socket
import time
from machine import Pin
# Server class from server.py
from server import Server
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config
```

```
# TM1638: credits to https://github.com/mcauser/micropython-tm1638
import tm1638ex

# Constants
NAME = 'tm1638-keyscontrol'
VERSION = 'v20230321'

"""
/json.htm?type=command&param=udevice&idx=IDX&nvalue=LEVEL&svalue=TEXT
IDX = id of your device (This number can be found in the devices tab in the column
"IDX")
Level = (0=gray, 1=green, 2=yellow, 3=orange, 4=red)
TEXT = Text you want to display
"""
URL_DOM_ALERT_DEVICE = 'http://' + config.DOMOTICZ_IP +
'/json.htm?type=command&param=udevice&idx={IDX}&nvalue={LEVEL}&svalue={TEXT}'
IDX_ALERT_DEVICE = 7

# Create the LED1 object (as indicator) using config.py settings
led1 = Pin(config.PIN_LED1, Pin.OUT)
led1.value(0)

# Create the tm1638 object with STB = GP13, CLK = GP14, DIO = GP15
tm = tm1638ex.TM1638(stb=Pin(tm1638ex.PIN_STB), clk=Pin(tm1638ex.PIN_CLK),
dio=Pin(tm1638ex.PIN_DIO))
# Turn all LEDs off
tm.leds(tm.STATE_OFF)

"""
Handle Request
"""
def handle_request(cmd):
    print(f'NOT USED')

"""
Main
"""
print(f'{NAME} {VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
# Connect to the network and get the server object
server = network.connect()

while True:
    # Listen to key pressed
    pressed = tm.keys()

    # Loop over the 8 keys S1-S8
    for i in range(8):
        # Check which key is pressed
        if ((pressed >> i) & 1):
            # Set the LED display with the key number
            tm.number(i+1)
            # tm.show(f'      S{i+1}')

            # Send request to Domoticz for the keys 1-5 (index 0-4)
            if 0 <= i <= 4:
                url = URL_DOM_ALERT_DEVICE.replace('{IDX}', str(IDX_ALERT_DEVICE))
                url = url.replace('{LEVEL}', str(i))
                url = url.replace('{TEXT}', f'TM1638 Pressed Key S{i + 1}')
                network.send_get_request(url)
    time.sleep(.01)
```

### Domoticz Widget Alert Device


Alerts
TM1638 Pressed Key S5
Last Seen: 2023-03-22 15:11:43
Type: General, Alert
Log
Edit
Notifications

## Key S1 press gets Domoticz Temp+Hum Device Data and set display

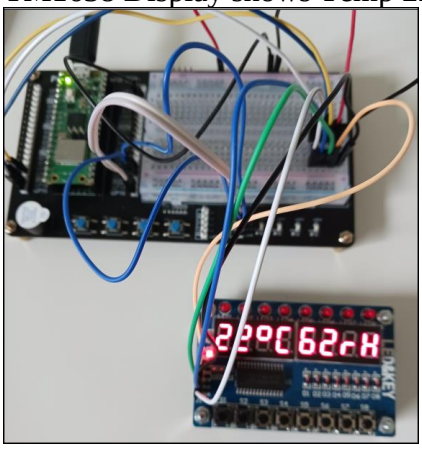
Pico W RESTful web server listening if key S1 of the TM1638LEDKEY component is pressed.
If pressed, an HTTP API/JSON request is sent to Domoticz to get the status of a Temp+Hum device.
The Domoticz server sends a JSON object as response back.
The JSON object is parsed to get the temp and hum values.
These are set on the 8-segment display i.e., 20°C54rH
Communication: Pico W to Domoticz and Domoticz to Pico W.

### MicroPython Script

```
"""
File: tm1638-keys1-temphum.py
Date: 20230321
Author: Robert W.B. Linn

:description
Pico W RESTful web server listening if key S1 of the TM1638LEDKEY module is
pressed.
If pressed, an HTTP API/JSON request is send to Domoticz to get the status of a
Temp+Hum device.
The Domoticz server sends a JSON object as response back.
The JSON object is parsed to get the temp and hum values.
These are set on the 8-segment display i.e., 20°C54rH

:log
tm1638-keys1-temphum v20230321
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Send GET request url=http://domoticz-ip:port/json.htm?type=devices&rid=15
Send GET request status=OK
Handle request status=1,json={'Sunset': '18:35', 'NautTwilightS ...
Handle request temp=20,hum=54
"""

# Libraries
import network
import socket
import time
from machine import Pin
import json
# Server class from server.py
from server import Server
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config
# TM1638: credits to https://github.com/mcauser/micropython-tm1638
import tm1638ex

# Constants
NAME = 'tm1638-keys1-temphum'
VERSION = 'v20230321'

# Define the url for the HTTP API/JSON GET request
URL_DOM = 'http://' + config.DOMOTICZ_IP + '/json.htm?type=devices&rid={IDX}'
# IDX of the Domoticz temp+hum device
IDX_DEVICE = 15

# Create the LED1 object (as indicator) using config.py settings
led1 = Pin(config.PIN_LED1, Pin.OUT)
led1.value(0)

# Create the tm1638 object with STB = GP13, CLK = GP14, DIO = GP15
```

```
tm = tm1638ex.TM1638(stb=Pin(tm1638ex.PIN_STB), clk=Pin(tm1638ex.PIN_CLK),
dio=Pin(tm1638ex.PIN_DIO))
# Turn all LEDs off
tm.leds(tm.STATE_OFF)

"""
Handle Request.
The status for the Domoticz temp+hum device is requested from the Domoticz server.
The Domoticz server sends a JSON response back.
The JSON object is parsed to get the temp and hum values.
These are set on the display i.e., 20°C,54rH
During the request handling, LED1 of the Pico W Breadboard is on, but also LED1 of
the TM1638 module.
"""
def handle_request(cmd):
    led1.value(1)
    tm.led(tm.LED1, tm.STATE_ON)
    status, content = network.send_get_request(url)
    print(f'Handle request status={status},json={content}')
    if status == 1:
        try:
            # Get key result first array entry
            result = content['result'][0]
            # print(result)
            # Get the properties Temp and Humidity
            temp = int(result['Temp'])
            hum = int(result['Humidity'])
            print(f'Handle request temp={temp},hum={hum}')
            # Set the temp + hum on the display
            tm.temperature(temp,0)
            tm.humidity(hum,4)
        except ValueError as e:
            # print(f'[ERROR] {e}, {r.content.decode()}')
            raise Exception(f'[ERROR] {e}, {r.content.decode()}')
    else:
        tm.show('ERR')
    led1.value(0)
    tm.led(tm.LED1, tm.STATE_OFF)

"""
Main
"""
print(f'{NAME} {VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
# Connect to the network and get the server object
server = network.connect()

while True:
    # Listen to key pressed
    pressed = tm.keys()

    # Loop over the 8 keys S1-S8
    for i in range(8):
        # Check which key is pressed
        if ((pressed >> i) & 1):
            key_nr = i+1
            if key_nr == 1:
                url = URL_DOM.replace('{IDX}', str(IDX_DEVICE))
                handle_request(url)
    time.sleep(.01)
```

TM1638 Display shows Temp 22°C + Humidity 62 RH



# OLED 0,96” I2C Display

## Description

This project displays and updates in regular intervals selective Domoticz data from the hardware motherboard sensors on an 0,96” I2C OLED display (OLED) connected to the Pico W.

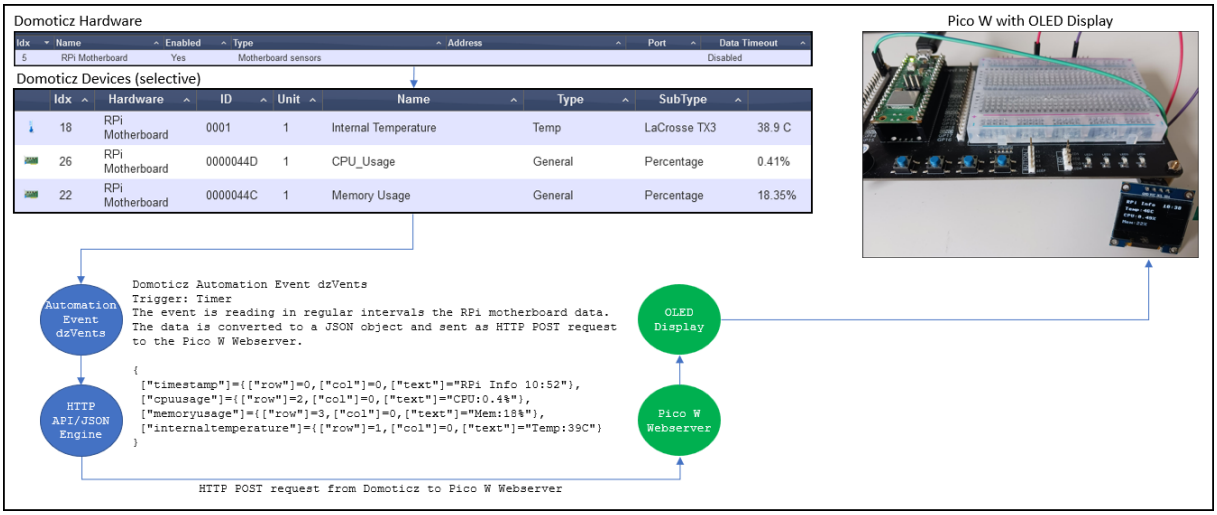
## Solution

Domoticz uses an Automation Event dzVents, triggered by a switch or timer, to send the selective Domoticz hardware motherboard sensors data (internal temperature, CPU usage, memory usage) via HTTP POST request to the Pico W web server.
The data is a JSON array with items text, row, col per OLED line.

Example:

```
{
  'timestamp': {'text': '15:53', 'col': 15, 'row': 1},
  'memoryusage': {'text': 'M:20', 'col': 14, 'row': 3},
  'cpuusage': {'text': 'C:0.39', 'col': 6, 'row': 3},
  'internaltemperature': {'text': 'T:39', 'col': 0, 'row': 3}
}
```

## Block Diagram

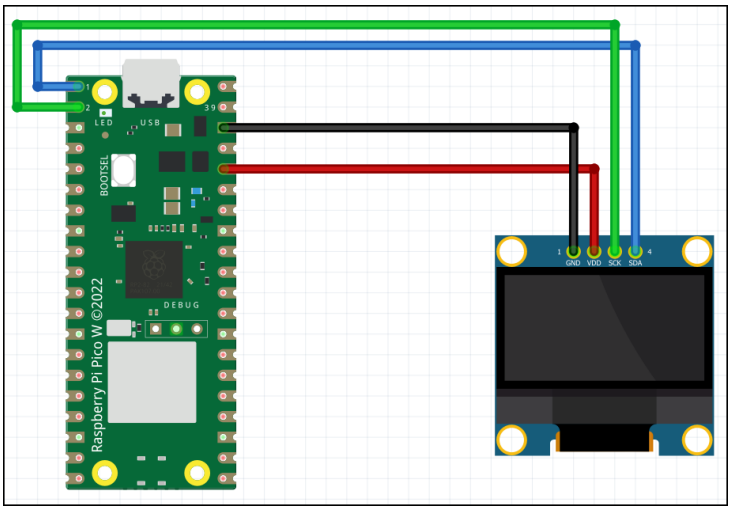



This project can be used as a base for solutions using this OLED display.

# Wiring

| **OLED** | **Pico W** |
|---|---|
| GND | GND (Pin #38) |
| VCC | 3V3 (Pin #36) |
| SDA | GP0 (Pin #1) |
| SCL | GP1 (Pin #2) |
| I2C Address | 0x3C |

# Circuit Diagram




# Domoticz Setup

## Devices

The hardware Motherboard sensors is added, and the devices Internal Temperature, CPU Usage and Memory Usage are added.
The other devices from the hardware Motherboard sensors are not used for this project.

After creating the device(s), the Domoticz devices list shows the entries:

```
IDX=26, Hardware=RPi Motherboard, ID=0000044D, Unit=1, Name=CPU Usage,
Type=General, SubType=Percentage, Data=0.43%
IDX=22, Hardware=RPi Motherboard, ID=0000044C, Unit=1, Name=Memory Usage,
Type=General, SubType=Percentage, Data=22.93%
IDX=18, Hardware=RPi Motherboard, ID=0001, Unit=1, Name=Internal Temperature,
Type=Temp, SubType=LaCrosse TX3, Data=38.4 C
```

## Automation Script

```
--[[
File:   oled_motherboard.dzvents
Date:   20230323
Author: Robert W.B. Linn

:description
Display selective motherboard information on an OLED display connected to the Pico
W.

:log
2023-03-23 10:04:00.563 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: ------
Start internal script: picow_oled_motherboard:, trigger: "every minute"
2023-03-23 10:04:00.586 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD:
{["internaltemperature"]={["col"]=0, ["row"]=1, ["text"]="Temp:44C"},
["cpuusage"]={["col"]=0, ["row"]=2, ["text"]="CPU:0.39%"},
["memoryusage"]={["col"]=0, ["row"]=3, ["text"]="Mem:22%"},
["timestamp"]={["col"]=0, ["row"]=0, ["text"]="RPi Info 10:04"}}
2023-03-23 10:04:00.587 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: ------
Finished picow_oled_motherboard
2023-03-23 10:04:00.587 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
2023-03-23 10:04:01.084 Status: dzVents: Info: Handling httpResponse-events for:
"RES_PICOW_OLED_MOTHERBOARD"
2023-03-23 10:04:01.084 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: ------
Start internal script: picow_oled_motherboard: HTTPResponse:
"RES_PICOW_OLED_MOTHERBOARD"
2023-03-23 10:04:01.084 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD:
{["title"]={["cpuusage"]={["col"]=0, ["row"]=2, ["text"]="CPU:0.39%"},
["timestamp"]={["col"]=0, ["row"]=0, ["text"]="RPi Info 10:04"},
["internaltemperature"]={["col"]=0, ["row"]=1, ["text"]="Temp:44C"},
["memoryusage"]={["col"]=0, ["row"]=3, ["text"]="Mem:22%"}}, ["status"]="OK",
["message"]=""}
2023-03-23 10:04:01.084 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: ------
Finished picow_oled_motherboard
]]--

-- Domoticz
-- For tests the trigger is an switch type onoff
local IDX_SWITCH = 16
--- IDX of the motherboard sensors (devices)
local IDX_INTERNALTEMPERATURE = 18      -- temperature
local IDX_ARMCLOCKSPEED = 19            -- sensorValue
local IDX_V3DCLOCKSPEED = 20            -- sensorValue
local IDX_CORECLOCKSPEED = 21           -- sensorValue
local IDX_MEMORYUSAGE = 22              -- percentage
local IDX_PROCESSUSAGE = 23                     -- sensorValue
local IDX_HDDBOOT = 24                          -- percentage
local IDX_HDD = 25              -- percentage
local IDX_CPUUSAGE = 26                         -- percentage

-- Round a number with digital places
local function round(num, numDecimalPlaces)
  return tonumber(string.format("%." .. (numDecimalPlaces or 0) .. "f", num))
end

-- Create table with keys col, row and text to be displayed on the OLED.
-- For an OLED row 0-3, col 0-15, text length max 16 characters
-- setText(3,0,'Hello World')
local function setText(row,col,text)
    local x = {}
    x['row']=row
    x['col']=col
    x['text']=text
  return x
end
```

```
-- Create table with sensor data for OLEDdisplay
-- setSensor(0,0,'T:',round(domoticz.devices(IDX_INTERNALTEMPERATURE).temperature,
2),'*C')
local function setSensor(row,col,pre,data,unit)
    return setText(row,col,string.format('%s%s%s', pre, tostring(data), unit))
end

-- {["internaltemperature"]={["col"]=5, ["data"]="T:39.4", ["row"]=3},
["cpuusage"]=0.68, ["coreclockspeed"]=500, ["v3dclockspeed"]=250,
["processusage"]=45.06, ["armclockspeed"]=600, ["hdd"]=38.19, ["memoryusage"]=20.9,
["hddboot"]=19.68}

local function getMotherboardData(domoticz)
    local data = {}

    -- row=0, col=0, data=HH:MM len=5, range 0+5=5
 data['timestamp'] = setText(0, 0, string.format('%s%s', 'RPi Info  ',
string.sub(domoticz.time.rawTime, 1, 5)))

    -- row=1, col=0, data=T:NN len=4, range = 0+4=4
    data['internaltemperature'] = setSensor(1, 0, 'Temp:',
round(domoticz.devices(IDX_INTERNALTEMPERATURE).temperature, 0), 'C')
    -- row=2, col=0, data=C:NN.NN len=5, range = 0+5=5
 data['cpuusage']            = setSensor(2, 0, 'CPU:',
round(domoticz.devices(IDX_CPUUSAGE).percentage, 2), '%')
    -- row=3, col=0, data=M:NN len=4, range = 0+4=4
 data['memoryusage']         = setSensor(3, 0, 'Mem:',
round(domoticz.devices(IDX_MEMORYUSAGE).percentage, 0), '%')


    --[[
    data['internaltemperature'] =
round(domoticz.devices(IDX_INTERNALTEMPERATURE).temperature, 2)
 data['armclockspeed']       =
round(domoticz.devices(IDX_ARMCLOCKSPEED).sensorValue, 0)
 data['v3dclockspeed']       =
round(domoticz.devices(IDX_V3DCLOCKSPEED).sensorValue, 0)
 data['coreclockspeed']      =
round(domoticz.devices(IDX_CORECLOCKSPEED).sensorValue, 0)
 data['memoryusage']         = round(domoticz.devices(IDX_MEMORYUSAGE).percentage,
2)
 data['processusage']        =
round(domoticz.devices(IDX_PROCESSUSAGE).sensorValue, 2)
 data['hddboot']             = round(domoticz.devices(IDX_HDDBOOT).percentage, 2)
 data['hdd']                 = round(domoticz.devices(IDX_HDD).percentage, 2)
 data['cpuusage']            = round(domoticz.devices(IDX_CPUUSAGE).percentage, 2)
    ]]--

 -- Test embedding json: r=row 0-3, c=col 0-19, d=data
    --[[
    local x = {}
    x['r']=3
    x['c']=5
    x['d']=string.format('T:%s', tostring(data['internaltemperature']))
 data['temp'] = x
    ]]--
 domoticz.log(data)
 return data
end

-- URL of the Pico W web server
local URL_SERVER      = 'http://picow-ip'

local PROJECT         = 'PICOW_OLED_MOTHERBOARD'
local RES_HTTP        = 'RES_' .. PROJECT
local LOG_MARKER      = 'LOG_' .. PROJECT

local TIMER_RULE = 'every minute'
```

```
return {
    -- Listen to switch device changes and HTTP responses
 on = { devices = { IDX_SWITCH }, timer = { TIMER_RULE }, httpResponses = {
RES_HTTP } },
 logging = { level = domoticz.LOG_INFO, marker = LOG_MARKER },

 execute = function(domoticz, item)
        -- domoticz.log(item)
        if (item.isTimer) then
            domoticz.openURL({
                url = URL_SERVER,
                method = 'POST',
                headers = { ['content-type'] = 'application/json' },
                postData = getMotherboardData(domoticz),
                callback = RES_HTTP,
               })
        end

   if (item.isDevice) then
       domoticz.log(string.format('Device %s state changed to %s', item.name,
item.state), domoticz.LOG_INFO)
            if (item.state == 'On') or (item.state == 'Off') then
                -- Submit remote HTTP POST request to set the OLED display
                domoticz.openURL({
                    url = URL_SERVER,
                    method = 'POST',
                    headers = { ['content-type'] = 'application/json' },
                    postData = getMotherboardData(domoticz),
                    callback = RES_HTTP,
                   })
            end
        end

        -- Handle HTTP response: OK is item statusCode 200 and item.ok true
        -- Else error like statusCode 7, item.ok false
        if (item.isHTTPResponse) then
            -- domoticz.log(string.format("%d %s", item.statusCode, item.ok))
            if (item.statusCode == 200) then
                if (item.isJSON) then
                    -- {["message"]="On", ["title"]="{"state": "on"}",
["status"]="OK"}
                    local data = item.json
                    domoticz.log(data)
                    -- domoticz.log(string.format("LED1 status=%s, title=%s,
message=%s", data.status, data.title, data.message))
                end
            else
                -- Error like 7 false; ERROR 7:Couldn't connect to server
                domoticz.log(string.format("ERROR %d:%s", item.statusCode,
item.statusText))
            end
        end
 end
}
```

## Domoticz Log

```
2023-03-23 10:04:00.563 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: ------
Start internal script: picow_oled_motherboard:, trigger: "every minute"
2023-03-23 10:04:00.586 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD:
{["internaltemperature"]={["col"]=0, ["row"]=1, ["text"]="Temp:44C"},
["cpuusage"]={["col"]=0, ["row"]=2, ["text"]="CPU:0.39%"},
["memoryusage"]={["col"]=0, ["row"]=3, ["text"]="Mem:22%"},
["timestamp"]={["col"]=0, ["row"]=0, ["text"]="RPi Info 10:04"}}
2023-03-23 10:04:00.587 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: ------
Finished picow_oled_motherboard
2023-03-23 10:04:00.587 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
2023-03-23 10:04:01.084 Status: dzVents: Info: Handling httpResponse-events for:
"RES_PICOW_OLED_MOTHERBOARD"
2023-03-23 10:04:01.084 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: ------
Start internal script: picow_oled_motherboard: HTTPResponse:
"RES_PICOW_OLED_MOTHERBOARD"
2023-03-23 10:04:01.084 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD:
{["title"]={["cpuusage"]={["col"]=0, ["row"]=2, ["text"]="CPU:0.39%"},
["timestamp"]={["col"]=0, ["row"]=0, ["text"]="RPi Info 10:04"},
["internaltemperature"]={["col"]=0, ["row"]=1, ["text"]="Temp:44C"},
["memoryusage"]={["col"]=0, ["row"]=3, ["text"]="Mem:22%"}}, ["status"]="OK",
["message"]=""}
2023-03-23 10:04:01.084 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: ------
Finished picow_oled_motherboard
```

# Web Server

## Libraries

The MicroPython script uses a modified version of the external OLED library MicroPython SSD1306 OLED driver, I2C and SPI interfaces.
The library is stored on the Pico W in folder lib (see sub chapter Additional Libraries).

**Parameter**

- A character has size 8px width x 16px height.
- Max char per row is 16, max rows is 4.
- The starting index for the columns and rows is 0.

**Credits**
Thanks for developing & sharing the OLED library.

## MicroPython Script

```
"""
File:oled_motherboard.py
Date:20230323
Author: Robert W.B. Linn

:description
On 0,96inch I2C OLED display connected to the Pico W, display text and selective
RPi motherboard sensor data received from Domoticz.
The Pico W runs a RESTful web server handling incoming data from a Domoticz
Automation event dzVents.
The incoming data is received from a HTTP POST request with JSON object to set the
text.
The JSON object contains for each of the displayed text, col and row.
{'sensor': {'text': 'TEXT', 'col': NN, 'row': N}, ...}
This enables to set the OLED display layout from the Domoticz event.

