Pico-MQTT-Module

Firmware Version 2.02
User Guide
Updated June 10th, 2025

Add-on C-language module to merge with your existing (C-language) program / project, giving it access to the MQTT protocol.

IMPORTANT :

This User Guide is about Pico-MQTT-Module Firmware Version 2.02 from Andre St. Louys. To get the full potential of this module, you must NOT simply clone the GitHub repository. Read this User Guide and carefully follow the indicated procedure.

Join our Pico-MQTT-Module discussion group on:
https://github.com/astlouys/Pico-MQTT-Module/discussions

Pico-MQTT-Module User Guide

Index

Pico-MQTT-Module User Guide

Introduction

NOTE: Explaining what is the MQTT protocol is beyond the scope of this User Guide. For those who need a crash course on it, you can read the Dhairya Parikh's excellent book: "Raspberry Pi and MQTT Essentials" (available from Amazon, among others). You may also refer to the MQTT organisation at https://mqtt.org

MQTT ("Message Queueing Telemetry Transport") is a lightweight publish-subscribe protocol for message transfer. It is popular in IoT devices given its small constraints on resources and network bandwidth. In the context of the Pico-MQTT-Module, we will use the available home Wi-Fi network to send and receive ("publish and subscribe") the MQTT messages.

The Pico-MQTT-Module is a C-Language add-on module that will give your existing C-language Pico program or project access to the MQTT protocol. Be aware that the Pico-MQTT-Module allows your Pico program to become a "MQTT client", but you also need a MQTT server ("MQTT broker" in MQTT language) to manage message passing between members of the MQTT network. Parikh's book mentioned above clearly explains that and also shows you how to deploy a MQTT broker on a Raspberry Pi.

To help you figure out the details on how to use the Pico-MQTT-Module, an example is included in the repository: « Pico-MQTT-Example ». This is a basic C-Language program making use of the Pico-MQTT-Module. This simple program is briefly described later in this User Guide. Again, you need the deployment of a MQTT broker ("server") to get something out of the Pico-MQTT-Example program.

To get the full potential of the Pico-MQTT-Module, make sure you carefully follow the instructions given in this User Guide.

1) The « Setup part 1 » covers the steps required to install the Pico-MQTT-Example to your system. Make sure everything is installed as required and that you can build and run the program without problem. You can then go on with the following step to merge the add-on module to your own program.

2) The « Setup part 2 » covers the steps to follow when you want to add the Pico-MQTT-Module to one of your own existing C-Language program / project.

NOTE: « Pico » is used throughout this User Guide to describe the microcontroller used but of course, since we are using the existing Wi-Fi network to subscribe and publish messages, a PicoW is required (the « plain Pico » does not have the circuitry to allow Wi-Fi communications). I didn't have the chance to work with the Pico2W so far, but I assume there should not be major changes, if not for the CMakeLists.txt where I assume you must specify which microcontroller is used and which cores. I'm interested in hearing from your experience in adapting this Firmware on the new Pico2W...

Pico-MQTT-Module User Guide

You can join other users of the Pico-MQTT-Module on the discussion group on:
https://github.com/astlouys/Pico-MQTT-Module/discussions
I can also be reached with my personal email at:
astlouys@gmail.com

Pico-MQTT-Module setup

Setup part 1: Installing / copying the Pico-MQTT-Module to your system.

NOTE: The instructions below assume that your development system is a Raspberry Pi running the Raspberry Pi O.S. (previously called "Raspbian") and that you're using Visual Studio Code. It also assumes that you followed the recommended naming and structure conventions for the directory structure on your development system. If this is not the case, you will have to adapt the instructions accordingly.

1) Before anything else, if you did not already done so, refer to the Pico-WiFi-Module in my GitHub repository and install it to your system while carefully following the Pico-WiFi-Module User Guide.

2) When you're done with the Pico-WiFi-Module installation, create the MQTT project directory on your system:
```
mkdir  /home/pi/pico/Pico-MQTT-Module
```

3) Clone / copy all files from my Pico-MQTT-Module GitHub repository to the « /home/pi/pico/Pico-MQTT-Module » project directory created in the previous step above.