:log
Domoticz Motherboard v20230323
Init OLED address=60, sda=GP0, scl=GP1
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Network client connected from client-ip
HTTP Command={'timestamp': {'text': 'RPi Info  10:52', 'col': 0, 'row': 0},
'memoryusage': {'text': 'Mem:22%', 'col': 0, 'row': 3}, 'cpuusage': {'text':
'CPU:0.3%', 'col': 0, 'row': 2}, 'internaltemperature': {'text': 'Temp:44C', 'col':
0, 'row': 1}}
HTTP Response={"status": "OK", "title": {"timestamp": {"text": "RPi Info  10:52",
"col": 0, "row": 0}, "memoryusage": {"text": "Mem:22%", "col": 0, "row": 3},
"cpuusage": {"text": "CPU:0.3%", "col": 0, "row": 2}, "internaltemperature":
{"text": "Temp:44C", "col": 0, "row": 1}}, "message": ""}
Network connection closed

:wiring
OLED = Pico W
GND = GND
VCC = 3V3
SCL = GP1 (Pin #2)
SDA = GP0 (Pin #1)
"""

# Libraries
import time
from time import sleep
```

```
from machine import Pin, I2C
import json
# Call server from server.py (must be uploaded to the picow)
from server import Server
# OLED display lib stored in Pico W folder lib
import ssd1306ex
from ssd1306ex import SSD1306_I2C
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
## Name (row 0), Version (row 1), Waiting (row 3) are displayed on the OLED
NAME = 'Domoticz Motherboard'
VERSION = 'v20230323'
WAITING = 'Waiting for data...'
## Title used for the HTTP JSON response to Domoticz key title
TITLE  = 'Set OLED'

def init_oled(Pin_sda=ssd1306ex.PIN_SDA, pin_scl=ssd1306ex.PIN_SCL,
width=ssd1306ex.DISPLAY_WIDTH, height=ssd1306ex.DISPLAY_HEIGHT):
    """
    Create the OLED object. The I2C 0 is used.

    :param int pin_sda

    :param int pin_scl

    :param int width
        Width of the display, default = 128

    :param int height
        Height of the display, default = 64

    :return
        OLED object

    :example
        init_oled()
    """
    try:
        # Init I2C
        i2c = I2C(0, sda=Pin(Pin_sda), scl=Pin(Pin_scl), freq=400000)

        # OLED display with 128px width and 64px height and i2C
        oled = SSD1306_I2C(width, height, i2c)
        print(f'Init OLED address={oled.addr}, sda=GP{pin_sda}, scl=GP{pin_scl}')

        # Return the OLED object
        return oled
    except OSError as e:
        raise RuntimeError('[ERROR] Init OLED: {e}.')

def set_oled_welcome(row1, row2, row3, row4):
    """
    Initial text at col 0 on row 1 to 4.
    """
    oled.text_col_row(row1, 0, 0)
    oled.text_col_row(row2, 0, 1)
    oled.text_col_row(row3, 0, 2)
    oled.text_col_row(row4, 0, 3)
    oled.show()

def set_oled_sensor_text(data, sensor):
    """
    Set the sensor text at col, row
    : param JSON data
        JSON object with keys for col, row and text.

    :param string Sensor
```

```
        String defining the RPi motherboard sensor, i.e. internaltemperature.

    :example
        set_oled_sensor_text('internaltemperature')
    """
    # Get the sensor data
    col = data[sensor]['col']
    row = data[sensor]['row']
    text = data[sensor]['text']
    oled.text_col_row(text, col, row)

def handle_request(cmd, status):
    """
    Handle the OLED command defined as JSON object.
    The command defines for every sensor data the text and the OLED start position
col/row.
    {'timestamp': {'text': 'RPi Info  10:52', 'col': 0, 'row': 0}, 'memoryusage':
{'text': 'Mem:22%', 'col': 0, 'row': 3}, 'cpuusage': {'text': 'CPU:0.3%', 'col': 0,
'row': 2}, 'internaltemperature': {'text': 'Temp:44C', 'col': 0, 'row': 1}}

    :param JSON object
        JSON object with key:value pair {"state":"on" or "off"}

    :status
        If status is 1 set the display else unknown command

    :return JSON object response
    """
    # Assign the command to the response title
    response[config.KEY_TITLE] = cmd

    # If the status is 1 (OK) then set the OLED display with the sensor data.
    if status == 1:
        # Clear the display first
        oled.clear()
        sleep(.1)

        # Set the sensor data (subset only)
        set_oled_sensor_text(cmd, 'timestamp')
        set_oled_sensor_text(cmd, 'internaltemperature')
        set_oled_sensor_text(cmd, 'cpuusage')
        set_oled_sensor_text(cmd, 'memoryusage')
        # Show the text
        oled.show()

        # Set the response
        response[config.KEY_STATE] = config.STATE_OK
        response[config.KEY_MESSAGE] = config.MESSAGE_EMPTY
    else:
        response[config.KEY_STATE] = config.STATE_ERR
        response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

    # Return the response which is send to Domoticz
    return response

print(f'{NAME} {VERSION}')

# Create the OLED display object
oled = init_oled(ssd1306ex.PIN_SDA, ssd1306ex.PIN_SCL, ssd1306ex.DISPLAY_WIDTH,
ssd1306ex.DISPLAY_HEIGHT)

# Show initial info on the OLED. Waiting is replaced by RPi motherboard sensor data
set_oled_welcome(NAME, VERSION, '', WAITING)

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()
```

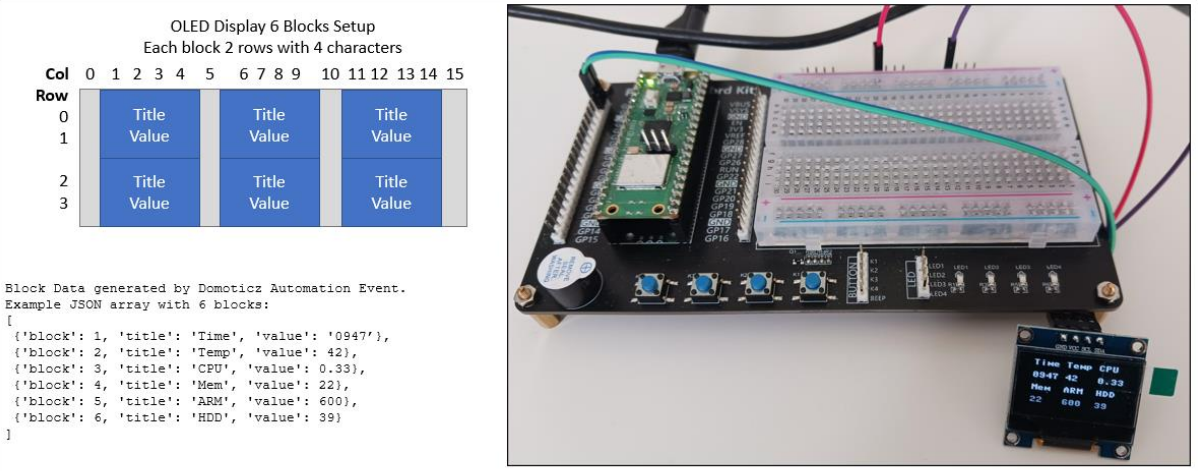
```
"""
Main Loop
"""
while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Get the cmd to set the OLED text as JSON object from the POST request
        cmd, status = network.parse_post_request(request)

        # Create the HTTP response JSON object
        response = {}

        # Handle the command to update the OLED text.
        # Set the response
        response = handle_request(cmd, status)

        # Send the response to Domoticz and close the connection (wait for new)
        network.send_response(cl, response, True)

    except OSError as e:
        ledstatus.off()
        cl.close()
        print('[ERROR] Network - Connection closed')
```

# Enhancement Ideas

## Display Blocks

Display data in 6 blocks Time, Temp, CPU, Mem, ARM, HDD with title & value for the timestamp (block 1) + Raspberry Pi selective Motherboard sensors data (blocks 2-6).

The display is segmented in 6 blocks, 3 top and 3 bottom.
Each block has two rows to show a title and value with max text length of 4 characters.




Pico W with OLED Display showing 6 blocks with RPi Motherboard sensor data

The data, a JSON array, is prepared by Domoticz Automation Event dzVents and submitted via HTTP POST request to the Pico W web server. The Pico W web server sends a response back.

**HTTP Request received from Domoticz parsed by the Pico W web server**

```
[
{'block': 1, 'title': 'Time', 'value': '1405'},
{'block': 2, 'title': 'Temp', 'value': 42},
{'block': 3, 'title': 'CPU', 'value': 0.45},
{'block': 4, 'title': 'Mem', 'value': 22},
{'block': 5, 'title': 'ARM', 'value': 600},
{'block': 6, 'title': 'HDD', 'value': 39}
]
```

**HTTP Response from the Pico W web server to Domoticz (see Domoticz Log)**

```
{
"status": "OK",
"title": [{"block": 1, "title": "Time", "value": "1405"}, {"block": 2, "title":
"Temp", "value": 42}, {"block": 3, "title": "CPU", "value": 0.45}, {"block": 4,
"title": "Mem", "value": 22}, {"block": 5, "title": "ARM", "value": 600}, {"block":
6, "title": "HDD", "value": 39}],
"message": ""
}
```

## Automation Script

```
--[[
File:   oled_blocks.dzvents
Date:   20230323
Author: Robert W.B. Linn

:description
Set raspberry pi selective motherboard data on an 0,96" I2C OLED display connected
to a Raspberry Pi Pico W ruinning as web server.
The data is submitted to the Pico W as HTTP POST request.
The OLED display has 6 block to display title:value for a motherboard sensor.

Domoticz Log
2023-03-23 14:29:36.115 VirtualSensors: Light/Switch (Pico W LED1 Control)
2023-03-23 14:29:36.109 Status: User: admin initiated a switch command (16/Pico W
LED1 Control/On)
2023-03-23 14:29:36.210 Status: dzVents: Info: Handling events for: "Pico W LED1
Control", value: "On"
2023-03-23 14:29:36.210 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: ------
Start internal script: picow_oled_blocks: Device: "Pico W LED1 Control
(VirtualSensors)", Index: 16
2023-03-23 14:29:36.210 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: Device
Pico W LED1 Control state changed to On
2023-03-23 14:29:36.216 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD:
{{["value"]="1429", ["block"]=1, ["title"]="Time"}, {["value"]=43, ["block"]=2,
["title"]="Temp"}, {["value"]=0.7, ["block"]=3, ["title"]="CPU"}, {["value"]=22,
["block"]=4, ["title"]="Mem"}, {["value"]=600, ["block"]=5, ["title"]="ARM"},
{["value"]=39, ["block"]=6, ["title"]="HDD"}}
2023-03-23 14:29:36.216 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: ------
Finished picow_oled_blocks
2023-03-23 14:29:36.217 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
2023-03-23 14:29:36.754 Status: dzVents: Info: Handling httpResponse-events for:
"RES_PICOW_OLED_MOTHERBOARD"
2023-03-23 14:29:36.754 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: ------
Start internal script: picow_oled_blocks: HTTPResponse:
"RES_PICOW_OLED_MOTHERBOARD"
2023-03-23 14:29:36.755 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD:
{["status"]="OK", ["message"]="", ["title"]={{["value"]="1429", ["block"]=1,
["title"]="Time"}, {["value"]=43, ["block"]=2, ["title"]="Temp"}, {["value"]=0.7,
["block"]=3, ["title"]="CPU"}, {["value"]=22, ["block"]=4, ["title"]="Mem"},
{["value"]=600, ["block"]=5, ["title"]="ARM"}, {["value"]=39, ["block"]=6,
["title"]="HDD"}}}
2023-03-23 14:29:36.755 Status: dzVents: Info: LOG_PICOW_OLED_MOTHERBOARD: ------
Finished picow_oled_blocks
]]--

-- URL of the Pico W web server
local URL_PICOW     = 'picow-ip'
local PROJECT       = 'PICOW_OLED_MOTHERBOARD'
local RES_HTTP      = 'RES_' .. PROJECT
local LOG_MARKER    = 'LOG_' .. PROJECT
local TIMER_RULE    = 'every minute'

-- Domoticz
-- For tests the trigger is an switch type onoff
local IDX_SWITCH = 16
--- IDX of the motherboard sensors (devices)
local IDX_INTERNALTEMPERATURE = 18      -- temperature
local IDX_ARMCLOCKSPEED = 19            -- sensorValue
local IDX_V3DCLOCKSPEED = 20            -- sensorValue
local IDX_CORECLOCKSPEED = 21           -- sensorValue
local IDX_MEMORYUSAGE = 22       -- percentage
local IDX_PROCESSUSAGE = 23             -- sensorValue
local IDX_HDDBOOT = 24                          -- percentage
local IDX_HDD = 25                        -- percentage
local IDX_CPUUSAGE = 26                   -- percentage
```

```
-- Round a number with digital places
local function round(num, numDecimalPlaces)
  return tonumber(string.format("%." .. (numDecimalPlaces or 0) .. "f", num))
end

-- Create table with keys block, title, value to be displayed on the OLED.
-- Block numbers are 1 to 6, title and value length max 4 characters
-- setBlock(1,'Title', 123)
local function setBlock(block, title, value)
    local x = {}
    x['block']=block
    x['title']=title
    x['value']=value
 return x
end

-- Create a json array with sensor data to display on the oled in block 1-6
--- {["title"]={{["block"]=1, ["title"]="Time", ["value"]="1353"}, {["block"]=2,
["title"]="Temp", ["value"]=42}, {["block"]=3, ["title"]="CPU", ["value"]=0.48},
{["block"]=4, ["title"]="Mem", ["value"]=22}, {["block"]=5, ["title"]="ARM",
["value"]=600}, {["block"]=6, ["title"]="HDD", ["value"]=39}}, ["status"]="OK",
["message"]=""}
local function setMotherboardData(domoticz)
    local data = {}
    t = string.sub(domoticz.time.rawTime, 1, 5)
    t = string.gsub(t, ':','')
 data[1] = setBlock(1, 'Time', t)
    data[2] = setBlock(2, 'Temp',
round(domoticz.devices(IDX_INTERNALTEMPERATURE).temperature, 0))
 data[3] = setBlock(3, 'CPU', round(domoticz.devices(IDX_CPUUSAGE).percentage, 2))
 data[4] = setBlock(4, 'Mem', round(domoticz.devices(IDX_MEMORYUSAGE).percentage,
0))
 data[5] = setBlock(5, 'ARM',
round(domoticz.devices(IDX_ARMCLOCKSPEED).sensorValue, 0))
 data[6] = setBlock(6, 'HDD', round(domoticz.devices(IDX_HDD).percentage, 0))

    --[[
    data['internaltemperature'] =
round(domoticz.devices(IDX_INTERNALTEMPERATURE).temperature, 2)
 data['armclockspeed']       =
round(domoticz.devices(IDX_ARMCLOCKSPEED).sensorValue, 0)
 data['v3dclockspeed']       =
round(domoticz.devices(IDX_V3DCLOCKSPEED).sensorValue, 0)
 data['coreclockspeed']      =
round(domoticz.devices(IDX_CORECLOCKSPEED).sensorValue, 0)
 data['memoryusage']         = round(domoticz.devices(IDX_MEMORYUSAGE).percentage,
2)
 data['processusage']        =
round(domoticz.devices(IDX_PROCESSUSAGE).sensorValue, 2)
 data['hddboot']             = round(domoticz.devices(IDX_HDDBOOT).percentage, 2)
 data['hdd']                 = round(domoticz.devices(IDX_HDD).percentage, 2)
 data['cpuusage']            = round(domoticz.devices(IDX_CPUUSAGE).percentage, 2)
    ]]--
 domoticz.log(data)
 return data
end

local function submitRequest(domoticz)
    domoticz.openURL({
        url = URL_PICOW, method = 'POST',
        headers = { ['content-type'] = 'application/json' },
        postData = setMotherboardData(domoticz), callback = RES_HTTP,
       })
end

return {
 on = { devices = { IDX_SWITCH }, timer = { TIMER_RULE }, httpResponses = {
RES_HTTP } },
```

```
  logging = { level = domoticz.LOG_INFO, marker = LOG_MARKER },
  execute = function(domoticz, item)
    if (item.isTimer) then submitRequest(domoticz) end
    if (item.isDevice) then
        domoticz.log(string.format('Device %s state changed to %s', item.name,
item.state))
        if (item.state == 'On') or (item.state == 'Off') then
submitRequest(domoticz) end
    end
    -- Handle HTTP response: OK is item statusCode 200 and item.ok true
    -- Else error like statusCode 7, item.ok false
    if (item.isHTTPResponse) then
      -- domoticz.log(string.format("%d %s", item.statusCode, item.ok))
      if (item.statusCode == 200) then
        if (item.isJSON) then
          -- {["message"]="On", ["title"]="{"state": "on"}", ["status"]="OK"}
          local data = item.json
          domoticz.log(data)
        end
      else
        -- Error like 7 false; ERROR 7:Couldn't connect to server
        domoticz.log(string.format("ERROR %d:%s", item.statusCode, item.statusText))
      end
    end
  end
}
```

## MicroPython Script

```
"""
File:oled_motherboard_block.py
Date:20230324
Author: Robert W.B. Linn

:description
On 0,96inch I2C OLED display connected to the Pico W, display text and selective
RPi motherboard sensor data received from Domoticz.
The Pico W runs a RESTful web server handling incoming data from a Domoticz
Automation event dzVents.
The incoming data is received from a HTTP POST request with JSON object to set the
text.
The JSON object contains for each of the displayed sensor data the block number,
title and value.
[{'block': 1, 'title': 'Time', 'value': '1403'}, {'block': 2, 'title': 'Temp',
'value': 42}, {'block': 3, 'title': 'CPU', 'value': 0.37}, {'block': 4, 'title':
'Mem', 'value': 22}, {'block': 5, 'title': 'ARM', 'value': 600}, {'block': 6,
'title': 'HDD', 'value': 39}]
This enables to set the OLED display layout from the Domoticz event.

:log
Domoticz Motherboard v20230323
Init OLED address=60, sda=GP0, scl=GP1
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Network client connected from client-ip
HTTP Command=[{'block': 1, 'title': 'Time', 'value': '1405'}, {'block': 2, 'title':
'Temp', 'value': 42}, {'block': 3, 'title': 'CPU', 'value': 0.45}, {'block': 4,
'title': 'Mem', 'value': 22}, {'block': 5, 'title': 'ARM', 'value': 600}, {'block':
6, 'title': 'HDD', 'value': 39}]
HTTP Response={"status": "OK", "title": [{"block": 1, "title": "Time", "value":
"1405"}, {"block": 2, "title": "Temp", "value": 42}, {"block": 3, "title": "CPU",
"value": 0.45}, {"block": 4, "title": "Mem", "value": 22}, {"block": 5, "title":
"ARM", "value": 600}, {"block": 6, "title": "HDD", "value": 39}], "message": ""}
Network connection closed

:wiring
OLED = Pico W
GND = GND
VCC = 3V3
SCL = GP1 (Pin #2)
SDA = GP0 (Pin #1)
"""

# Libraries
import time
from time import sleep
from machine import Pin, I2C
import json
# Call server from server.py (must be uploaded to the picow)
from server import Server
# OLED display lib stored in Pico W folder lib
import ssd1306ex
from ssd1306ex import SSD1306_I2C
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
## Name (row 0), Version (row 1), Waiting (row 3) are displayed on the OLED
NAME = 'Domoticz Motherboard'
VERSION = 'v20230324'
WAITING = 'Waiting for data...'
## Title used for the HTTP JSON response to Domoticz key title
TITLE  = 'Set OLED'
```

```
def init_oled(Pin_sda=ssd1306ex.PIN_SDA, pin_scl=ssd1306ex.PIN_SCL,
width=ssd1306ex.DISPLAY_WIDTH, height=ssd1306ex.DISPLAY_HEIGHT):
    """
    Create the OLED object. The I2C 0 is used.