4) While in the « /home/pi/pico/Pico-MQTT-Module » project directory create a symbolic link pointing to the pico_sdk_import.cmake file. This way, if / when you update the Pico SDK, your project will always use the latest version when it is built:
```
ln  -s  /home/pi/pico/pico-sdk/external/pico_sdk_import.cmake
```

5) Create a new « UTILITIES » directory in the pico directory (should have already been done while you installed the Pico-WiFi-Module in step 1 above). I use to put in this directory bash scripts and other utilities used while developing project for the Pico:
```
mkdir  /home/pi/pico/UTILITIES
```

6) Move the file « baseline.h » from the Pico-MQTT-Module project directory to the UTILITIES directory just created (should have already been done while you installed the Pico-WiFi-Module in step 1 above):
```
mv  baseline.h  /home/pi/pico/UTILITIES
```

7) Create a symbolic link from the project directory to the « baseline.h » file:
```
ln  -s  /home/pi/pico/UTILITIES/baseline.h
```

8) The baseline.h file contains basic definitions that I almost always use in most projects. You may want to take a look at it and add your own pieces of code that you use often. When you update the « baseline.h » file in the UTILITIES directory, it is automatically updated in all your projects through the symbolic link.

9) Move the file « get_pico_identifier.c » from the project directory to the UTILITIES directory:

```
mv  get_pico_identifier.c  /home/pi/pico/UTILITIES
```

This C-Language file (containing a single function) will be included in your program. It makes a correspondence between the Pico Unique ID you use in a specific device and a unique "device name" that will be used in further MQTT projects. You will want to edit this file and replace the Pico Unique IDs listed by the Unique IDs of the Pico's that you are using and give them a device name that is significant for you. For example, if you want to send a MQTT message to a device in the kitchen, you may specify "kitchen" for the MQTT topic and only the "kitchen" device will receive the message. Since all devices on your network must agree on "who-is-who", it is a good idea to keep all those device identifications in a center place that can be updated when a new device is added.

10) Create two symbolic links from the project directory to the Pico-WiFi-Module files:

```
ln  -s  /home/pi/pico/Pico-WiFi-Module/Pico-WiFi-Module.c
ln  -s  /home/pi/pico/Pico-WiFi-Module/Pico-WiFi-Module.h
```

11) Use your text editor to modify the .bashrc file...

```
sudo  nano  /home/pi/.bashrc
```

12) ...and add the following lines:

```
export MQTT_BROKER_IP=address of your MQTT broker here
export MQTT_PASSWORD=your MQTT password here
```

This will create the environment variables containing your MQTT broker IP address on your Wi-Fi network and the MQTT password to be used when logging to the MQTT broker (read next paragraph for more explanations). It is a good idea to refer to your router's User Guide to see how you can assign a specific static IP address to your MQTT broker. If you don't, it may happen (in case of a reboot, power failure or the like) that your MQTT broker will change its IP address. You may wonder why your MQTT devices don't work anymore and chase the bug for a long time before realising that it is simply because the broker IP address has changed.

13) The CMakeLists.txt file of the Pico-MQTT-Example could be used as an example. It reads the environment variables for your MQTT IP address and MQTT password, and makes them available to the program being built. This way, even if you copy and / or give away your source code, your network name and password are never included in the files (as long as you don't give the executable file where they will be put during the build process). NOTE: The environment variables will be created / generated only the next time you open a session (or next time you reboot to make things simple).

14) This completes the first part of the setup. At this point, make sure you are able to build and use the Pico-MQTT-Example application. Refer to the section of this User Guide on how to use it.

Setup part 2: Integrating the Pico-MQTT-Module to one of your projects.

NOTE: The instructions below assume that you development system is a Raspberry Pi running the Raspberry Pi O.S. (previously called « Raspbian ») and that you're using Visual Studio Code. It also assumes that you followed the recommended naming and structure conventions for the directory structure on your development system. If this is not the case, you will have to adapt the instructions accordingly.

Make sure you carefully followed the instructions in the « Setup part 1 » above, since they are a prerequisite for this section.

1) From your project directory: « /home/pi/pico/Pico-My-Project », create a symbolic link pointing to both the « .c » and « .h » files of the Pico-WiFi-Module:
```
ln  -s  /home/pi/pico/Pico-WiFi-Module/Pico-WiFi-Module.c
ln  -s  /home/pi/pico/Pico-WiFi-Module/Pico-WiFi-Module.h
```

2) From your project directory: « /home/pi/pico/Pico-My-Project », create a symbolic link pointing to both the « .c » and « .h » files of the Pico-MQTT-Module:
```
ln  -s  /home/pi/pico/Pico-MQTT-Module/Pico-MQTT-Module.c
ln  -s  /home/pi/pico/Pico-MQTT-Module/Pico-MQTT-Module.h
```

3) From your project directory: « /home/pi/pico/Pico-My-Project », create a symbolic link pointing to the « baseline.h » file:
```
ln  -s  /home/pi/pico/UTILITIES/baseline.h
```

4) From your project directory: « /home/pi/pico/Pico-My-Project », create a symbolic link pointing to the « get_pico_identifier.c » file:
```
ln  -s  /home/pi/pico/UTILITIES/get_pico_identifier.c
```

   You should edit this file so that the specific Pico used in your project is given a unique "device ID", allowing you to send a message only to a specific device if you wish to do so. This is the way the "Topic" will be used in the other members of the ASTL Smart Home ecosystem in my repositories. If you're not planning to use it this way, you may proceed another way.

5) In your project file: « /home/pi/pico/Pico-My-Project.c », add the include file « Pico-WiFi-Module.h » file:
```
#include "Pico-WiFi-Module.h"
```

6) In your project file: « /home/pi/pico/Pico-My-Project.c », add the include file « Pico-MQTT-Module.h » file:
```
#include "Pico-MQTT-Module.h"
```

7) In your project file: « /home/pi/pico/Pico-My-Project.c », add the file « get_pico_identifier.c » file:
```
#include "get_pico_identifier.c"
```

   This file contains a function. I would recommend inserting it where you would normally write the "get_pico_identifier()" function (in alphabetical order ?). Cut-and-paste the function declaration to the beginning of your program, with other function declarations.

8) In your project file: « /home/pi/pico/Pico-My-Project.c », declare the main Wi-Fi structure defined in the Pico-WiFi-Module.h file:
```
struct  struct_wifi  StructWiFi;
```

9) As indicated in the Pico-WiFi-Module.h file, some members of the Wi-Fi structure must be initialized / specified by the user. If you properly followed the instructions in this User Guide, the Wi-Fi network name and network password should have already been extracted from environment variables. Adjust the country code according to your country. You may refer to the CYW43 documentation for your specific country. This allows the PicoW to use the Wi-Fi frequencies used in your country.

10) In your project file: « /home/pi/pico/Pico-My-Project.c », declare the main MQTT structure defined in the Pico-MQTT-Module.h file:
```
struct  struct_mqtt  StructMQTT;
```

11) Many members of the MQTT structure must be initialized / specified by the user. I suggest that you refer to the Pico-MQTT-Example program source code to see how to cover these initializations.

12) Copy-paste the function « log_info() » to your program since it is referenced in Pico-WiFi-Module (in future releases, the function log_info() may be moved to the UTILITIES directory, similar to the get_pico_identifier() function).

13) Remember that your CMakeLists.txt file must be properly configured / adapted to read the environment variables for the network name, network password, MQTT broker IP address and MQTT password. It must also make reference to all required libraries for your specific program / project. You may want to copy the CMakeLists.txt file from the Pico-MQTT-Module directory to your project directory and modify it accordingly.