    :param int pin_sda

    :param int pin_scl

    :param int width
        Width of the display, default = 128

    :param int height
        Height of the display, default = 64

    :return
        OLED object

    :example
        init_oled()
    """
    try:
        # Init I2C
        i2c = I2C(0, sda=Pin(Pin_sda), scl=Pin(Pin_scl), freq=400000)

        # OLED display with 128px width and 64px height and i2C
        oled = SSD1306_I2C(width, height, i2c)
        print(f'Init OLED address={oled.addr}, sda=GP{pin_sda}, scl=GP{pin_scl}')

        # Return the OLED object
        return oled
    except OSError as e:
        raise RuntimeError('[ERROR] Init OLED: {e}.')

def handle_request(cmd, status):
    """
    Handle the OLED command defined as JSON object.
    The command defines for every sensor data the text and the OLED start position
col/row.
    {'timestamp': {'text': 'RPi Info  10:52', 'col': 0, 'row': 0}, 'memoryusage':
{'text': 'Mem:22%', 'col': 0, 'row': 3}, 'cpuusage': {'text': 'CPU:0.3%', 'col': 0,
'row': 2}, 'internaltemperature': {'text': 'Temp:44C', 'col': 0, 'row': 1}}

    :param JSON object
        JSON object with key:value pair {"state":"on" or "off"}

    :status
        If status is 1 set the display else unknown command

    :return JSON object response
    """
    # Assign the command to the response title
    response[config.KEY_TITLE] = cmd

    # If the status is 1 (OK) then set the OLED display with the sensor data.
    if status == 1:
        # Clear the display first
        oled.clear()
        sleep(.1)

        # Set the sensor data in the text blocks 1-6
        for item in cmd:
            oled.text_block(item['block'],item['title'],item['value'])

        # Show the text blocks
        oled.show()

        # Set the response
        response[config.KEY_STATE] = config.STATE_OK
```

```
        response[config.KEY_MESSAGE] = config.MESSAGE_EMPTY
    else:
        response[config.KEY_STATE] = config.STATE_ERR
        response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

    # Return the response which is send to Domoticz
    return response

print(f'{NAME} {VERSION}')

# Create the OLED display object
oled = init_oled(ssd1306ex.PIN_SDA, ssd1306ex.PIN_SCL, ssd1306ex.DISPLAY_WIDTH,
ssd1306ex.DISPLAY_HEIGHT)

# Show initial info on the OLED. Waiting is replaced by RPi motherboard sensor data
oled.text_rows(NAME, VERSION, '', WAITING)

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()

"""
Main Loop
"""
while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Get the cmd to set the OLED text as JSON object from the POST request
        cmd, status = network.parse_post_request(request)

        # Create the HTTP response JSON object
        response = {}

        # Handle the command to update the OLED text.
        # Set the response
        response = handle_request(cmd, status)

        # Send the response to Domoticz and close the connection (wait for new)
        network.send_response(cl, response, True)

    except OSError as e:
        ledstatus.off()
        cl.close()
        print('[ERROR] Network - Connection closed')
```

# PIR Motion Sensor

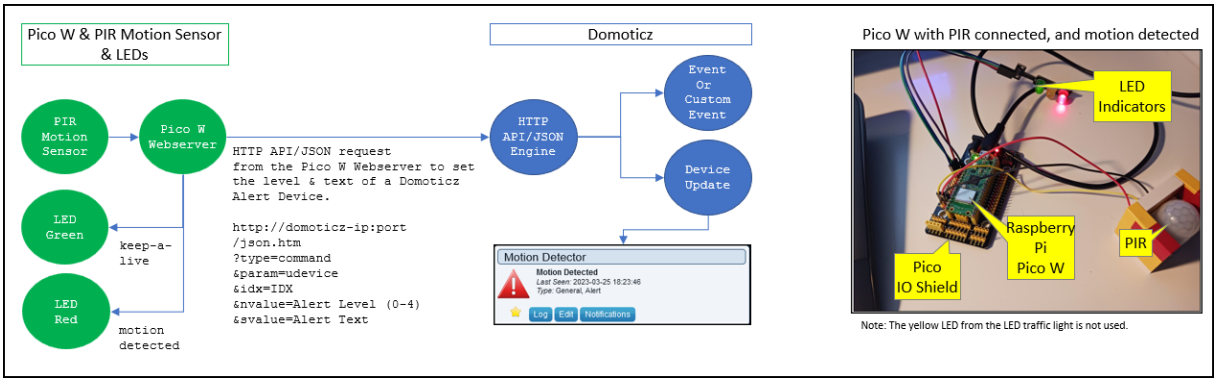
## Description

This project detects motion and sends a message to a Domoticz Alert sensor.

## Solution

The Pico W has a Passive Infrared (PIR) sensor connected and listens to an interrupt assigned to the PIR sensor signal pin. The Pico W runs as a web server.
If a motion is detected, an HTTP API/JSON request is sent to Domoticz to set for an Alert Device the Level to 4 and the Text “Motion Detected”.
Also, a RED LED is blinking for a second.
In addition, a GREEN LED is blinking every 5 seconds to indicate the motion detector is working.

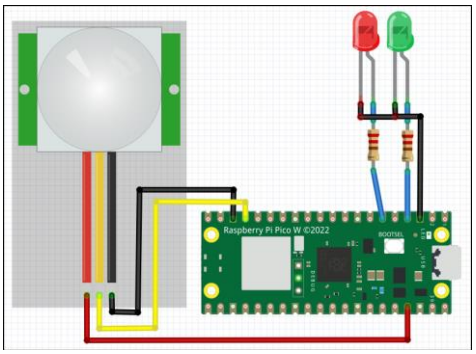
## Block Diagram



# Wiring

| **PIR** | **Pico W** |
|---|---|
| VCC | 3V3 (Pin #36) |
| Signal | GP13 (Pin #1) |
| GND | GND (Pin #38) |
| **LED RED** | |
| + | GP2 (Pin #4) |
| **LED GREEN** | |
| + | GP4 (Pin #6) |

# Circuit Diagram




# Domoticz Setup

## Devices

From the hardware “Dummy”, create an Alert device:
Name: Motion Detector, Sensor Type: Alert

After creating the device, the Domoticz devices list shows the entry:

```
IDX=7, Hardware=VirtualSensors, ID=82007, Unit=1, Name=Motion Detector,
Type=General, SubType=Alert, Data=Motion Detected
```

## Automation Script

No automation event used, but if additional action(s) required in case motion is detected, then for example an Event with trigger “Device Change” could be used.

```
--[[
File: pir_motion_sensor.dzvents
Date: 20230326
Author: Robert W.B. Linn

:description
Listen to PIR motion device changes
]]--

-- Alert device
IDX_ALERT = 7

return {
    -- Listen to device changes.
 on = { devices = { IDX_ALERT } },
 logging = { level = domoticz.LOG_INFO, marker = 'PIR', },
 execute = function(domoticz, device)
  domoticz.log(string.format('Device %s - Motion detected.', device.name))
 end
}
```

# Web Server

## Libraries

No dedicated PIR library used.

## MicroPython Script

```
"""
File:pir-motion-detection.py
Date:20230325
Author: Robert W.B. Linn

:description
Pico W RESTful web server listening if a motion is detected.
If a motion is detected, an HTTP API/JSON request is send to Domoticz to set an the
Level and Text for an Alert Device.
Also a RED LED is blinking for a sec. In addition a GREEN LED is on every 5 seconds
to indicate the motion detector is working.
Example: Motion detected sets alert level 4 with text "Motion detected".

:log
pir-motion-detection v20230326
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Motion detected!
Send GET request url=http://domoticz-
ip:port/json.htm?type=command&param=udevice&idx=7&nvalue=4&svalue=Motion Detected
Send GET request status=OK
"""

# Libraries
import network
import socket
import time
from machine import Pin
# Server class from server.py
from server import Server
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
NAME = 'pir-motion-detection'
VERSION = 'v20230326'

# Domoticz alert sensor
URL_DOM_ALERT_DEVICE = 'http://' + config.DOMOTICZ_IP +
'/json.htm?type=command&param=udevice&idx={IDX}&nvalue={LEVEL}&svalue={TEXT}'
IDX_ALERT_DEVICE = 7
ALERT_LEVEL = 4
ALERT_TEXT = 'Motion Detected'

# Motion detection duration (red led blinking): 1s = 10 cycles a 100ms
MOTION_DETECTION_DURATION = 10

# Create LED objects
# LED RED GPIO2 indicated motion detected (blinking)
led_red = machine.Pin(2, machine.Pin.OUT)
led_red.value(0)
# LED GREEN GPIO4 indicates motion detection process running
led_green = machine.Pin(4, machine.Pin.OUT)
led_green.value(0)

# Set GPIO13 PIR_Interrupt as input
sensor_pir=Pin(13, Pin.IN, Pin.PULL_UP)
```

```
# Handle motion detection triggered by the interrupt
def pir_handler(Pin):
    print("Motion detected!")
    url = URL_DOM_ALERT_DEVICE
    url = url.replace('{IDX}', str(IDX_ALERT_DEVICE))
    url = url.replace('{LEVEL}', str(ALERT_LEVEL))
    url = url.replace('{TEXT}', str(ALERT_TEXT))

    network.send_get_request(url)

    # Let red led blink for a sec
    for i in range(MOTION_DETECTION_DURATION):
        led_red.toggle()
        time.sleep_ms(100)
    led_red.value(0)

# Attach external interrupt to GPIO13 and rising edge as an external event source
sensor_pir.irq(trigger=machine.Pin.IRQ_RISING, handler=pir_handler)

"""
Handle Request
"""
def handle_request(cmd):
    print(f'NOT USED')

"""
Main
"""
print(f'{NAME} {VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
# Connect to the network and get the server object
server = network.connect()

while True:
    led_green.toggle()
    time.sleep(5)
```

# BMP280 Temperature + Barometer

## Description

This project reads in regular intervals, from a Bosch BMP280 environmental sensor, the temperature (°C) & barometric pressure (hPa) and sends the data to a Domoticz Temp+Baro device.

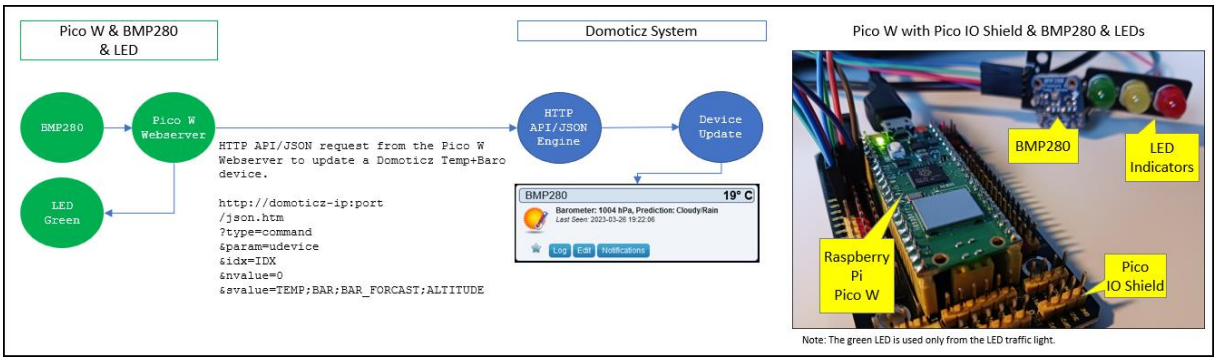
**Ideas for Use**

- Mini weather station,
- Room temperature control.

## Solution

The Pico W has a BMP280 pressure & temperature sensor and LED connected and runs as a web server.
The web server reads in regular intervals the temperature & barometric pressure and sends the data via Domoticz HTTP API/JSON request to a Domoticz Temp+Baro device.

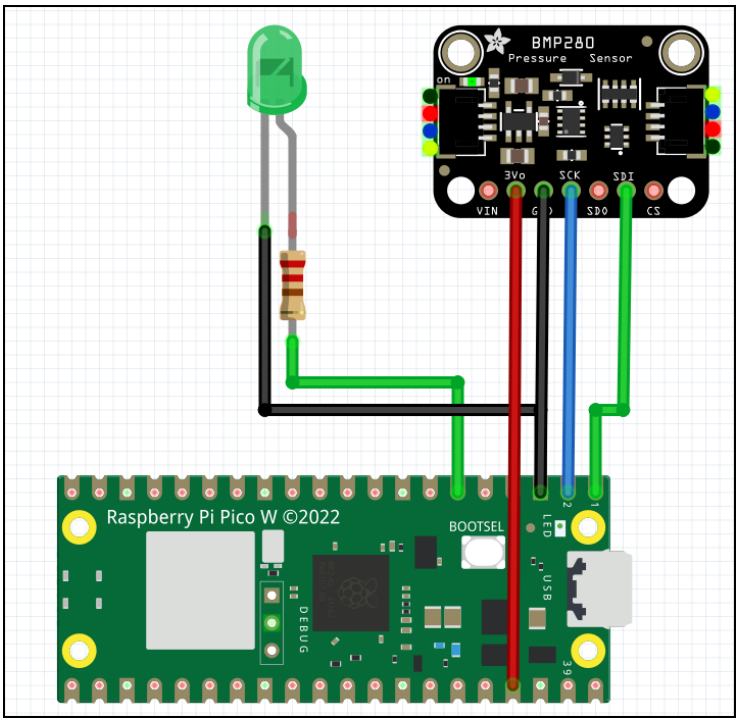
## Block Diagram



# Wiring

| **BMP280** | **Pico W** |
|---|---|
| VCC | 3V3 (Pin #36) |
| GND | GND (Pin #28) |
| SDI | GP0 (Pin #1) |
| SCK | GP1 (Pin #2) |
| | |
| **LED (green)** | **Pico W** |
| + (Anode) | GP4 (Pin #6) |
| GND (Cathode) | GND (Pin #38) |

# Circuit Diagram




# Domoticz Setup

## Devices

From the hardware “Dummy”, create a Temp+Baro device:
Name: BMP280, Sensor Type: Temp+Baro

After creating the device, the Domoticz devices list shows the entry:

```
IDX=29, Hardware=VirtualSensors, ID=1406D, Unit=1, Name=BMP280, Type=Temp + Baro,
SubType=BMP085 I2C, Data=32.0 C, 1004.0 hPa
```

## Automation Script

No automation event used, but if additional action(s) required in case device has changed, then for example an Event with trigger “Device Change” be used.

```
--[[
File:   bmp280.dzvents
Date:   20230418
Author: Robert W.B. Linn

:description
Listen to bmp280 device changes. No action defined.
]]--

-- BMP280 device IDX
IDX_BMP280 = 29

return {
  -- Listen to device changes.
 on = { devices = { IDX_BMP280 } },
 logging = { level = domoticz.LOG_INFO, marker = 'BMP280', },
 execute = function(domoticz, device)
     -- domoticz.log(device)
   domoticz.log(string.format('Device %s has changed. New state %s', device.name,
device.state))
 end
}
```

# Web Server

## Libraries

The MicroPython script uses external the MicroPython BMP280 library.
The library is stored on the Pico W in folder lib (see sub chapter Additional Libraries).

**Credits**
Thanks for developing & sharing the micropython-bmp280 library.

## MicroPython Script

```
"""
File:   bmp280.py
Date:   20230327
Author: Robert W.B. Linn

:description
Read in regular intervals the BMP280 temperature and barometric pressure and update
a Domoticz Temp+Baro device.
The Domoticz devices are updated using HTTP API/JSON POST request Custom Event to
the Domoticz server.

:log
BMP280 v20230326
Sampling Rate: 60s.
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
BMP280 t=24,p=101340.5,hpa=1013,bar=1.013404,mmhg=760.1157,svalue=24;1013;0;0
Send GET request url=http://domoticz-
ip:port/json.htm?type=command&param=udevice&idx=29&nvalue=0&svalue=24;1013;0;0
Send GET request status=OK
"""

# Imports
from machine import Pin,I2C
from utime import sleep
# BMP280
from bmp280 import *
# Call server from server.py (must be uploaded to the picow)
from server import Server
# Configuration (must be uploaded to the picow)
import config

# Constants
VERSION = 'BMP280 v20230326'

# Create the led object (GP4, Pin #6) indicating bmp280 measurement in progress
led_green = Pin(4, Pin.OUT)
led_green.value(0)

# BMP280 SCK & SDA Pins and address
BMP_PIN_SDI = 0
BMP_PIN_SCK = 1
BMP_ADDR = 0x77
# BMP280 measurement sampling rate in seconds
SAMPLING_RATE = 60
# BMP280 IDX of the Domoticz Temp+Baro device
IDX_TEMP_BARO = 29
# URL Domoticz
# Note the idx of the domoticz device ( see GUI > Setup > Devices)
# The svalue is added in the main loop after getting the data from the BMP280.
# The svalue format: TEMP;BAR;BAR_FOR;ALTITUDE
```

```
URL_DOM = "http://"+ config.DOMOTICZ_IP
+"/json.htm?type=command&param=udevice&idx={IDX}&nvalue=0&svalue={SVALUE}"

# Create the bmp280 sensor object
# Init the bus with GP0 (SDA) and GP1 (CLK)
bus = I2C(0, scl=Pin(BMP_PIN_SCK), sda=Pin(BMP_PIN_SDI), freq=200000)
# Create the bmp280 object with address 0x77. The default address 0x76 gives error
OSError: [Errno 5] EIO
bmp = BMP280(bus, addr=BMP_ADDR)
# The use case is indoor
bmp.use_case(BMP280_CASE_INDOOR)

"""
Set the barometer forecast depending pressure.

:param int pressure

:return int forecast
    0=Stable,1=Sunny,2=Cloudy,3=Unstable,4=Thunderstorm,5=Unknown,6=Cloudy/Rain
"""
def barometer_forecast(pressure):
    if pressure < 966:
        return 4 # THUNDERSTORM
    elif pressure < 993:
        return 2 # CLOUDY
    elif pressure < 1007:
        return 6 # PARTLYCLOUDY
    elif pressure < 1013:
        return 3 # UNSTABLE
    elif pressure < 1033:
        return 0 # STABLE
    else:
        return 5 # Unknown

"""
BMP280 measurement with rounded values for temperature and pressure.
During measurement, the green led is on.

:return string svalue
    svalue with TEMP;BAR;BAR_FOR;ALTITUDE

:example svalue
    19;1004;6;0
"""
def get_bmp280_data():
    led_green.value(1)
    sleep(1)
    # print(f'BMP280 measuring...')
    temperature = round(bmp.temperature)
    pressure    = bmp.pressure
    p_pa   = pressure
    p_hpa  = round(pressure/100)
    p_bar  = pressure/100000
    p_mmHg  = pressure/133.3224
    forecast = barometer_forecast(p_hpa)

    # Set the svalue, i.e. svalue=TEMP;BAR;BAR_FOR;ALTITUDE
    svalue = str(temperature) + ';' + str(p_hpa) + ';' + str(forecast) + ';' +
str(0)
    print(f"BMP280
t={temperature},p={p_pa},hpa={p_hpa},bar={p_bar},mmhg={p_mmHg},svalue={svalue}")

    # Return the svalue
    led_green.value(0)
    return svalue

# Info
print(f'{VERSION}')
print(f'Sampling Rate: {SAMPLING_RATE}s.')
```

```
# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()

# Domoticz url
url = URL_DOM
url = url.replace('{IDX}', str(IDX_TEMP_BARO))

# Main
while True:
    # Measure & submit BMP280 data to Domoticz
    network.send_get_request(url = url.replace('{SVALUE}', get_bmp280_data()))
    # Delay till next sample
    sleep(SAMPLING_RATE)
```

# Potentiometer Dimmer

## Description

This project enables to set the level 0-100% of a Domoticz Dimmer switch via potentiometer connected to the Pico W running as a web server.

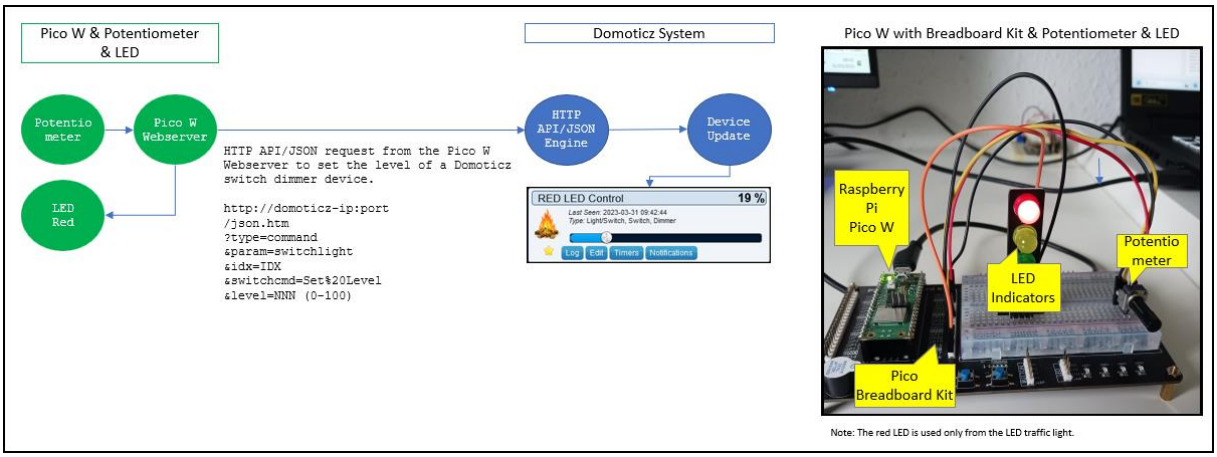
**Ideas for Use**

- Control lights

## Solution

The Pico W has a potentiometer and LED connected and runs as a web server.
The web server listens to potentiometer signal changes and sends the converted potentiometer signal via Domoticz HTTP API/JSON request to a Domoticz dimmer device (Type Light/Switch, Subtype Switch) to set the level between 0-100%.

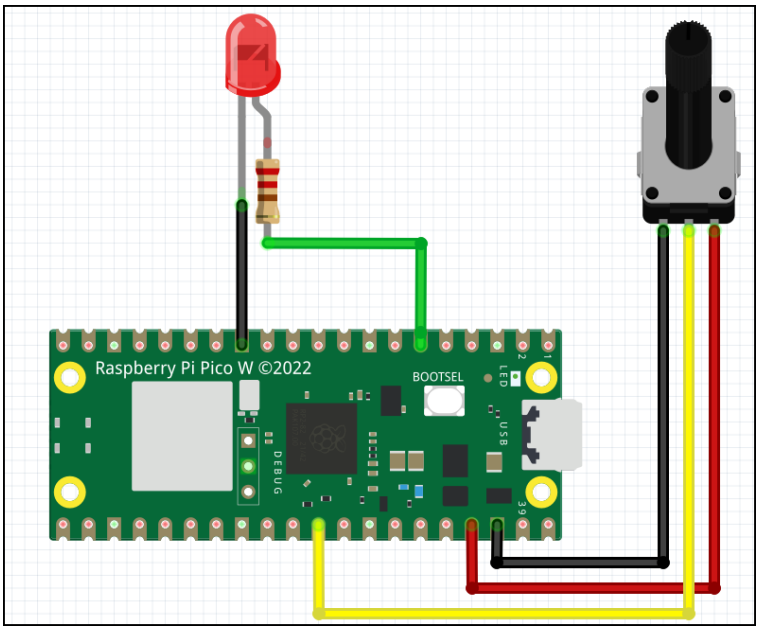
## Block Diagram



# Wiring

| **Potentiometer** | **Pico W** |
|---|---|
| VCC | 3V3 (Pin #36) |
| GND | GND (Pin #28) |
| Signal | GP26 (Pin #31, ADC0) |
| | |
| **LED (red)** | **Pico W** |
| + (Anode) | GP16 (Pin #6) |
| GND (Cathode) | GND (Pin #21) |

# Circuit Diagram




# Domoticz Setup

## Devices

Create a virtual sensor, hardware dummy, named Dimmer from sensor type Switch/Light.

After creating the device, the Domoticz devices list shows the entry:

```
Idx=1, Hardware=VirtualSensors, ID=00014051, Unit=1, Name=Dimmer,
Type=Light/Switch, SubType=Switch, Data=Off
```

The switch type Dimmer is set in the Switch device properties (edit the widget).


## Automation Event

No automation event used, but if additional action(s) required in case device has changed, then for example an Event with trigger “Device Change” be used.

# Web Server

## Libraries

No additional libraries used, beside the server and the config script.

## MicroPython Script

```
"""
File:potmeterdimmer.py
Date:20230330
Author: Robert W.B. Linn

:description
To dim a Domoticz dimmer switch.
The Domoticz device is updated using HTTP API/JSON request to the Domoticz server.

:notes
Pico Breadboard Kit is used to wire up the potmeter.
Configuration stored in config.py, ensure to upload to the picow.

:log
PotMeterDimmer v20230330
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
value=11026, level=17, prev_level=-1, abs=18
Send GET request url=http://domoticz-
ip:port/json.htm?type=command&param=switchlight&idx=1&switchcmd=Set%20Level&level=1
7
Send GET request status=OK
value=224, level=0, prev_level=17, abs=17
Send GET request url=http://domoticz-
ip:port/json.htm?type=command&param=switchlight&idx=1&switchcmd=Set%20Level&level=0
Send GET request status=OK

:wiring
PotMeter = Pico W
VCC (+) = VBUS (Pin #40)
OUT = GP26 (Pin #31, ADC0)
GND (-) = GND (Pin #28)

LED = Pico W
+ (Anode) = GP16 (Pin #21)
"""

# Imports
from machine import Pin, ADC, PWM
from utime import sleep
# Call server from server.py (must be uploaded to the picow)
from server import Server
# Configuration (must be uploaded to the picow)
import config

# Constants
VERSION = 'PotMeterDimmer v20230330'

# Dimmer min max range using offsets
DIMMER_MAX = const(65500)  # 65535
DIMMER_MIN = const(250)          # 0

# Helper to map the PWM range to 0-100%
def mapRange(value, inMin, inMax, outMin, outMax):
    return outMin + (((value - inMin) / (inMax - inMin)) * (outMax - outMin))

# Create ADC0 object GP26
```

```
adc0 = ADC(0)

# Create PWM LED GP16 to control the dimmer level 0-100%
pwmled = PWM(Pin(16, Pin.OUT))
pwmled.freq(1000)

# IDX of the Domoticz Dimmer Switch device
IDX_DIMMER = 1
# URL Domoticz
# Note the idx of the domoticz device ( see GUI > Setup > Devices)
# /json.htm?type=command&param=switchlight&idx=99&switchcmd=Set%20Level&level=6
URL_DOM =
'http://'+config.DOMOTICZ_IP+'/json.htm?type=command&param=switchlight&idx='+str(ID
X_DIMMER)+'&switchcmd=Set%20Level&level='

# Por meter noise level 2%
NOISE_LEVEL = 2
# Keep the previous level
prev_level = -1

"""
Get the dimmer level between 0-100.

:return int level
    level between 0-100

:example
    18
"""
def set_dimmer_level():
    global prev_level

    # Read ADC0
    # Noise reduction: Not used but either LSB divide (adc0.read_u16() >> 2) or
remove (adc0.read_u16() & 0b1111111111111100)
    value = adc0.read_u16()

    # Map the potmeter range to 0-100
    level = round(mapRange(value, DIMMER_MIN, DIMMER_MAX, 0, 100))

    # Check if the abs value between the current and prev reading is greater noise
level
    if abs(prev_level - level) > NOISE_LEVEL:
        print(f'value={value}, level={level}, prev_level={prev_level},
abs={abs(level - prev_level)}')

        # Set PWM-Duty-Cycle = brightness of the control LED
        pwmled.duty_u16(value)

        # Keep the prev level
        prev_level = level

        # Submit Domoticz HTTP API/JSON GET request to update the device
        network.send_get_request(URL_DOM + str(level))
        sleep(0.5)

# Info
print(f'{VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()

# Main
while True:
    # Listen to potmeter changes & set domoticz dimmer to 0-100%
    set_dimmer_level()
```

# DS18B20 Temperature (Push)

## Description

This project reads in regular intervals the temperature (°C), from DS18B20 1-wire digital thermometer sensors and sends (push) the data to Domoticz Temperature device(s).

**Ideas for Use**

- Mini weather station,
- Room temperature control.

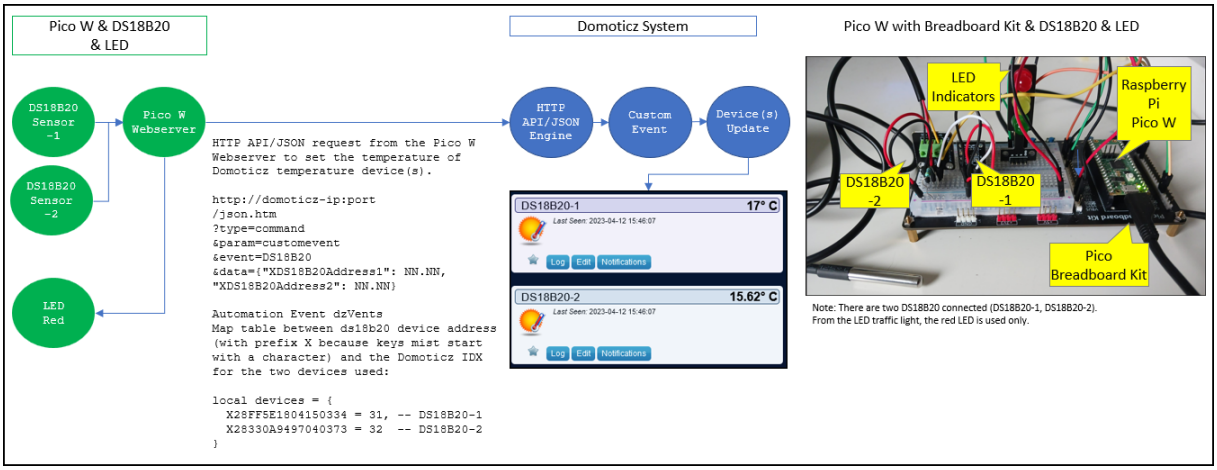
## Solution

The Pico W has two DS18B20 sensors and an LED connected and runs as a web server. The web server reads in regular intervals the temperature and sends the data via Domoticz HTTP API/JSON request to Domoticz Temperature device(s).

The two DS18B20 sensors are:

- Keyes DS18B20 sensor (address 28FF5E1804150334)
- Oumefar Digital Temperature DS18B20 sensor (address 28330A9497040373)

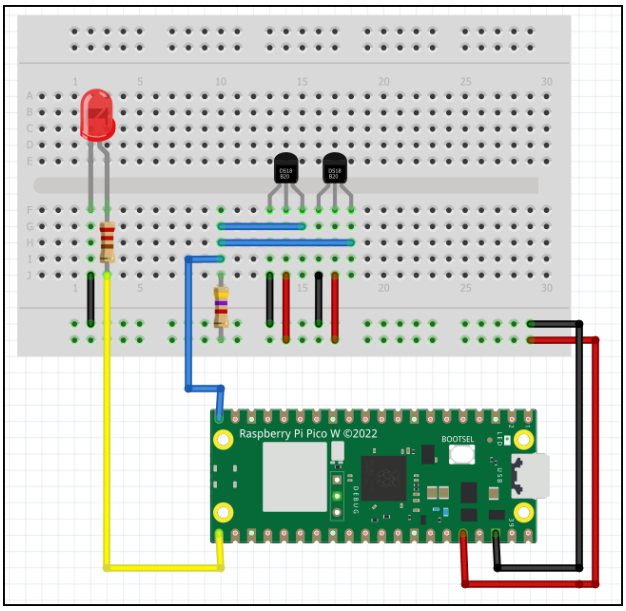
## Block Diagram



# Wiring

| **DS18B20 (2x)** | **Pico W** |
|---|---|
| VCC | 3V3 (Pin #36) |
| GND | GND (Pin #28) |
| Data | GP15 (Pin #20) |
| The wiring applies to the two DS18B20 sensors. | |
| **LED (green)** | **Pico W** |
| + (Anode) | GP4 (Pin #6) |
| GND (Cathode) | GND (Pin #38) |

# Circuit Diagram




# Domoticz Setup

## Devices

From the hardware “Dummy”, create a Temperature device:
Name: DS18B20-1, Sensor Type: Temperature
Do the same if more DS18B20 devices, i.e.
Name: DS18B20-2, Sensor Type: Temperature

After creating the device(s), the Domoticz devices list shows the entries:

```
IDX=31, Hardware=VirtualSensors, ID=1406F, Unit=1, Name=DS18B20-1, Type=Temp,
SubType=LaCrosse TX3, Data=0.0 C
IDX=32, Hardware=VirtualSensors, ID=14070, Unit=1, Name=DS18B20-2, Type=Temp,
SubType=LaCrosse TX3, Data=0.0 C
```

## Automation Script

A custom event handles updating the Domoticz temperature devices.
The event receives data as JSON array with key:value pairs:

```
device address : temperature

Example
{'X28FF5E1804150334': 16.5, 'X28330A9497040373': 15.25}
```

The DS18B20 device address has a prefix X to get handled by the custom event devices table.

```
--[[
File:   ds18b20_customevent.dzvents
Date:   20230412
Author: Robert W.B. Linn

:description
Listen to picow webserver request custom event command and update the temp devices
assigned to the ds18b20 device(s).

:log
023-04-12 15:42:03.651 Status: dzVents: Info: Handling Domoticz custom event for:
"DS18B20"
2023-04-12 15:42:03.652 Status: dzVents: Info: ------ Start internal script:
ds18b20_customevent: Custom event: "DS18B20"
2023-04-12 15:42:03.653 Status: dzVents: Info: d=X28FF5E1804150334, t=16.75
2023-04-12 15:42:03.678 Status: dzVents: Info: d=X28330A9497040373, t=15.50
2023-04-12 15:42:03.679 Status: dzVents: Info: ------ Finished ds18b20_customevent
2023-04-12 15:42:03.680 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
]]--

-- Custom event name as used by the PicoW webserver HTTP API/JSON POST request
local CUSTOM_EVENT_NAME = 'DS18B20'

-- Define map table between ds18b20 device address (with prefix X because keys must
start with a character) and the domoticz idx
local devices = {
    X28FF5E1804150334 = 31, -- DS18B20-1
    X28330A9497040373 = 32  -- DS18B20-2
}

-- Update the devices by looping over the Lua table containing address &
temperature
local function updateDevices(domoticz, data)
    for k,v in pairs(data) do
        -- d=X28FF5E1804150334, t=16.75
        -- d=X28330A9497040373, t=15.50
        domoticz.log(string.format('d=%s, t=%0.2f', k, v))
        -- Select the device and update
        domoticz.devices(devices[k]).updateTemperature(v)
    end
end

return {
 on = { customEvents = { CUSTOM_EVENT_NAME } },
 execute = function(domoticz, triggeredItem)
  if (triggeredItem.isCustomEvent) then
       -- Check the custom event name in case there are more custom events
       if (triggeredItem.trigger == CUSTOM_EVENT_NAME) then
        -- domoticz.log(triggeredItem.data)
        local data = triggeredItem.json
        updateDevices(domoticz, data)
       end
  end
```

```
  end
}
```

# Web Server

## Libraries

The MicroPython script uses the MicroPython internal libraries OneWire and DS18X20.

## MicroPython Script

```
"""
File: ds18b20_customevent.py
Date: 20230412
Author: Robert W.B. Linn

:description
Read in regular intervals the temperature of two DS18B20 devices and update the
temperature of the Domoticz devices named DS18B20-1, DS18B20-2.
Each DS18B20 has an unique 8-byte address, like 28 FF 5E 18 04 15 03 34.
The two DS18B20 sensors are:
* Keyes DS18B20 sensor (address 28FF5E1804150334)
* Oumefar Digital Temperature DS18B20 sensor (address 28330A9497040373)

The Domoticz devices are updated from a Domoticz Custom Event (dzVents) triggered
by a HTTP API/JSON CustomEvent request to the Domoticz server.
Example:
http://domoticz-
ip:port/json.htm?type=command&param=customevent&event=DS18B20&data={'X28FF5E1804150
334': 16.5, 'X28330A9497040373': 15.25}

:notes
Pico Breadboard Kit is used to wire up the DS18B20.
Configuration stored in config.py, ensure to upload to the picow.
DS18B20 measures every 60 seconds.

:log
DS18B20 v20230412
Sampling Rate: 60s.
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
One-Wire Devices found: 2
Device: 28FF5E1804150334
Device: 28330A9497040373
-----
Send POST request url=http://domoticz-
ip:port/json.htm?type=command&param=customevent&event=DS18B20&data=,
postdata={'X28FF5E1804150334': 16.5, 'X28330A9497040373': 15.25}
Send POST request status=OK

:wiring
DS18B20 = Raspberry Pi Pico W
VDD = 3V3 (Pin #36)
GND = GND (Pin #38)
Data = GP15 (Pin #20)
The wiring applies to N sensors.
"""

# Imports
from machine import Pin
from utime import sleep, sleep_ms
# The onewire and ds18x20 are micropython internal libs.
from onewire import OneWire
from ds18x20 import DS18X20
import json
# Call server from server.py (must be uploaded to the picow)
from server import Server
# Configuration (must be uploaded to the picow)
```

```
import config

# Constants
VERSION = 'DS18B20 v20230412'

# Create the led object indicating sensor measurement in progress
led_indicator = Pin(config.PIN_LED1, Pin.OUT)
led_indicator.value(0)

# DS18B20 Signal Pin GP15 #Pin 20
PIN_DS18B20 = 15
# DS18B20 measurement sampling rate in seconds
SAMPLING_RATE = 60
# URL Domoticz
URL_DOM = "http://"+ config.DOMOTICZ_IP
+"/json.htm?type=command&param=customevent&event=DS18B20&data="

# Init OneWire with the pin to which one or more DS18B20 sensors are connected
one_wire_bus = Pin(Pin_DS18B20)
# Init the DS18X20 class with constructor function
ds_sensor = DS18X20(OneWire(one_wire_bus))

"""
Read the temperature of the DS18B20 sensor(s).

:return json array
    JSON array with key:value pairs {"device address":temperature, ...}
    NOTE: The device address has prefix X to get handled by the Domoticz dzVents
custom event.
    {'X28FF5E1804150334': 16.5, 'X28330A9497040373': 15.1875}
"""
def read_ds_sensor():
    # Read & convert temperature
    ds_sensor.convert_temp()
    # Wait: min. 750 ms
    sleep_ms(750)
    # Loop over the devices to get the temperature
    result = {}
    for device in devices:
        device_address = 'X' + bytes(device).hex().upper()
        # print(f'Sensor: {device_address}')
        temperature = ds_sensor.read_temp(device)
        # print(f'Temperatur: {temperature}°C')
        # Add the device address and temperature to the json array
        result[device_address] = temperature
    # Return the json array with the data
    return result

# Info
print(f'{VERSION}')
print(f'Sampling Rate: {SAMPLING_RATE}s.')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()

# Scan for One-Wire devices
# Get list of ROM addresses for all attached slaves. Each ROM address is an 8-byte
long byte array.
devices = ds_sensor.scan()
print(f'One-Wire Devices found: {len(devices)}')
for device in devices:
    print(f'Device: {bytes(device).hex().upper()}')
print(f'-----')

# Main
# Measure every NN seconds (see constant SAMPLING_DELAY)
```

```
while True:
    led_indicator.value(1)
    # Read the sensor(s)
    data = read_ds_sensor()
    # print(f'{data}')
    # Submit Domoticz HTTP API/JSON POST request to update the devices via
customevent
    network.send_post_request(URL_DOM, data)
    led_indicator.value(0)
    # Delay till next sample
    sleep(SAMPLING_RATE)
```

# DS18B20 Temperature (Pull)

## Description

This project listens to HTTP client requests to read the temperature (°C), from DS18B20 1-wire digital thermometer sensors and sends an HTTP response with data to the client (Domoticz, Node-RED or other).

## Solution

The Pico W has two DS18B20 sensors and an LED connected and runs as a web server.

The two DS18B20 sensors are (same as previous project with Push solution):

- Keyes DS18B20 sensor (address 28FF5E1804150334)
- Oumefar Digital Temperature DS18B20 sensor (address 28330A9497040373)

The web server listens to HTTP POST requests (JSON object) to read the sensor temperature (°C).
The JSON request object has key:value pair:

- request:1 | 0 where 1 is requesting for data.

```
{'request': 1} or {'request': 0}
```

The web server sends an HTTP response back to the client containing the data (JSON object).
The JSON object structure is Domoticz like with the keys

- status: OK or ERROR
- title: the HTTP POST request
- message: the sensor data as array with fields id, temperature and address for each sensor.

```
{
 "status": "OK",
 "title": "{'request': 1}",
 "message":
   [
    {"id": 1, "temperature": 17.0, "address": "28FF5E1804150334"},
    {"id": 2, "temperature": 13.875, "address": "28330A9497040373"}
   ]
}
```

## Wiring

The wiring is described in project DS18B20 Temperature (Push).

## Circuit Diagram

The circuit diagram is described in project DS18B20 Temperature (Push).

# Web Server

## Libraries

The MicroPython script uses the MicroPython internal libraries OneWire and DS18X20.

## MicroPython Script

```
"""
File: ds18b20_client_pull.py
Date: 20230413
Author: Robert W.B. Linn

:description
The Pico W runs as a web server and listens to post request from clients (PULL).
If the post request is {"request":1} then the temperatures of the connected sensors
are read.
The data is returned to the client as JSON object in Domoticz format:
{"status": "OK", "title": "{'request': 1}", "message": [{"id": 1, "temperature":
22.0, "address": "28FF5E1804150334"}, {"id": 2, "temperature": 17.4375, "address":
"28330A9497040373"}]}
The key message contains the JSON array with the DS18B20 address and temperature.

Each DS18B20 has a unique 8-byte address, like 28 FF 5E 18 04 15 03 34.
The two DS18B20 sensors are:
* Keyes DS18B20 (address 28FF5E1804150334)
* Oumefar Digital Temperature sensor DS18B20 (address 28330A9497040373)

:example
Using curl with HTTP response JSON object.
HTTP Request
curl -v -H "Content-Type: application/json" -d "{\"request\":1}" http://picow-ip
HTTP Response
{"status": "OK", "title": "{'request': 1}", "message": [{"id": 1, "temperature":
22.0, "address": "28FF5E1804150334"}, {"id": 2, "temperature": 17.4375, "address":
"28330A9497040373"}]}

:notes
Pico Breadboard Kit is used to wire up the DS18B20.
Configuration stored in config.py, ensure to upload to the picow.
DS18B20 measures every 60 seconds.

:log
DS18B20 v20230413
Sampling Rate: 60s.
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
One-Wire Devices found: 2
Device: 28FF5E1804150334
Device: 28330A9497040373
-----
Network client connected from client-ip
HTTP Command={'request': 1}
HTTP Response={"status": "OK", "title": "{'request': 1}", "message": [{"id": 1,
"temperature": 17.0, "address": "28FF5E1804150334"}, {"id": 2, "temperature":
13.875, "address": "28330A9497040373"}]}
Network connection closed
HTTP Command={'request': 0}
HTTP Response={"status": "ERROR", "title": {"request": 0}, "message": ""}
Network connection closed
Network client connected from client-ip

:wiring
DS18B20 = Raspberry Pi Pico W
VDD = 3V3 (Pin #36)
```

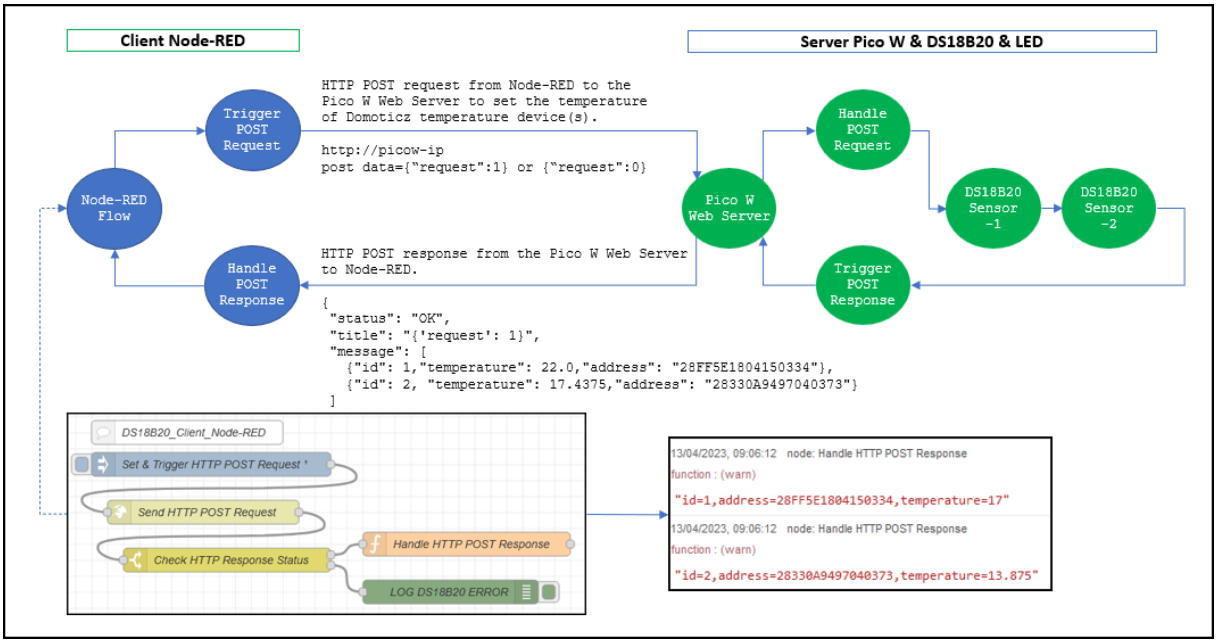
```
GND = GND (Pin #38)
Data = GP15 (Pin #20)
The wiring applies to N sensors.
"""

# Imports
from machine import Pin
from utime import sleep, sleep_ms
# The onewire and ds18x20 are micropython internal libs.
from onewire import OneWire
from ds18x20 import DS18X20
import json
# Call server from server.py (must be uploaded to the picow)
from server import Server
# Configuration (must be uploaded to the picow)
import config

# Constants
VERSION = 'DS18B20 v20230412'

# Create the led object indicating sensor measurement in progress
led_indicator = Pin(config.PIN_LED1, Pin.OUT)
led_indicator.value(0)

# DS18B20 Signal Pin GP15 #Pin 20
PIN_DS18B20 = 15

# Init OneWire with the pin to which one or more DS18B20 sensors are connected
one_wire_bus = Pin(Pin_DS18B20)
# Init the DS18X20 class with constructor function
ds_sensor = DS18X20(OneWire(one_wire_bus))

"""
Read the temperature of the DS18B20 sensor(s).

:return json array
    JSON array with key:value pairs:
    [{"id": 1, "temperature": 17.0, "address": "28FF5E1804150334"},
     {"id": 2, "temperature": 13.875,    "address": "28330A9497040373"}]
"""
def read_ds_sensor():
    # Read & convert temperature
    ds_sensor.convert_temp()
    # Wait: min. 750 ms
    sleep_ms(750)
    # Loop over the devices to get the temperature
    result = []
    id = 0
    for device in devices:
        device_address = bytes(device).hex().upper()
        # print(f'Sensor: {device_address}')
        temperature = ds_sensor.read_temp(device)
        # print(f'Temperatur: {temperature}°C')
        # Add the device address and temperature to the json array
        device = {}
        id = id + 1
        device["id"] = id
        device["address"] = device_address
        device["temperature"] = temperature
        result.append(device);
    # Return the json array with the data
    return result

"""
Handle the request containing the command as JSON object.

The DS18B20 sensor data is read if the command is {"request":1}.
The response JSON object is updated.
```

```
:param string data
    JSON object with key:value pair containing the command

:return JSON object response
"""
def handle_request(data):
    # Get the JSON key request {"request":0 or 1}
    request = data["request"]
    if request == 1:
        # Response is OK
        response[config.KEY_STATE] = config.STATE_OK
        response[config.KEY_MESSAGE] = read_ds_sensor()
    else:
        response[config.KEY_STATE] = config.STATE_ERR
        response[config.KEY_MESSAGE] = ""
    return response

# Info
print(f'{VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()

# Scan for One-Wire devices
# Get list of ROM addresses for all of the attached slaves. Each ROM address is an
8-byte long bytearray.
devices = ds_sensor.scan()
print(f'One-Wire Devices found: {len(devices)}')
for device in devices:
    print(f'Device: {bytes(device).hex().upper()}')
print(f'-----')

# Main
while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Create the HTTP response JSON object
        response = {}

        # Parse the post data. In case of error, the status is 0.
        data, status = network.parse_post_request(request)

        # Assign the postdata to the response KEY_TITLE: {'request': 1} or 0
        response[config.KEY_TITLE] = str(data)

        # If status is 1, then the post response is properly parsed, lets get the
sensor data.
        if status == 1:
            response = handle_request(data)
            # HTTP Response={"status": "OK", "title": {"request": 1}, "message":
{"28FF5E1804150334": 19.75, "28330A9497040373": 15.5625}}
        else:
            # Error with unknown command
            response[config.KEY_STATE] = config.STATE_ERR
            response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

        # Send response to the client and close the connection
        network.send_response(cl, response, True)

    except OSError as e:
        network.ledstatus.off()
        cl.close()
        print('[ERROR] Network Connection closed')
```

# Node-RED Client

This solution uses Node-RED as client.
Node-RED is running on the Domoticz Test System.

## Block Diagram




## Node-RED Flow

The flow starts with the Inject node “Set & Trigger HTTP POST Request”, to the msg.payload = {“request”:1} and the msg.url = “picow-ip”.

The node injects after 0.1 second, the HTTP Request node “Send HTTP POST Request” to submit the HTTP POST request to the Pico W Web Server.

The HTTP response JSON object, key status is checked by the Switch node “Check HTTP Response” on OK or ERROR.

If the msg.payload.status is OK, then the Function node “Handle HTTP POST Response” lists the DS18B20 sensor data derived from the JSON array in the key msg.payload.message.

```
// Get the device data from the key message of the payload
data = msg.payload.message;
// node.warn(data);

// Loop over the data
data.forEach(getDevices);

function getDevices(item, index) {
    // node.warn(JSON.stringify(item), index)
    // get the device data
    let device = JSON.parse(JSON.stringify(item));
    node.warn("id=" + device.id + ",address=" + device.address + ",temperature=" +
device.temperature);
}
```

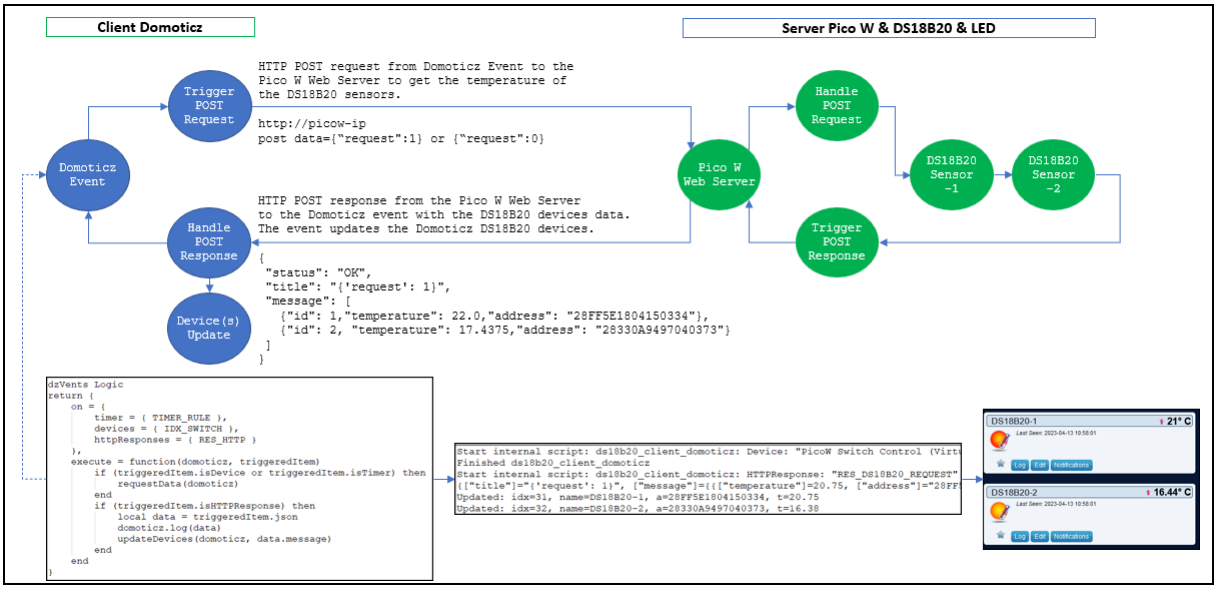
Node-RED Flow Source

```
[{"id":"5e7fdee23c9d653f","type":"tab","label":"DMPP","disabled":false,"info":"domo
ticz-micropython-
projects","env":[]},{"id":"23a82a4e9c805de6","type":"inject","z":"5e7fdee23c9d653f"
,"name":"Set & Trigger HTTP POST
Request","props":[{"p":"payload"},{"p":"url","v":"http://picow-
ip","vt":"str"}],"repeat":"","crontab":"","once":true,"onceDelay":0.1,"topic":"","p
ayload":"{\"request\":1}","payloadType":"json","x":190,"y":80,"wires":[["b88c7ce9a1
832e9a"]]},{"id":"b88c7ce9a1832e9a","type":"http
request","z":"5e7fdee23c9d653f","name":"Send HTTP POST
Request","method":"POST","ret":"obj","paytoqs":"ignore","url":"","tls":"","persist"
:false,"proxy":"","insecureHTTPParser":false,"authType":"","senderr":false,"headers
":[],"x":180,"y":140,"wires":[["17d1f4aaf93e79c5"]]},{"id":"7b7b06756ee15219","type
":"debug","z":"5e7fdee23c9d653f","name":"LOG DS18B20
ERROR","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"
payload","targetType":"msg","statusVal":"","statusType":"auto","x":490,"y":240,"wir
es":[]},{"id":"854cb3f9d6670309","type":"comment","z":"5e7fdee23c9d653f","name":"DS
18B20_Client_Node-RED","info":"DS18B20 Get Temperature\n\nSent a HTTP POST request
to the Pico W web server.\nThe post data contains a JSON object:\n{\"request\":1}
or {\"request\":0}\nSet 1 to ask for data.\n\nThe HTTP response is a JSON
object.\nThe key message holds the data from the sensors.\nThe data is a JSON
array.\n{\n    \"status\":\"OK\",\n    \"title\":\"{'request': 1}\",\n
\"message\":\n        [\n
{\"id\":1,\"temperature\":22,\"address\":\"28FF5E1804150334\"},\n
{\"id\":2,\"temperature\":17.4375,\"address\":\"28330A9497040373\"}\n
]\n}\n","x":160,"y":40,"wires":[]},{"id":"ef2c522da453a408","type":"function","z":"
5e7fdee23c9d653f","name":"Handle HTTP POST Response","func":"// Get the device data
from the key message of the payload\ndata = msg.payload.message;\n//
node.warn(data);\n\n// Loop over the data\ndata.forEach(getDevices);\n\nfunction
getDevices(item, index) {\n    // node.warn(JSON.stringify(item), index)\n    //
get the device data\n    let device = JSON.parse(JSON.stringify(item));\n
node.warn(\"id=\" + device.id + \",address=\" + device.address + \",temperature=\"
+
device.temperature);\n}\n","outputs":1,"noerr":0,"initialize":"","finalize":"","lib
s":[],"x":510,"y":180,"wires":[[]]},{"id":"17d1f4aaf93e79c5","type":"switch","z":"5
e7fdee23c9d653f","name":"Check HTTP Response
Status","property":"payload.status","propertyType":"msg","rules":[{"t":"eq","v":"OK
","vt":"str"},{"t":"eq","v":"ERROR","vt":"str"}],"checkall":"true","repair":false,"
outputs":2,"x":210,"y":200,"wires":[["ef2c522da453a408"],["7b7b06756ee15219"]]}]
```

# Domoticz Client

This solution uses Domoticz as client.
An Automation Event dzVents triggers every minute or via a switch, an HTTP POST request to the Pico W Web Server.
The web server reads the temperature of the DS18B20 sensors and sends HTTP response back to the Automation Event. The response is a JSON object which is parsed to update the Domoticz Temperature devices.

## Block Diagram



## Automation Script

```
--[[
File:   ds18b20_client_domoticz.dzvents
Date:   20230413
Author: Robert W.B. Linn

:description
Requests in regular intervals DS18B20 sensor data from a Pico W Web Server.
The HTTP response JSON object is parsed and the Domoticz temperature devices
assigned to the ds18b20 device(s) are updated.
{["message"]={{["temperature"]=19.5, ["address"]="28FF5E1804150334", ["id"]=1},
{["temperature"]=15.4375, ["address"]="28330A9497040373", ["id"]=2}},
["status"]="OK", ["title"]="{'request': 1}"}

:log
2023-04-13 10:54:29.351 VirtualSensors: Light/Switch (PicoW Switch Control)
2023-04-13 10:54:29.345 Status: User: admin initiated a switch command (16/PicoW
Switch Control/On)
2023-04-13 10:54:29.452 Status: dzVents: Info: Handling events for: "PicoW Switch
Control", value: "On"
2023-04-13 10:54:29.452 Status: dzVents: Info: LOG_DS18B20_REQUEST: ------ Start
internal script: ds18b20_client_domoticz: Device: "PicoW Switch Control
(VirtualSensors)", Index: 16
2023-04-13 10:54:29.453 Status: dzVents: Info: LOG_DS18B20_REQUEST: ------ Finished
ds18b20_client_domoticz
2023-04-13 10:54:29.453 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
2023-04-13 10:54:30.428 Status: dzVents: Info: Handling httpResponse-events for:
"RES_DS18B20_REQUEST"
2023-04-13 10:54:30.428 Status: dzVents: Info: LOG_DS18B20_REQUEST: ------ Start
internal script: ds18b20_client_domoticz: HTTPResponse: "RES_DS18B20_REQUEST"
2023-04-13 10:54:30.428 Status: dzVents: Info: LOG_DS18B20_REQUEST:
{["title"]="{'request': 1}", ["message"]={{["temperature"]=20.75,
["address"]="28FF5E1804150334", ["id"]=1}, {["temperature"]=16.375,
["address"]="28330A9497040373", ["id"]=2}}, ["status"]="OK"}
2023-04-13 10:54:30.440 Status: dzVents: Info: LOG_DS18B20_REQUEST: Updated:
idx=31, name=DS18B20-1, a=28FF5E1804150334, t=20.75
2023-04-13 10:54:30.441 Status: dzVents: Info: LOG_DS18B20_REQUEST: Updated:
idx=32, name=DS18B20-2, a=28330A9497040373, t=16.38
2023-04-13 10:54:30.441 Status: dzVents: Info: LOG_DS18B20_REQUEST: ------ Finished
ds18b20_client_domoticz
2023-04-13 10:54:30.441 Status: EventSystem: Script event triggered:
/home/pi/domoticz/dzVents/runtime/dzVents.lua
]]--

-- Domoticz IDX of the switch triggering the request
local IDX_SWITCH  = 16

local URL_SERVER    = 'http://picow-ip'
local PROJECT       = 'DS18B20_REQUEST'
local RES_HTTP      = 'RES_' .. PROJECT
local LOG_MARKER    = 'LOG_' .. PROJECT

-- Define map table between ds18b20 device address (with prefix X because keys must
start with a character) and the domoticz idx
local devices = {
    X28FF5E1804150334 = 31, -- DS18B20-1
    X28330A9497040373 = 32  -- DS18B20-2
}

-- Send HTTP POST request to get the sensor data from the Pico W Web Server
local function requestData(domoticz)
    local postdata = {}
    postdata["request"] = 1
    domoticz.openURL({
       url = URL_SERVER, method = 'POST',
```

```
       postData = postdata, callback = RES_HTTP,
       })
end

-- Update the devices by looping over the Lua array data and the Lua device table
containing address & temperature.
-- The data is taken from the key message from the HTTP JSON response.
-- {{["address"]="28FF5E1804150334", ["id"]=1, ["temperature"]=19.25},
{["address"]="28330A9497040373", ["id"]=2, ["temperature"]=15.5}}
local function updateDevices(domoticz, data)
    for i, device in ipairs(data) do
        -- Select the device and update
        local deviceaddress = 'X' .. device.address
        local deviceidx = devices[deviceaddress]
        domoticz.devices(deviceidx).updateTemperature(device.temperature)
        domoticz.log(string.format('Updated: idx=%s, name=%s, a=%s, t=%0.2f',
domoticz.devices(deviceidx).idx, domoticz.devices(deviceidx).name, device.address,
device.temperature))
    end
end

--Timer Rule (for tests use every minute)
local TIMER_RULE = 'every minute'

return {
 on = {
     timer = { TIMER_RULE },
     devices = { IDX_SWITCH },
     httpResponses = { RES_HTTP }
    },
    logging = {
        level = domoticz.LOG_INFO,
        marker = LOG_MARKER,
    },
 execute = function(domoticz, triggeredItem)
     -- Trigger the HTTP POST request to the Pico W Web Server
        if (triggeredItem.isDevice or triggeredItem.isTimer) then
requestData(domoticz) end

        -- Handle HTTP response: OK is item statusCode 200 and item.ok true
        -- Else error like statusCode 7, item.ok false
        if (triggeredItem.isHTTPResponse) then
            -- domoticz.log(string.format("%d %s", item.statusCode, item.ok))
            if (triggeredItem.statusCode == 200) then
                if (triggeredItem.isJSON) then
                    local data = triggeredItem.json
                    domoticz.log(data)
                    updateDevices(domoticz, data.message)
                end
            else
                -- Error like 7 false; ERROR 7:Couldn't connect to server
                domoticz.log(string.format("ERROR %d:%s", item.statusCode,
item.statusText))
            end
        end
 end
}
```

# Stepper Motor Selector Switch Angle Move

## Description

This project moves a stepper motor by step via a Domoticz Selector Switch device.

**Ideas for Use**

- [TODO]

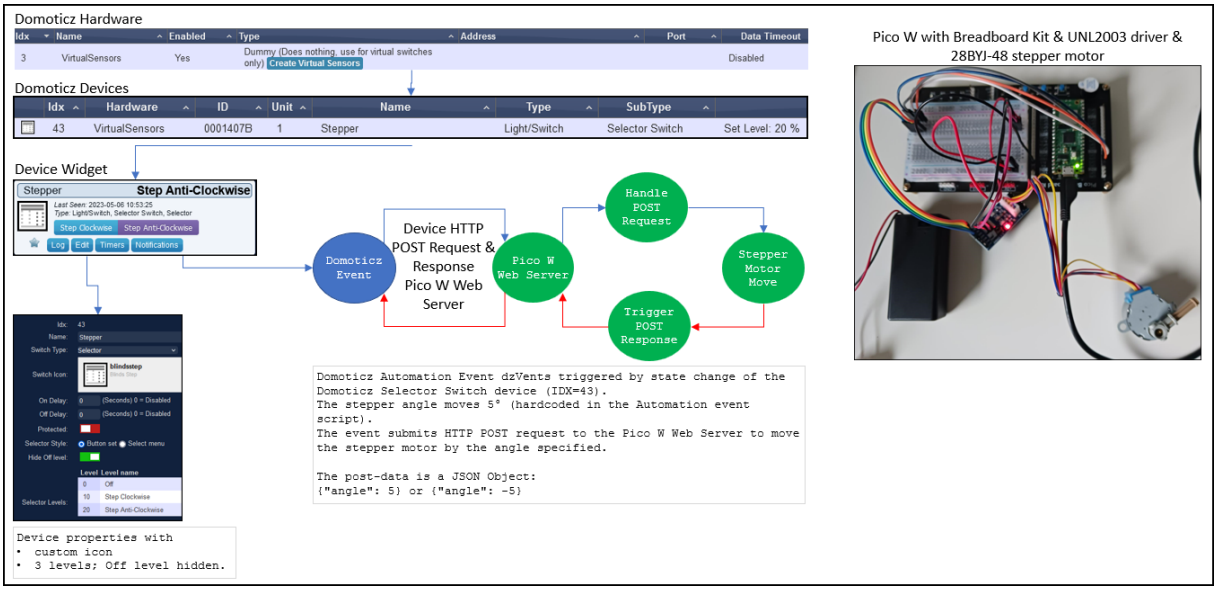
## Solution

The Pico W has a 28BYJ-48 Stepper Motor with ULN2003 motor driver connected and runs as a web server.
The web server waits for a HTTP POST requests from a client connection. The POST data contains the command for moving the stepper motor.
The commands supported are

- “angle” - move the stepper motor by a given angle clockwise or anti-clockwise,
- “steps” - move the stepper motor by number of steps clockwise or anti-clockwise,
- “rotate” - rotate the stepper motor by number of rotations clockwise or ant1-clockwise,

The block diagram shows the solution to move the stepper motor by an angle 5° clockwise or anti-clockwise by a Domoticz Selector Switch.

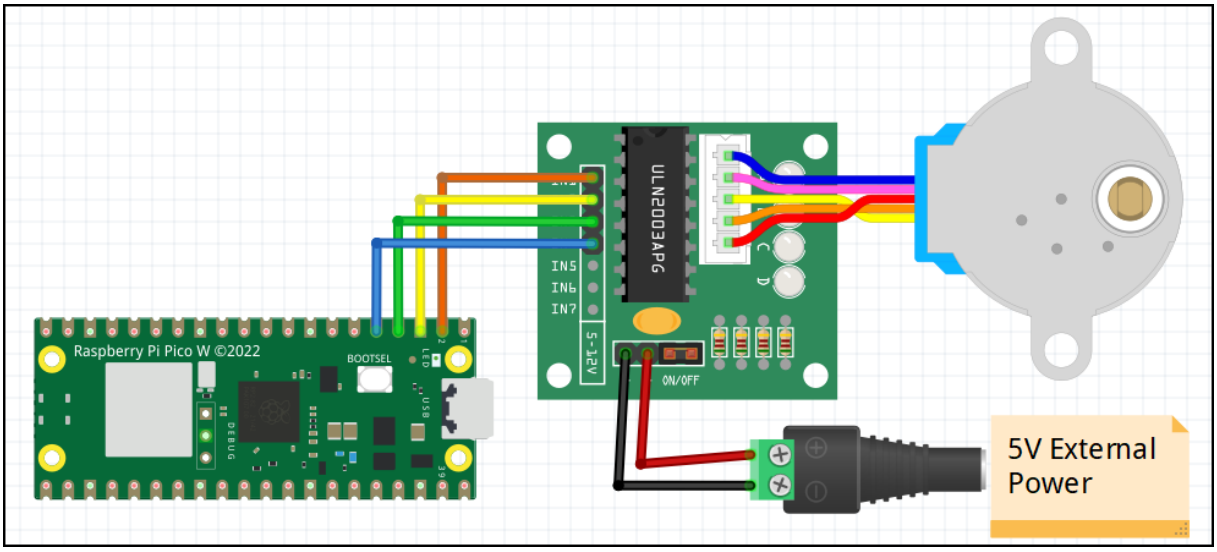
## Block Diagram



# Wiring

| Stepper Motor Driver ULN2003 | Pico W |
|---|---|
| IN1 | GP2 (Pin #4) |
| IN2 | GP3 (Pin #5) |
| IN3 | GP4 (Pin #6) |
| IN4 | GP5 (Pin #7) |
| VCC = External 5V | |
| GND = External 5V | |

# Circuit Diagram



# Domoticz Setup

## Devices

From the hardware “Dummy”, create a Selector Switch device:
Name: Stepper, Sensor Type: Selector Switch

After creating the device, the Domoticz devices list shows the entry:

```
IDX=43, Hardware=VirtualSensors, ID=0001407B, Unit=1, Name=Stepper,
Type=Light/Switch, SubType=Selector Switch, Data=0ff
```

## Automation Script

The automation script is triggered by the state change of the selector switch.

```
--[[
File:   steppermotor.dzvents
Date:   20230506
Author: Robert W.B. Linn

:description
Turn the stepper motor stepwise in the direction clockwise or anticlockwise by
using a Domoticz Selector Switch. The selector switch has 3 levels 0, 10, 20.
The level 0, Off, is not used. Level 10 is Step Clockwise, and level 20 is Step
Anti-Clockwise. This means the selector switch shows two buttons Step Clockwise and
Step Anti-Clockwise. Each step moves the stepper motor in the direction by an angle
of 5°. The angle is hardcoded but could be set flexible via a User Variable.
]]--

-- Domoticz device Selector Switch
local IDX_STEPPER_SELECTOR = 43
-- Pico W web server IP address
local HTTP_URL       = 'http://picow-ip'
-- Callback and logging
local PROJECT       = 'STEPPER'
local HTTP_RES      = 'RES_' .. PROJECT
local LOG_MARKER    = 'LOG_' .. PROJECT
-- Stepper
local ANGLE_MOVE = 5

-- Post Data to the Pico W Webserver.
-- The data is a JSON object with the servo angle to move: {"angle":NNN}
local function HTTPPost(domoticz, angle)
    local data = {}
    data['angle'] = angle
    domoticz.log(data)
    domoticz.openURL({
        url = HTTP_URL, method = 'POST',
        headers = { ['content-type'] = 'application/json' },
        postData = data, callback = HTTP_RES,
       })
end

return {
 on = { devices = { IDX_STEPPER_SELECTOR }, httpResponses = { HTTP_RES } },
 logging = { level = domoticz.LOG_INFO, marker = LOG_MARKER },
 execute = function(domoticz, item)
  if (item.isDevice) then
   if item.idx == IDX_STEPPER_SELECTOR then
      domoticz.log(string.format("device=%s, state=%s, level=%d",
       item.name, item.state, item.level))
       local angle
       if item.level == 10 then angle = ANGLE_MOVE end
```

```
      if item.level == 20 then angle = ANGLE_MOVE * -1 end
      HTTPPost(domoticz, angle)
    end
   end
   if (item.isHTTPResponse) then
    if (item.isJSON) then
     -- {"status": "OK", "title": "{\"angle\": 5}", "message": "5"}
     local data = item.json
     domoticz.log(string.format("status=%s, title=%s, message=%s",
       data.status, data.title, data.message))
    end
   end
 end
}
```

# Web Server

## Libraries

The MicroPython script uses the MicroPython library *stepper* stored in Pico W folder lib.

**Credits**
The stepper library is based on the ULN2003 library (c) IDWizard 2017, # MIT License.
Thanks for developing & sharing.

```
"""
File: stepper.py
Date: 20230429
Author: https://github.com/IDWizard/uln2003 (c) IDWizard 2017, # MIT License.
Thanks for developing & sharing.
Enhanced: Robert W.B. Linn

Library for the stepper motor 8BYJ-48 5V DC with ULN2003 motor driver.
Tested on a Raspberry Pi Pico W.

:usage
stepper = Stepper('HALF_STEP',2 , 3, 4, 5, delay=1)

stepper.step(60, 1)
time.sleep(1)
stepper.step(-60)
time.sleep(1)

stepper.angle(90, 1)
time.sleep(1)
stepper.angle(-90)
time.sleep(1)

stepper.rotate(1)
time.sleep(1)
stepper.rotate(-1)
time.sleep(1)
"""

# Imports
from machine import Pin
import time

#
class Stepper:
    # Reference: http://www.jangeox.be/2013/10/stepper-motor-28byj-48_25.html
    FULL_ROTATION = int(4075.7728395061727 / 8)

    # Modes
    HALF_STEP_MODE = 'HALF_STEP'
    FULL_STEP_MODE = 'FULL_STEP'

    # Mode bit sequences for the 4 motor pins
    HALF_STEP = [
        [0, 0, 0, 1],
        [0, 0, 1, 1],
        [0, 0, 1, 0],
        [0, 1, 1, 0],
        [0, 1, 0, 0],
        [1, 1, 0, 0],
        [1, 0, 0, 0],
        [1, 0, 0, 1],
    ]

    FULL_STEP = [
```

```
        [1, 0, 1, 0],
        [0, 1, 1, 0],
        [0, 1, 0, 1],
        [1, 0, 0, 1]
    ]

    def __init__(self, mode=HALF_STEP_MODE, IN1=2, IN2=3, IN3=4, IN4=5, delay=1):
        """Init the class with default Pico pins GP2 - GP5.
        """
        if mode == self.FULL_STEP_MODE:
         self.mode = self.FULL_STEP
        else:
         self.mode = self.HALF_STEP

        self.pin1 = Pin(IN1, Pin.OUT)
        self.pin2 = Pin(IN2, Pin.OUT)
        self.pin3 = Pin(IN3, Pin.OUT)
        self.pin4 = Pin(IN4, Pin.OUT)
        # Recommend 10+ for FULL_STEP, 1 is OK for HALF_STEP
        self.delay = delay

        # Initialize all to 0
        self.reset()

    def step(self, count, direction=1):
        """Rotate count steps. direction = -1 means anti-clockwise"""
        if count < 0:
            direction = -1
            count = abs(count)
        for x in range(count):
            for bit in self.mode[::direction]:
                self.pin1(bit[0])
                self.pin2(bit[1])
                self.pin3(bit[2])
                self.pin4(bit[3])
                time.sleep_ms(self.delay)
        self.reset()

    def angle(self, r, direction=1):
        """Move the stepper by angle."""
        if r < 0:
            direction = -1
            r = abs(r)
     self.step(int(self.FULL_ROTATION * r / 360), direction)

    def rotate(self, count, direction=1):
        """Rotate the stepper by 360°."""
        if count < 0:
            direction = -1
            count = abs(count)
        for n in range(count):
            self.angle(360, direction)

    def reset(self):
        # Reset to 0, no holding, these are geared, you can't move them
        self.pin1(0)
        self.pin2(0)
        self.pin3(0)
        self.pin4(0)
```

## MicroPython Script

```
"""
File: steppermotor.py
Date: 20230506
Author: Robert W.B. Linn

:description
PicoW RESTful webserver to move a stepper motor (28BYJ-48 Stepper Motor with
ULN2003 motor driver).
The incoming data is from a HTTP POST request with JSON object (key:value pair)
containing the command to control the stepper motor:
angle: NNN
steps: NNN
rotate: NNN

:examples
Command submitted using curl with HTTP response.

curl -v -H "Content-Type:application/json" -d "{\"angle\":90}" http://picow-ip
{"title": "{'angle': 90}", "message": "90", "status": "OK"}

curl -v -H "Content-Type:application/json" -d "{\"angle\":-90}" http://picow-ip
{"title": "{'angle': -90}", "message": "-90", "status": "OK"}

curl -v -H "Content-Type:application/json" -d "{\"steps\":-200}" http://picow-ip
{"status": "OK", "title": "{'steps': -200}", "message": "-200"}

curl -v -H "Content-Type:application/json" -d "{\"rotate\":2}" http://picow-ip
{"status": "OK", "title": "{'rotate': 2}", "message": "2"}

curl -v -H "Content-Type:application/json" -d "{\"rotate\":-2}" http://picow-ip
{"status": "OK", "title": "{'rotate': -2}", "message": "-2"}

Error example:
curl -v -H "Content-Type:application/json" -d "{\"step\":100}" http://picow-ip
{"status": "ERROR", "title": "{'step': 100}", "message": "Unknown command."}

:log
Stepper Motor v20230505
Network waiting for connection...
Network connected OK
Network IP picow-ip
Network listening on ('0.0.0.0', 80)
Network client connected from client-ip
HTTP Command={'steps': -200}
HTTP Response={"status": "OK", "title": "{'steps': -200}", "message": "-200"}
Network connection closed
Network client connected from client-ip
HTTP Command={'step': 100}
HTTP Response={"status": "ERROR", "title": "{'step': 100}", "message": "Unknown
command."}
Network connection closed
Network client connected from client-ip
HTTP Command={'steps': -200}
HTTP Response={"status": "OK", "title": "{'steps': -200}", "message": "-200"}
Network connection closed
Network client connected from client-ip
HTTP Command={'rotate': 2}
HTTP Response={"status": "OK", "title": "{'rotate': 2}", "message": "2"}
Network connection closed
Network client connected from client-ip
HTTP Command={'rotate': -2}
HTTP Response={"status": "OK", "title": "{'rotate': -2}", "message": "-2"}
Network connection closed

:wiring
Stepper Motor = Pico W
```

```
IN1 = GP2
IN2 = GP3
IN3 = GP4
IN4 = GP5
VCC = External 5V
GND = GND common with external GND
"""

# Libraries
# Steppermotor lib created by the author
from machine import Pin
from time import sleep
import json
# Call server from server.py (must be uploaded to the picow)
from server import Server
# Stepper stored in PicoW folder lib
# Credits: This library is based on: https://github.com/IDWizard/uln2003 (c)
IDWizard 2017, # MIT License.
from stepper import Stepper
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
NAME = 'Stepper Motor'
VERSION = 'v20230506'

# Create a stepper motor object with defaults
stepper = Stepper()

"""
Handle the request to move the stepper.
"""
def handle_request(cmd, status):
    # Assign the command to the response title
    response[config.KEY_TITLE] = str(cmd)
    # Set the response initially to ok
    response[config.KEY_STATE] = config.STATE_OK

    # If the status is 1 (OK) then action
    if status == 1:

        # Get the command by checking the json key
        if cmd.get('angle') != None:
            # Get the angle of the stepper to move
            angle = cmd['angle']
            # Move the stepper by angle
            stepper.angle(angle)
            response[config.KEY_MESSAGE] = str(angle)

        elif cmd.get('steps') != None:
            # Get the number of steps for the stepper to move
            steps = cmd['steps']
            # Move the stepper by steps
            stepper.step(steps)
            response[config.KEY_MESSAGE] = str(steps)

        elif cmd.get('rotate') != None:
            # Get the number of steps for the stepper to move
            ntimes = cmd['rotate']
            # Move the stepper by rotation
            stepper.rotate(ntimes)
            response[config.KEY_MESSAGE] = str(ntimes)

        else:
            response[config.KEY_STATE] = config.STATE_ERR
            response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

    else:
        response[config.KEY_STATE] = config.STATE_ERR
```

```
        response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

    # Return the response which is send to Domoticz
    return response

# Main
print(f'{NAME} {VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()

"""
Main Loop
"""
while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Get the cmd to set the LCD text as JSON object from the POST request
        cmd, status = network.parse_post_request(request)

        # Create the HTTP response JSON object
        response = {}

        # Handle the command to set the servo pos
        # Set the response
        response = handle_request(cmd, status)

        # Send the response to Domoticz and close the connection (wait for new)
        network.send_response(cl, response, True)

    except OSError as e:
        ledstatus.off()
        cl.close()
        print('[ERROR] Network Connection closed')
```

# Enhancement Ideas

**LED Indicators**

If the stepper motor is moving, turn a RED LED on else turn a GREEN LED on indicating stand-by.

# Stepper Motor Blind Simulation

## Description

This project simulates a blind by moving the stepper up and down by a given angle.

## Solution

The solution is based up-on the previous described project Stepper Motor.

Instead using a Domoticz Selector Switch to move the stepper motor by an angle (hardcoded 5°), a Domoticz Blind device, with states Open & Closed is used.

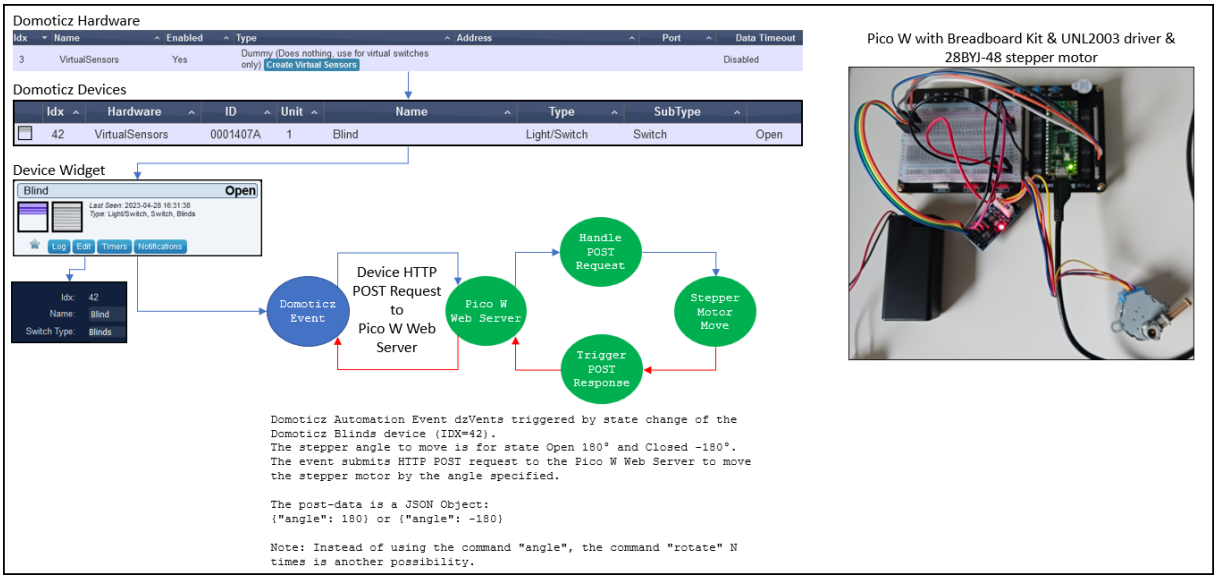
The stepper motor is moved by setting the angle. For the blind simulation:

- Blind Open = Angle 180°
- Blind Closed = Angle -180°

*Notes*
The previous described Domoticz Selector Switch can be used to set the stepper motor starting position (calibrate).

*Alternatives*
Instead changing the angle (command “angle”:NNN) of the stepper motor, the stepper motor could rotate N times (command “rotate”: N).

## Block Diagram



# Domoticz Setup

## Devices

From the hardware “Dummy”, create a Blind device:
Name: Blind, Sensor Type: Switch
After creating the device, the Domoticz devices list shows the entry:

```
IDX=43, Hardware=VirtualSensors, ID=0001407A, Unit=1, Name=Blind,
Type=Light/Switch, SubType=Switch, Data=0ff
```

To set the switch type, select the device widget and change the switch type to blind.
See Block Diagram.

## Automation Script

```
--[[
File:   stepperblind.dzvents
Date:   20230506
Author: Robert W.B. Linn

:description
Simulating a blind state Open or Closed by moving a stepper motor by angle 180
(Open) or -180 (Closed).
]]--

-- Domoticz device from type light/switch, light, blinds
local IDX_BLIND = 42
-- Pico W web server IP address
local HTTP_URL       = 'http://picow-ip'
-- Callback and logging
local PROJECT       = 'STEPPERBLIND'
local HTTP_RES      = 'RES_' .. PROJECT
local LOG_MARKER    = 'LOG_' .. PROJECT
-- Stepper
local ANGLE_OPEN = 180
local ANGLE_CLOSED = ANGLE_OPEN * -1
local STATE_OPEN = 'Open'
local STATE_CLOSED = 'Closed'

-- Post Data to the Pico W Webserver.
-- The data is a JSON object with the servo angle rounded: {"angle":NNN}
local function HTTPPost(domoticz, angle)
    local data = {}
    data['angle'] = angle
    domoticz.log(string.format('device=%s, angle=%d',
     domoticz.devices(IDX_SWITCH).name, angle), domoticz.LOG_INFO)
    domoticz.openURL({
        url = HTTP_URL, method = 'POST',
        headers = { ['content-type'] = 'application/json' },
        postData = data, callback = HTTP_RES,
       })
end

-- Set the blind angle
local function setblind(domoticz, state)
    local angle = ANGLE_OPEN
    if state == STATE_CLOSED then angle = ANGLE_CLOSED end
    HTTPPost(domoticz, angle)
end

return {
 on = { devices = { IDX_BLIND }, httpResponses = { HTTP_RES } },
 data = { stateprev = {initial = STATE_OPEN}, },
```

```
 logging = { level = domoticz.LOG_INFO, marker = LOG_MARKER },
 execute = function(domoticz, item)
  if (item.isDevice) then
   domoticz.log(string.format("device=%s, state=%s, stateprev=%s",
    item.name, item.state, domoticz.data.stateprev))
    -- Check if the state has changed i.e., Open to Closed or Closed to Open
   if item.state ~= domoticz.data.stateprev then
    -- Set the blind state
    setblind(domoticz, item.state)
    -- Capture the new state
    domoticz.data.stateprev = item.state
   end
   end
  if (item.isHTTPResponse) then
   if (item.isJSON) then
    -- {"status": "OK", "title": "{\"angle\": 180}", "message": "180"}
    local data = item.json
    domoticz.log(string.format("status=%s, title=%s, message=%s",
     data.status, data.title, data.message))
   end
  end
 end
}
```

# Web Server

## Libraries

See project Stepper Motor > Web Server. The same solution is used.

## MicroPython Script

See project Stepper Motor > Web Server. The same solution is used.

# Stepper Motor Timer Run Stop

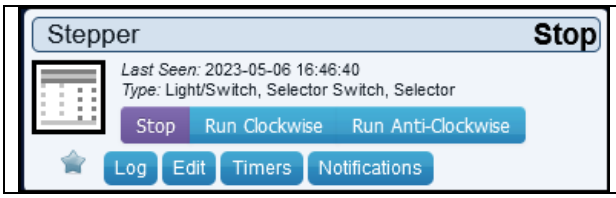
## Description

This project enables to run or stop a stepper motor.

## Solution

The solution is based up-on the previous described project Stepper Motor.

A Domoticz Selector Switch is used to move the stepper motor depending selected level:

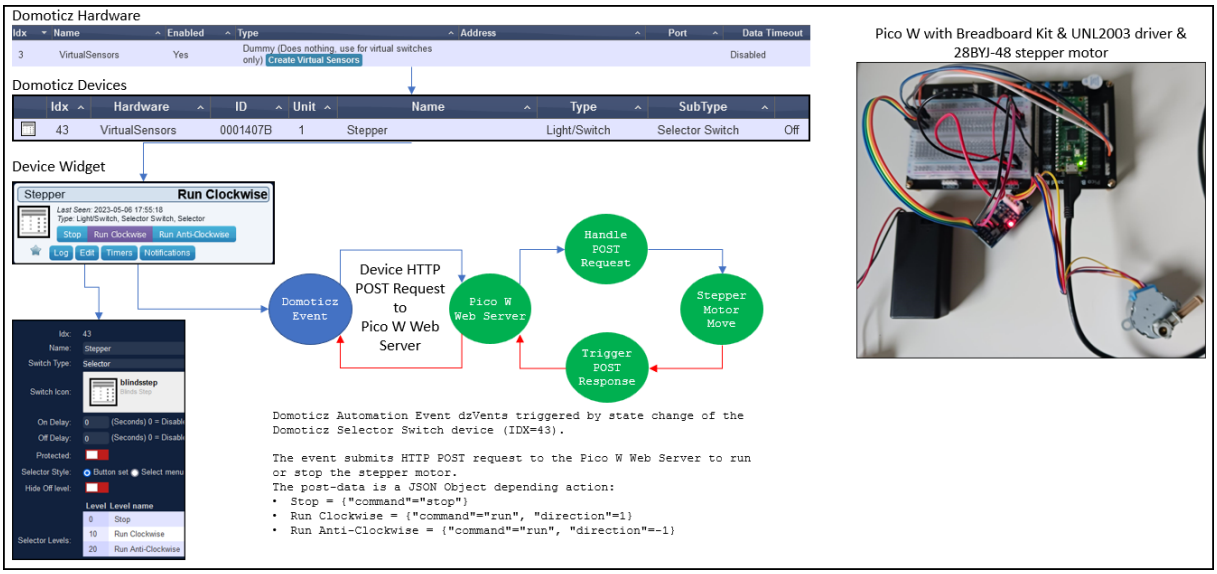



Levels
0 = Stop
10 = Run Clockwise
20 = Run Anti-Clockwise

By clicking on the selected selector switch button, an automation event is triggered. The event creates a JSON object used as post-data for an HTTP POST request to the Pico W, which is running as a web server.
The Pico W web server parses the post-data and executes the command. If the command is run then the stepper motor starts running in the direction given, until a stop command is received.
The stepper motor runs by using a timer with callback function.

*Note*
Prior using the timer class, explored other libraries like asyncio or thread ... but found (for now) too complex to use.
MicroPython's Timer class defines a baseline operation of executing a callback with a given period (or once after some delay),

# Block Diagram







# Domoticz Setup

## Devices

From the hardware “Dummy”, create a Selector Switch device:
Name: Stepper, Sensor Type: Selector Switch
After creating the device, the Domoticz devices list shows the entry:

```
IDX=43, Hardware=VirtualSensors, ID=0001407A, Unit=1, Name=Stepper,
Type=Light/Switch, SubType=Selector Switch, Data=Off
```

## Automation Script

```
--[[
File:   steppermotor_timer.dzvents
Date:   20230506
Author: Robert W.B. Linn

:description
Turn the stepper motor stepwise in the direction clockwise or anticlockwise by
using a Domoticz Selector Switch.
The selector switch has 3 levels 0, 10, 20.
The level 0, Off stops the stepper motor. Level 10 is Run Clockwise, and level 20
is Run Anti-Clockwise.
This means the selector switch shows two buttons Run Clockwise and Run Anti-
Clockwise.
]]--

-- Domoticz device Selector Switch
local IDX_STEPPER_SELECTOR = 43

-- Pico W web server IP address
local HTTP_URL       = 'http://picow-ip'
-- Callback and logging
local PROJECT        = 'STEPPERTIMER'
local HTTP_RES       = 'RES_' .. PROJECT
local LOG_MARKER     = 'LOG_' .. PROJECT

-- Post Data to the Pico W Webserver.
```

```
-- The data is a JSON object with the command run or stop.
local function HTTPPost(domoticz, data)
    domoticz.log(data)
    domoticz.openURL({
        url = HTTP_URL, method = 'POST',
        headers = { ['content-type'] = 'application/json' },
        postData = data, callback = HTTP_RES,
       })
end

return {
 on = { devices = { IDX_STEPPER_SELECTOR }, httpResponses = { HTTP_RES } },
 logging = { level = domoticz.LOG_INFO, marker = LOG_MARKER },
 execute = function(domoticz, item)
 if (item.isDevice) then
  if item.idx == IDX_STEPPER_SELECTOR then
   domoticz.log(string.format("device=%s, state=%s, level=%d",
    item.name, item.state, item.level))
   local data = {}
   if item.level == 0 then
    data['command'] = 'stop'
   end
   if item.level == 10 then
    data['command'] = 'run'
    data['direction'] = 1
   end
   if item.level == 20 then
    data['command'] = 'run'
    data['direction'] = -1
   end
   HTTPPost(domoticz, data)
  end
 end

  if (item.isHTTPResponse) then
   if (item.isJSON) then
    local data = item.json
    domoticz.log(string.format("status=%s, title=%s, message=%s",
     data.status, data.title, data.message))
   end
  end
 end
}
```

# Web Server

## Libraries

See project Stepper Motor > Web Server. The same solution is used.

## MicroPython Script

```
"""
File: steppermotor_timer.py
Date: 20230506
Author: Robert W.B. Linn

:description
PicoW RESTful webserver to move a stepper motor (28BYJ-48 Stepper Motor with
ULN2003 motor driver).
The incoming data is from a HTTP POST request with JSON object (key:value pair)
containing the command:
{"command":"run", "direction":1 or -1}. Clockwise is 1 and anti-clockwise is -1.
{"command":"stop"}
```

```
The motor is running using a timer with periodic=10ms and callback which sets one
step. The stepper motor object uses 1ms for a step.

Additionally, a RED LED indicates if the stepper motor is running

:note
Prior using the timer class, explored other libraries like asyncio or thread ...
but found (for now) too complex to use.
MicroPython's Timer class defines a baseline operation of executing a callback with
a given period (or once after some delay),

:examples
Command submitted using curl with HTTP response.
curl -v -H "Content-Type:application/json" -d "{\"command\":\"run\",
\"direction\":1}" http://picow-ip
{"status": "OK", "title": "{'command': 'run', 'direction': 1}", "message": "run"}

curl -v -H "Content-Type:application/json" -d "{\"command\":\"run\",
\"direction\":-1}" http:// picow-ip {"status": "OK", "title": "{'command': 'run',
'direction': -1}", "message": "run"}

curl -v -H "Content-Type:application/json" -d "{\"command\":\"stop\"}" http://
picow-ip
{"status": "OK", "title": "{'command': 'stop'}", "message": "stop"}

:wiring
Stepper Motor = Pico W
IN1 = GP2
IN2 = GP3
IN3 = GP4
IN4 = GP5
VCC = External 5V
GND = GND common with external GND

LED RED = Pico W
+ = GP21 (Pin #27)
GND = GND
"""

# Libraries
from machine import Pin
from machine import Timer
from time import sleep
import json
# Call server from server.py (must be uploaded to the picow)
from server import Server
# Stepper stored in PicoW folder lib
# Credits: This library is based on: https://github.com/IDWizard/uln2003 (c)
IDWizard 2017, # MIT License.
from stepper import Stepper
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
NAME = 'Stepper Motor Timer'
VERSION = 'v20230506'

# Create the LED
led1 = Pin(21, Pin.OUT)
led1.off()

# Create a stepper motor object with defaults
stepper = Stepper()
sleep(0.1)

# Create the timer object used by the function handle_request
timer_stepper = Timer()
```

```
"""
Run the stepper motor by moving a single step in direction clockwise (cw) or anti-
clockwise (acw).
Function is used by the timer_stepper object as callback.
Argument t is mandatory.
"""
def run_stepper_motor_cw(t):
    stepper.step(1)

def run_stepper_motor_acw(t):
    stepper.step(-1)

"""
Handle the request to move the stepper.
"""
def handle_request(cmd, status):
    # Assign the command to the response title
    response[config.KEY_TITLE] = str(cmd)
    # Set the response initially to ok
    response[config.KEY_STATE] = config.STATE_OK

    # If the status is 1 (OK) then action
    if status == 1:

        if cmd.get('command') != None:
            command = cmd['command']
            # print(f'Stepper Motor Command: {command}')
            if command == 'run':
                # Set the direction clockwise or anti-clockwise
                direction = 1
                if cmd.get('direction') != None:
                    direction = cmd['direction']
                # periodic with 10ms period
                if direction == 1:
                    timer_stepper.init(mode=Timer.PERIODIC, period=10,
callback=run_stepper_motor_cw)
                elif direction == -1:
                    timer_stepper.init(mode=Timer.PERIODIC, period=10,
callback=run_stepper_motor_acw)
                else:
                    print(f'[ERROR] Wrong direction')
                led1.on()
                response[config.KEY_MESSAGE] = 'run'
            elif command == 'stop':
                timer_stepper.deinit()
                led1.off()
                response[config.KEY_MESSAGE] = 'stop'
            else:
                response[config.KEY_STATE] = config.STATE_ERR
                response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

        else:
            response[config.KEY_STATE] = config.STATE_ERR
            response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN
    else:
        response[config.KEY_STATE] = config.STATE_ERR
        response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN

    # Return the response which is send to Domoticz
    return response

# Main
print(f'{NAME} {VERSION}')

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD, DEBUG=True)

# Connect to the network and get the server object
server = network.connect()
```

```
"""
Main Loop
"""
while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Get the cmd
        cmd, status = network.parse_post_request(request)

        # Create the HTTP response JSON object
        response = {}

        # Handle the command to set the servo pos
        # Set the response
        response = handle_request(cmd, status)

        # Send the response to Domoticz and close the connection (wait for new)
        network.send_response(cl, response, True)

    except OSError as e:
        ledstatus.off()
        cl.close()
        print('[ERROR] Network Connection closed')
```

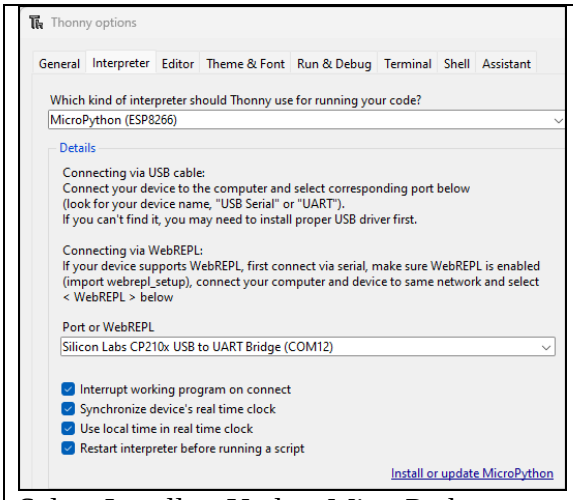
# ESP8266 Projects

## Introduction

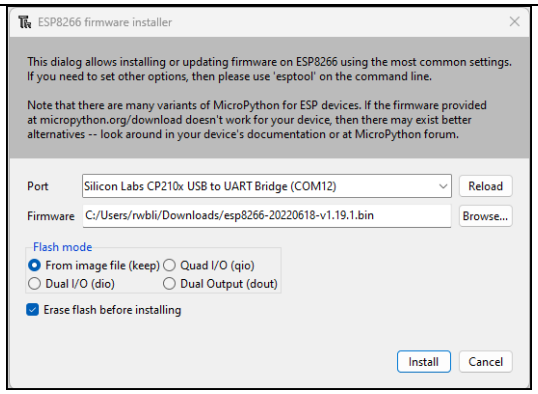
This chapter explores how to use the ESP8266 microcontroller with MicroPython.
As mentioned in the Introduction, the core of the projects uses the Raspberry Pi Pico W microcontroller, but out of curiosity made few simple ESP8266 projects.

## Setup

The hardware is an ESP8266 NodeMCU (Model ESP8266MOD, Vendor AI-THINKER).
The NodeMCU is connected to USB COM12 of the development device.
Thonny is used to flash and program the NodeMCU.
The firmware is download from here.
To flash, open Thonny, go to bottom right, select MicroPython (ESP8266) COM12.
Then select menu Tools > Options > Interpreter




Select Install or Update MicroPython.
See picture right.




Press Install.
*Notes*
During flash the onboard blue LED is blinking.
If the option “Erase flash before installing” is disabled, the MicroPython files are not deleted.
After flash the Thonny log shows (REPL):
MicroPython v1.19.1 on 2022-06-18; ESP module with ESP8266

# LED Blink

## Description

This project let an LED blink every 2 seconds.

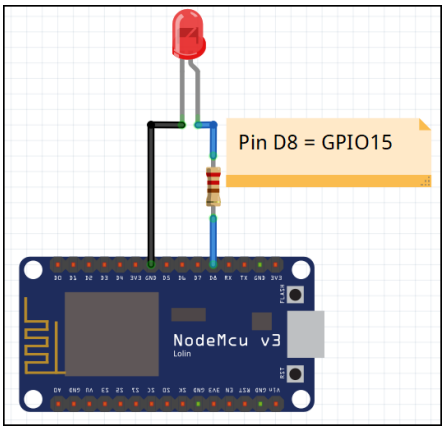
## Solution

The ESP8266, NodeMCU, has an LED connected.

## Wiring

| **LED (red)** | **NodeMCU** |
|---|---|
| + (Anode) with resistor 220k | GPIO15 (Pin #D8) |
| GND (Cathode) | GND |

## Circuit Diagram




## MicroPython Script

```
"""
File: esp8266-ledblink.py
Date: 20230413
Author: Robert W.B. Linn

:description
Let LED connected to an ESP8266 pin #D8 (GPIO15) blink every 2 seconds.

:log
esp8266-ledblink v20230413
LED state On
LED state Off
LED state On
"""

# Imports
from machine import Pin
from time import sleep

VERSION = 'esp8266-ledblink v20230413'

# Define the LED pin D8=GPIO15
```

```
PIN_LED_D8 = 15
# Create the LED object and set state off
led = Pin(PIN_LED_D8, Pin.OUT)
led.value(0)

# Set blink delay
DELAY = 2# seconds

def IIF(state):
    """Convert the state 1 | 0 to On | Off string.
    """
    if state == 1:
        return 'On'
    else:
        return 'Off'

print(VERSION)
# Loop forever
while True:
    led.value(not led.value())
    state = led.value()

    # Print the led state On or Off
    print('LED state', IIF(led.value()))

    # This is not working on the ESP8266
    # print(f'LED state {led.value()}')

    # Wait few seconds
    sleep(DELAY)
```

# LED Remote Control

## Description

This project sets the state of an LED (connected to the NodeMCU) On/Off via a Domoticz Switch submitting an HTTP GET request.

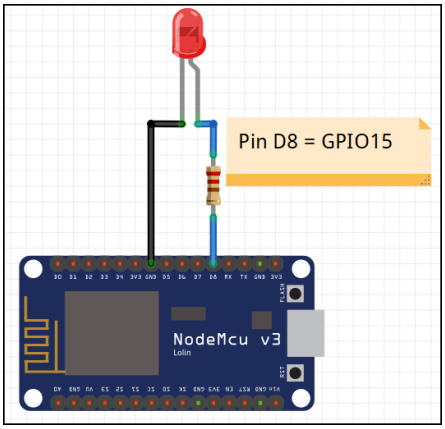
## Solution

The NodeMCU runs as a web server and listens to HTTP GET requests to turn the LED On or Off.
This is the same project as described for the Pico W project Domoticz Switch Device Action.

## Wiring

| **LED (red)** | **NodeMCU** |
|---|---|
| + (Anode) with resistor 220k | GPIO15 (Pin #D8) |
| GND (Cathode) | GND |

## Circuit Diagram



# Web Server

## Libraries

The MicroPython script uses the MicroPython library “espserver.py” developed by the author. This library is based upon the Pico W Web Server solution.

```
"""
File:espserver.py
Date:20230423
Author: Robert W.B. Linn

:description
Class to manage the ESP8266 RESTful webserver.
Commands set via HTTP GET or POST requests with HTTP response JSON object.
"""

# Libraries
import network
import urequests
import socket
import time
from machine import Pin
import json

"""
Class Server
"""
class Server:
    # Constants
    NAME = 'ESPServer'
    VERSION = 'v20230414'

    CRLF = chr(13) + chr(10)
    SPACE = chr(32)

    # Domoticz
    # HTTP response JSON keys
    KEY_STATE= 'status'
    KEY_TITLE= 'title'
    KEY_MESSAGE = 'message'

    # Messages used for HTTP response
    STATE_OK    = 'OK'
    STATE_ERR   = 'ERROR'
    MESSAGE_EMPTY  = ''
    MESSAGE_UNKNOWN = 'Unknown'
    MESSAGE_CMD_UNKNOWN    = 'Unknown command.'
    MESSAGE_ON   = 'On'
    MESSAGE_OFF   = 'Off'

    def __init__(self, wifi_ssid, wifi_password, STATUS_PIN=16, DEBUG=True):
        """
        Init the network with defaults.

        :param string wifi_ssid
            SSID of the network to connect

        :param string wifi_password
            Passord of the network to connect

        :param string | int STATUS_PIN
            Pin number of the LED indicating network status connected
            For an NodeMCU this is GPIO16 (pin #D0)

        :param bool DEBUG
```

```
            Flag to set the log for debugging purposes
        """
        self.debug = DEBUG
        self.wifi_ssid = wifi_ssid
        self.wifi_password = wifi_password

        # Create the onboard LED object to indicate controller is up and network
connected
        self.ledstatus = Pin(STATUS_PIN, Pin.OUT)
        self.ledstatus.off()

    def log(self, msg):
        """
        Log to the console if debug flag is true.

        :param string msg
            Message to print
        """
        if self.debug:
            print(msg)

    def connect(self):
        """
        Connect to the network using the class SSID and password.

        :param None

        :return object server
            Server object.

        :example
            # Create network object
            network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
            # Connect to the network and get the server object
            server = network.connect()
        """
        try:
            wlan = network.WLAN(network.STA_IF)
            if wlan.isconnected():
                wlan.active(False)
            wlan.active(True)
            wlan.connect(self.wifi_ssid, self.wifi_password)
            # Network connection
            max_wait = 10
            self.log('Network waiting for connection...')
            while wlan.isconnected() == False:
                if wlan.status() < 0 or wlan.status() >= 3:
                    break
                max_wait -= 1
                pass
                max_wait -= 1

            if wlan.isconnected() == False:
                self.ledstatus.off()
                raise RuntimeError('[ERROR] Network connection failed!')
            else:
                self.ledstatus.on()
                self.log('Network connected OK')
                status = wlan.ifconfig()
                self.log('Network IP ' + status[0] )

            # Network Get address
            addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]

            # Network Create the server socket
            server = socket.socket()

            # Option to reuse addr to avoid error EADDRINUSE
            server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
            # Bind the address befor starting to listen for icoming client
connections
            server.bind(addr)
            server.listen(1)
            self.log('Network listening on ' + str(addr))
            # self.log(server)
            return server
        except OSError as e:
            self.ledstatus.off()
            cl.close()
            raise RuntimeError('[ERROR] Network connection closed')

    def parse_get_request(self, request):
        """
        Parse the command from the HTTP GET Request.
        The first line of the request contains the command.
        The first line is split and the 2nd item holds the command + data.
        Example first line with the command:
        GET /led1/on HTTP/1.1
        The command is /led1/on.

        :param string request
            HTTP GET request

        :return string command
            Command, i.e. /led1/on

        :return int status
            0 = Error, 1 = OK

        :example
            # Parse the get data. In case of error, the status is 0.
            cmd, status = network.parse_get_request(request)
        """
        status = 0
        cmd = self.MESSAGE_CMD_UNKNOWN

        # Split the decoded request string into a list
        data = str(request.decode()).split(self.CRLF)

        # Check if there is data to get the first item
        if (len(data) > 0):
            # print(data[0])
            # Split the first item which is the command string into a list with 3
items
            cmds = data[0].split(self.SPACE)
            # Check length and get the 2nd item, i.e. /led1/on
            if len(cmds) == 3:
                cmd = cmds[1]
                status = 1
            else:
                print('[ERROR] HTTP GET number of command items invalid. Expect 3,
got ' + len(cmds))
        else:
            print('[ERROR] HTTP GET request not valid.')
        self.log('HTTP Command=' + cmd)

        # Return the command, i.e. /led1/on etc.
        return cmd, status

    def parse_post_request(self, request):
        """
        Parse the command from the HTTP POST Request.
        The last line of the HTTP request contains the command + data.
        The HTTP request is decoded and split as a string list.
        The last line is a JSON object with key:value pair(s).

        :param string request
```

```
            HTTP request

        :return string command
            Command as JSON key:value pair(s), i.e. {"led":1}

        :return int status
            0 = Error, 1 = OK

        :example
            # Parse the post data. In case of error, the status is 0.
            data, status = network.parse_get_request(request)
        """
        status = 0
        cmd = self.MESSAGE_CMD_UNKNOWN

        # Split the decoded request string into a list
        data = str(request.decode()).split(self.CRLF)

        # Check if there is data to get the last item
        # At least 8 items
        if (len(data) > 7):
            # JSON parse the last list item holding the command as JSON string
            # Convert the string to a JSON object
            try:
                cmd = json.loads(data[len(data) - 1])
                status = 1
            except ValueError:
                # In case the JSON data can not be parsed
                cmd = data[len(data) - 1]
                print('[ERROR] HTTP POST request not valid (ValueError).')
        else:
            print('[ERROR] HTTP POST request not valid (Not enought items, must be
8 or more).')
        self.log('HTTP Command=' + cmd)

        # Return the command as JSON object, i.e. HTTP Command: {'state': 'on'}
        return cmd, status

    def get_client_connection(self, server):
        """
        Get the client connection.

        :param object server
            Server object which is listening

        :return object cl

        :return string data
            The requested data format depends on the request

        :example
            cl, request = network.get_client_connection(server)
        """
        # Get client connection
        cl, addr = server.accept()
        self.log('Network client connected from ' + addr[0])

        # Get the request data used to extract the command
        request = cl.recv(1024)
        # Return cl and the request data
        return cl, request

    def send_response(self, cl, response, close):
        """
        Send the response to the client, i.e. Domoticz, curl etc. as JSON object.

        :param object cl

        :param JSON response
```

```
        :param bool close
        """
        self.log('HTTP Response=' + json.dumps(response))

        # Important to have a blank line prior JSON response string
        # Note the use of json.dumps for the response
        cl.send('HTTP/1.1 200 OK'+self.CRLF+'content-type:
application/json'+self.CRLF+self.CRLF+json.dumps(response))

        # If flag close is set, ensure to close the connection
        if close == True:
            cl.close()
            self.log('Network connection closed')

    def send_get_request(self, url):
        """
        Network submit http get request to the domoticz server.

        :param string url
            URL of the HTTP request

        :return int status
            0 = Error, 1 = OK

        :return string content
            HTTP response content sent by Domoticz engine

        :example
            Update the Domoticz temp+hum device with IDX 15
            http://domoticz-
ip:port/json.htm?type=command&param=udevice&idx=15&nvalue=0&svalue=16;55;1
        """
        status = 0
        content = ''
        self.log('Send GET request url=',url)
        try:
            # URL encode space
            url = url.replace(' ', '%20')
            r = urequests.get(url)
            j = json.loads(r.content)
            content = j
            self.log('Send GET request status=' + j['status'])
            r.close()
            status = 1
        except OSError as e:
            # print('[ERROR] Sending data', e)
            raise Exception('[ERROR] Sending data ' + e)
        except ValueError as e:
            # print('[ERROR]', e, r.content.decode()')
            raise Exception('[ERROR]', e, r.content.decode())
        return status, content

    def send_post_request(self, url, postdata):
        """
        Network submit http post request to the domoticz server.

        :param string url
            URL of the HTTP request

        :param string postdata
            postdata as JSON object

        :return int status
            0 = Error, 1 = OK

        :example
            Trigger the Domoticz custom event named DHT22 with data JSON object
```

```
            http://domoticz-
ip:port/json.htm?type=command&param=customevent&event=DHT22&data={"h": 58, "t": 16,
"s": 0}
        """
        status = 0
        self.log('Send POST request url='+url+',postdata='+postdata)
        try:
            r = urequests.post(url, data=json.dumps(postdata))
            j = json.loads(r.content)
            self.log('Send POST request status='+j['status'])
            r.close()
            status = 1
        except OSError as e:
            print('[ERROR] Sending data', e)
            # raise Exception('Network Connection failed.')
        return status
```

## MicroPython Script

```
"""
File: esp8266-ledremotecontrol.py
Date: 20230413
Author: Robert W.B. Linn

ESP8266 RESTful webserver listening to control an LED via Domoticz Switch.
Commands are set via HTTP GET request with HTTP response JSON object.

:commands
LED ON:
HTTP Request: http://esp8266-ip/led1/on
HTTP response: {"status": "OK", "title": "/led1/on", "message": "On"}

LED OFF:
HTTP Request: http://esp8266-ip/led1/off
HTTP response: {"status": "OK", "title": "/led1/off", "message": "Off"}

LED STATE:
HTTP Request: http://esp8266-ip/led1/state
HTTP response: {"status": "OK", "title": "/led1/state", "message": "On"}

In case of an error:
HTTP response: {"status": "ERROR", "title": "/led1/x", "message": "Unknown
command."}

Example using curl to turn LED1 on:
curl -v http://esp8266-ip/led1/on

:log
esp8266-ledremotecontrol v20230414
#7 ets_task(4020f560, 28, 3fff9430, 10)
Network waiting for connection...
Network connected OK
Network IP esp8266-ip
Network listening on ('0.0.0.0', 80)
Network client connected from client-ip
HTTP Command=/led1/on
HTTP Response={"title": "/led1/on", "message": "On", "status": "OK"}
Network connection closed
Network client connected from client-ip
HTTP Command=/led1/off
HTTP Response={"title": "/led1/off", "message": "Off", "status": "OK"}
Network connection closed
"""

# Libraries
import network
import socket
import time
from machine import Pin
import json
# Import server class from server.py
from espserver import Server
# Configuration read from config.py (must be uploaded to the picow prior testing)
import config

# Constants
NAME = 'esp8266-ledremotecontrol'
VERSION = 'v20230413'

# URL params to switch LED1 on or off or request state
# http://pico-ip/command
CMD_LED_ON  = '/led1/on'
CMD_LED_OFF  = '/led1/off'
CMD_LED_STATE= '/led1/state'
```

```
# Create the LED object using config.py settings
# Define the LED pin D8=GPIO15
PIN_LED_D8 = 15
# Create the LED object and set state off
led = Pin(PIN_LED_D8, Pin.OUT)
led.value(0)

"""
Handle the request containing the command.
The LED is turned on/off or the state is requested.
The response JSON object is updated.

:param string cmd
    Command to set the LED1 state on/off or get the state.

:return JSON object response
"""
def handle_request(cmd):
    # Turn the LED on
    if cmd == CMD_LED_ON:
        led.value(1)
        response[config.KEY_MESSAGE] = config.MESSAGE_ON
        response[config.KEY_STATE] = config.STATE_OK
    # Turn the LED off
    elif cmd == CMD_LED_OFF:
        led.value(0)
        response[config.KEY_MESSAGE] = config.MESSAGE_OFF
        response[config.KEY_STATE] = config.STATE_OK
    # Get the LED state
    elif cmd == CMD_LED_STATE:
        if led.value() == 1:
            response[config.KEY_MESSAGE] = config.MESSAGE_ON
        else:
            response[config.KEY_MESSAGE] = config.MESSAGE_OFF
        response[config.KEY_STATE] = config.STATE_OK
    else:
        response[config.KEY_STATE] = config.STATE_ERR
        response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN
    return response

# Main
print(NAME, VERSION)

# Create network object
network = Server(config.WIFI_SSID, config.WIFI_PASSWORD)
# Connect to the network and get the server object
server = network.connect()

while True:
    try:
        # Get client connection and the request data
        cl, request = network.get_client_connection(server)

        # Create the HTTP response JSON object
        response = {}

        # Parse the get data. In case of error, the status is 0.
        cmd, status = network.parse_get_request(request)

        # Assign the command to the response KEY_TITLE
        response[config.KEY_TITLE] = cmd

        # If the status is 1, handle the command
        if status == 1:
            response = handle_request(cmd)
        else:
            response[config.KEY_STATE] = config.STATE_ERR
            response[config.KEY_MESSAGE] = config.MESSAGE_CMD_UNKNOWN
```

```
        # Send response to the client and close the connection
        network.send_response(cl, response, True)

    except OSError as e:
        network.ledstatus.off()
        cl.close()
        print('[ERROR] Network Connection closed')
```

# Domoticz Setup

## Devices

Create a virtual sensor, hardware dummy, named LED1 Control from sensor type Switch/Light.

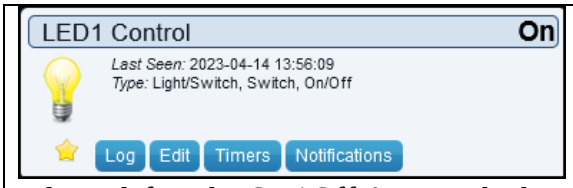
After creating the device, the Domoticz devices list shows the entry:

```
Idx=16, Hardware=VirtualSensors, ID=00014060, Unit=1, Name=LED1 Control,
Type=Light/Switch, SubType=Switch, Data=On
```

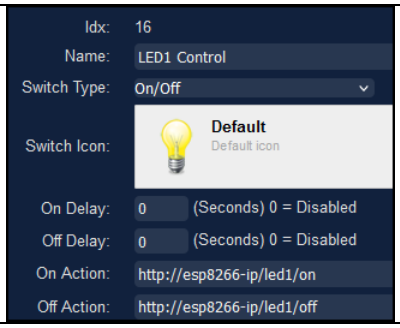
## Device Actions

For the switch device with IDX=16, two device actions are defined:

```
On Action: http://esp8266-ip/led1/on
Off Action: http://esp8266-ip/led1/off
```




Edit to define the On / Off Action which is an HTTP request to the NodeMCU web server.




## Automation Script

There is no automation script.
If required to take any other action rather switching the LED On/Off, create an dzVents script.

```
--[[
File:   esp8266_remotecontrol.dzvents
Date:20230224
Author: Robert W.B. Linn

:description
Listen to the state change of a switch device.
The state change is triggered by device actions http://esp8266-ip/led1/on or Off

:log
2023-04-14 14:02:38.682 VirtualSensors: Light/Switch (LED1 Control)
2023-04-14 14:02:38.676 Status: User: admin initiated a switch command (16/LED1
Control/Off)
2023-04-14 14:02:38.779 Status: dzVents: Info: Handling events for: "LED1 Control",
value: "Off"
2023-04-14 14:02:38.779 Status: dzVents: Info: template: ------ Start internal
script: esp8266_remotecontrol: Device: "LED1 Control (VirtualSensors)", Index: 16
2023-04-14 14:02:38.779 Status: dzVents: Info: template: Device LED1 Control state
changed to Off
2023-04-14 14:02:38.779 Status: dzVents: Info: template: ------ Finished
esp8266_remotecontrol
2023-04-14 14:02:41.070 VirtualSensors: Light/Switch (LED1 Control)
2023-04-14 14:02:41.067 Status: User: admin initiated a switch command (16/LED1
Control/On)
```

```
2023-04-14 14:02:41.117 Status: dzVents: Info: Handling events for: "LED1 Control",
value: "On"
2023-04-14 14:02:41.117 Status: dzVents: Info: template: ------ Start internal
script: esp8266_remotecontrol: Device: "LED1 Control (VirtualSensors)", Index: 16
2023-04-14 14:02:41.117 Status: dzVents: Info: template: Device LED1 Control state
changed to On
2023-04-14 14:02:41.117 Status: dzVents: Info: template: ------ Finished
esp8266_remotecontrol
]]--

IDX_SWITCH = 16

return {
 on = {
  devices = {
      IDX_SWITCH
  }
 },
 logging = {
  level = domoticz.LOG_INFO,
  marker = 'template',
 },
 execute = function(domoticz, device)
  domoticz.log(string.format('Device %s state changed to %s', device.name,
device.state), domoticz.LOG_INFO)
 end
}
```

# Appendix

# MQTT Autodiscover

## Description

Whilst starting to write this book, the projects require to create manually Domoticz devices. For example, the DHT22 Temperature + Humidity project requires to create manually a Domoticz device from type Temp + Humidity using the Dummy hardware controller.

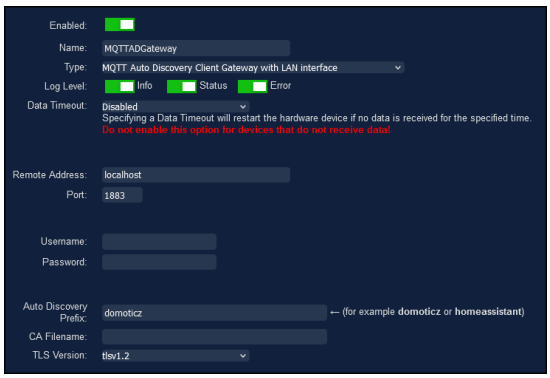
As from Domoticz version 2022.1 stable, the hardware MQTT Auto Discovery Client Gateway with LAN interface has been added.
This means that for a device, own state topics can be defined, handled by Domoticz.
No need to use the Domoticz topics “domoticz/in” or domoticz/out”.
A device state can be set by its own MQTT topic with payload.

To mention is that the MQTT broker mosquitto and the clients mosquitto_pub and mosquitto_sub run on the Domoticz Test System (Raspberry Pi 4B). The Domoticz hardware controller MQTT Client with LAN interface is added and enabled.

## Domoticz Setup

**MQTT Auto Discovery Client Gateway with LAN interface**
The first step is to add new hardware “MQTT Auto Discovery Client Gateway with LAN interface”.

**Hardware device settings**

| Screenshot | Settings |
|---|---|
|  | Name:<br>**MQTTADGateway**<br>Note: Used a short name for easier log reading.<br>Remote Address:<br>**localhost**<br>Note: The gateway runs on the same system but can be accessed remote via MQTT.<br>Port:<br>**1883**<br>Note: This is the default MQTT port.<br>Auto Discovery Prefix:<br>**domoticz**<br>Note: Ensure to use the same prefix as for MQTT client gateway. |

### Domoticz MQTT Debug

For exploring how to use the MQTT Autodiscover feature, it is helpful to log MQTT messages at debug level.

To enable (tested on Raspberry Pi) the mosquitto configuration file needs changes followed by a restart of Domoticz.

Modify the file "/etc/init.d/domoticz.sh" by a adding the two lines:

```
DAEMON_ARGS="$DAEMON_ARGS -loglevel normal,status,error,debug"
DAEMON_ARGS="$DAEMON_ARGS -debuglevel hardware"
```

Then restart Domoticz

```
sudo systemctl daemon-reload
sudo service domoticz restart
```

### Domoticz Log

Below is a snippet of the Domoticz log after restarting Domoticz.

The gateways MQTT client (MQTTClientGateway) and Autodiscover (MQTTADGateway) are connected to the localhost.

The MQTTADGateway discovers two devices.

```
2023-04-18 14:29:15.468 Status: MQTTClientGateway: Connecting to localhost:1883
2023-04-18 14:29:15.468 Status: MQTTADGateway: Connecting to localhost:1883
2023-04-18 14:29:15.669 Status: MQTTADGateway: connected to: localhost:1883
2023-04-18 14:29:15.669 Status: MQTTClientGateway: connected to: localhost:1883
2023-04-18 14:29:15.970 Status: MQTTADGateway: discovered: SGT001/Garden Temp
(unique_id: SGT001)
2023-04-18 14:29:15.970 Debug: MQTTADGateway: topic:
domoticz/sensor/sensorGardenT/config, message: {"name": "Garden Temp",
"device_class": "temperature", "state_topic": "domoticz/sensor/sensorGarden/state",
"value_template": "{{value_json.temperature}}", "unit_of_measurement": "°C",
"unique_id": "SGT001"}
2023-04-18 14:29:16.071 Status: MQTTADGateway: discovered: SGH001/Garden Hum
(unique_id: SGH001)
2023-04-18 14:29:16.070 Debug: MQTTADGateway: topic:
domoticz/sensor/sensorGardenH/config, message: {"name": "Garden Hum",
"device_class": "humidity", "state_topic": "domoticz/sensor/sensorGarden/state",
"value_template": "{{value_json.humidity}}", "unit_of_measurement": "%",
"unique_id": "SGH001"}
2023-04-18 14:29:16.171 Debug: MQTTADGateway: topic: domoticz/status, message:
online
```

# MQTT Hints

**MQTT Clients**
Both clients “mosquitto_pub” and “mosquitto_sub”, are used for testing configuration, publish, and subscribe MQTT messages i.e., topics with payload.
The clients are running from a terminal on Domoticz Test System (Raspberry Pi).
Using the command line clients enables to understand the MQTT message topics and payloads.
These are used by the MicroPython scripts in the various projects.
So, for every project the command line clients are used prior developing the MicroPython code. This is essential.

**Remote Access**
To be able to access MQTT from remote clients, the mosquitto configuration require to “allow anonymous” and a listener on the default port.

Edit the Mosquitto configuration file (this is done on a Raspberry Pi)

```
sudo nano /etc/mosquitto/conf.d/mosquitto.conf
```

Add the lines

```
allow_anonymous true
listener 1883 0.0.0.0
```

Restart Mosquitto service

```
sudo service mosquitto restart
```

Check the mosquitto log message on errors

```
sudo cat /var/log/mosquitto/mosquitto.log
```

**Test Remote Access**
To test if able to access remotely, like publishing a message to update the temperature & humidity of the devices assigned to the object with id sensorGarden.

```
mosquitto_pub -h domoticz-ip -p 1883 -t "domoticz/sensor/sensorGarden/state" -m
'{"temperature":21.9,"humidity":71}'
```

**Remove Retained Messages**

```
mosquitto_sub -h localhost --remove-retained -t '#' -W 1
```

During testing a whole bunch of messages are published. Lots with errors. These are kept in the MQTT queue. It is therefore required to remove.
Here an example log output from the remove-retain command

```
{"name": "Garden Temp", "cmd_t" : "~/set", "stat_t" : "~/state", "uniq_id":"gt123"
}
{"name": "Garden Temp Sensor", "device_class": "temperature", "state_topic":
"domoticz/sensor/garden/tempsensor/state"}, "value_template": "{{
value_json.temperature|default(0)}}"
```

```
{"name": "plant_sensor_1", "state_topic": "domoticz/plants/1/state",
"value_template": "{{ value_json.temperature|default(0)}}"
{"name": "plant_sensor_1", "state_topic": "domoticz/plants/1/state"
{"name": "plantsensor1", "device_class": "temperature", "state_topic":
"domoticz/plants/1/state", "value_template": "{{ value_json.temperature|default(0)
}}"
{"name": "Garden Temp", "device_class": "temperature", "state_topic":
"domoticz/sensor/sensorGarden/state", "value_template": "{{
value_json.temperature}}", "unit_of_measurement": "°C" }
{"name": "Garden Hum", "device_class": "humidity", "state_topic":
"domoticz/sensor/sensorGarden/state", "value_template": "{{ value_json.humidity}}",
"unit_of_measurement": "%RH" }
```

IMPORTANT
Restart Domoticz after removing the retained messages and check the Domoticz log.

```
sudo service domoticz restart
```

# MQTT Autodiscover Hints

Some hints gathered during testing.

**Sensor Multiple Measurement Values**
The Autodiscover feature does not allow to create a Temp+Humidity device but require two devices Temperature and Humidity instead.
These are created by submitting two configuration topics using the same state topic.

**IP-Address**
The ip address used was localhost or 127.0.0.1 (because running from local terminal) but can also access remote, like domoticz-ip (see previous Hints > Remote Access).

**Unit of Measure**
Important to use the correct unit_of_measure.
Example if for a Humidity device the unit was set to %RH instead %, the device created was from type "General" instead "Humidity".
See Domoticz source code

```
void MQTTAutoDiscover::GuessSensorTypeValue(...
```

**Device Unique ID**
Ensure that each device has a unique id (key "unique_id") in the configuration message.
*Hint*
If familiar with Piping and Instrument Diagrams, an option is to use for the device an Instrumentation ID, like for the temperature TR001.

**Clean All Retained Messages**
To start with a clean sheet, all MQTT retained messages are removed, the related Domoticz devices deleted, and the Domoticz Test System is restarted.

Example with command and output:

```
mosquitto_sub -h localhost --remove-retained -t '#' -W 1

{"name": "Garden Temp", "device_class": "temperature", "state_topic":
"domoticz/sensor/sensorGarden/state", "value_template":
"{{value_json.temperature}}", "unit_of_measurement": "°C", "unique_id": "SGT001"}
{"name": "Garden Hum", "device_class": "humidity", "state_topic":
"domoticz/sensor/sensorGarden/state", "value_template": "{{value_json.humidity}}",
"unit_of_measurement": "%", "unique_id": "SGH001"}

sudo service domoticz.sh restart
```

# Example Devices

## Sensor Temperature & Humidity

**Create Devices Temperature, Humidity**

```
mosquitto_pub -r -h localhost -p 1883
-t "domoticz/sensor/sensorGardenH/config"
-m '{"name": "Garden Hum", "device_class": "humidity", "state_topic":
"domoticz/sensor/sensorGarden/state", "value_template":
"{{value_json.humidity}}", "unit_of_measurement": "%", "unique_id": "SGH001"}'

mosquitto_pub -r -h localhost -p 1883
-t "domoticz/sensor/sensorGardenT/config"
-m '{"name": "Garden Temp", "device_class": "temperature", "state_topic":
"domoticz/sensor/sensorGarden/state", "value_template":
"{{value_json.temperature}}", "unit_of_measurement": "°C", "unique_id":
"SGT001"}'
```

**Publish to Set values**

```
mosquitto_pub -h localhost -p 1883
-t "domoticz/sensor/sensorGarden/state"
-m '{"temperature":22.1,"humidity":64}'
```

**Subscribe to Get values**

```
mosquitto_sub -h localhost -p 1883
-t "domoticz/sensor/sensorGarden/#"

{"temperature":22.1,"humidity":64}
```

**Domoticz Devices List**

```
IDX=33, Hardware=MQTTADGateway, ID=SGH001, Unit=1, Name=Garden Hum,
Type=Humidity, SubType=LaCrosse WS2300, Data=Humidity 64 %

IDX=34, Hardware=MQTTADGateway, ID=SGT001, Unit=1, name=Garden Temp, Type=Temp,
SubType=THR128/138, THC138, Data=22.1 C
```

**Remove Retained Message**

```
mosquitto_sub -h localhost --remove-retained
-t 'domoticz/sensor/sensorGardenH/config' -W 1

mosquitto_sub -h localhost --remove-retained
-t 'domoticz/sensor/sensorGardenT/config' -W 1
```

# Push On Button

**Create Device**

```
mosquitto_pub -r -h 127.0.0.1 -p 1883
-t "domoticz/button/makelab/config"
-m '{"name": "MakeLabButton", "state_topic": "domoticz/button/makelab/state",
"unique_id": "BM001"}'
```

**Publish to Set values**

```
mosquitto_pub -h 127.0.0.1 -p 1883
-t "domoticz/button/makelab/state"
-m {'ON'}
OR
-m ON
```

**Subscribe to Get values**

```
mosquitto_sub -h localhost -p 1883
-t "domoticz/button/makelab/#"

ON
```

**Domoticz Devices List**

```
IDX=39, Hardware=MQTTADGateway, ID=BM001, Unit=1, Name=MakeLabButton,
Type=Light/Switch, SubType=Switch, Data=Off
```

**Remove Retained Message**

```
mosquitto_sub -h localhost --remove-retained
-t 'domoticz/button/makelab/config' -W 1
```

# MicroPython Development Hints

(in progress)

| Item | Hint |
|---|---|
| Python | PEP 8 – Style Book for Python Code |
| REPL Command for Help on Modules<br>>>> help('modules') | __main__ lwip uasyncio/lock ure<br>_boot math uasyncio/stream urequests<br>_boot_fat micropython ubinascii uselect<br>_onewire mip/__init__ ucollections usocket<br>_rp2 neopixel ucryptolib ussl<br>_thread network uctypes ustruct<br>_uasyncio ntptime uerrno usys<br>_webrepl onewire uhashlib utime<br>builtins rp2 uheapq<br>uwebsocket<br>cmath uarray uio uzlib<br>dht uasyncio/__init__ ujson webrepl<br>ds18x20 uasyncio/core umachine<br>webrepl_setup<br>framebuf uasyncio/event uos<br>gc uasyncio/funcs urandom<br>Plus, any modules on the filesystem |
| REPL Command for Help on rp2 module<br>>>> import rp2<br>>>> help('rp2') | object rp2 is of type str<br>find -- <function>, rfind -- <function><br>index -- <function>, rindex -- <function><br>join -- <function>, split -- <function><br>splitlines -- <function>, rsplit -- <function><br>startswith -- <function>, endswith -- <function><br>strip -- <function>, lstrip -- <function><br>rstrip -- <function>, format -- <function><br>replace -- <function>, count -- <function><br>partition -- <function>, rpartition -- <function><br>center -- <function>, lower -- <function><br>upper -- <function>, isspace -- <function><br>isalpha -- <function>, isdigit -- <function><br>isupper -- <function>, islower -- <function><br>encode -- <function> |
| | |
| | |

# Abbreviations

Some of the abbreviations used. These apply esp. for the scripts.

| | |
|---|---|
| CLI | Terminal Command Line Interface |
| domoticz-ip:port | Domoticz system IP address and port |
| ~domoticz | Path to the Domoticz folder (Home Directory) on the RPi |
| dzVents | Domoticz Easy Events |
| GUI | Domoticz UI in Web Browser |
| Hm, HmIP | Homematic, HomematicIP |
| JSON | JavaScript Object Notation |
| MCU | Microcontroller Unit (Raspberry Pi Pico/Pico W, ESP8266, ESP32) |
| MQTT | Message Queuing Telemetry Transport |
| NodeMCU | ESP8266 MCU |
| Pico W or PicoW | Raspberry Pi Pico W |
| picow-ip | Raspberry Pi Pico W web server IP address |
| RPi | Raspberry Pi |
| webserver | Web Server |