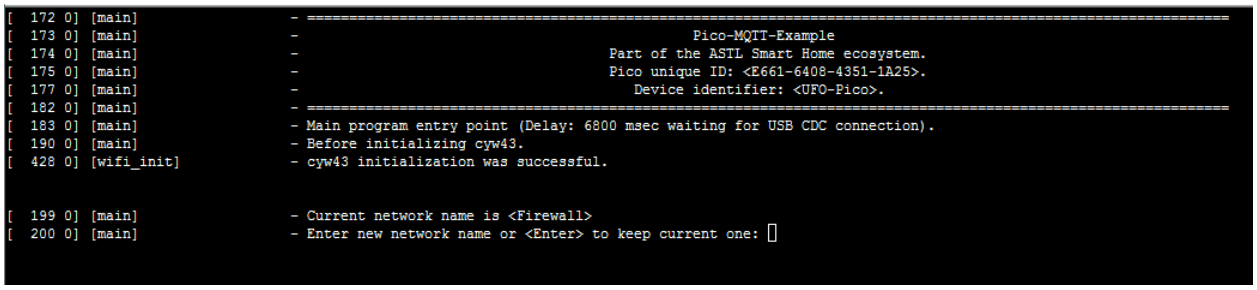
Pico-MQTT-Example

Build the Pico-MQTT-Example application and download it to your Pico. The application will automatically be launched when uploaded to the Pico.

Once started, the application will wait for a USB CDC connection. So, start you preferred terminal emulator program while connected to the Pico. You may have to setup the serial port of the terminal program to enable the communication between it and the Pico. For those not familiar with this, I strongly recommend version 5 of the TeraTerm program. Its features make it an excellent choice. If you use it frequently, consider sending a contribution to the developer.

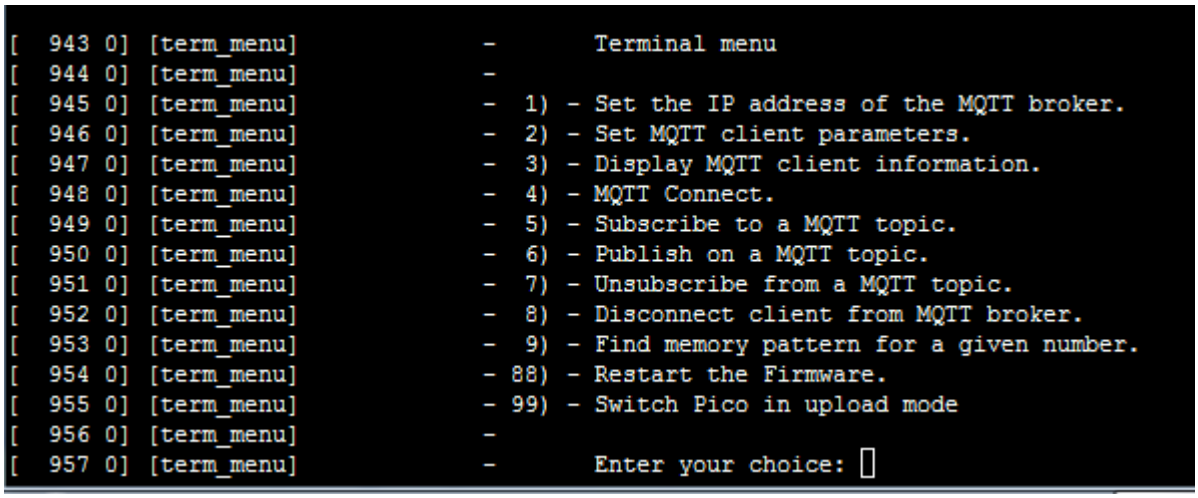
When the USB CDC connection has been established, it first displays a title including the Pico Unique ID (see it as the "Pico serial number") that you must use in the "get_pico_identifier.c" file to identify this specific device. If your Pico Unique ID has not been recognized, it shows the default device name "UFO-Pico" (it is OK for now, but this won't work if you have more than one device with the same name on your MQTT network).

Then, you have the opportunity to change the Wi-Fi SSID ("network name") and Wi-Fi password, in case something went wrong with the environment variables that were setup during Firmware installation.



After logged to your Wi-Fi network, you also have the opportunity to change de MQTT broker IP address and MQTT password in case something went wrong with the environment variables that were setup during Firmware installation..

You will then see a menu on the screen. Most items are self-explanatory:



NOTE: As a general guideline, menu choices should be executed in sequence, beginning with choice number 1 and going down with choices number 2, 3, and so on and so forth.

REMINDER: There must be a MQTT broker (“server”) running on your Wi-Fi network, otherwise, you won’t be able to get anything out of this Firmware.

1) « Set the IP address of the MQTT broker »: Allows you change / modify the IP address of your MQTT broker.

2) « Set MQTT client parameters »: Currently not implemented. If you need to change some of these parameters, it must be done in the Firmware itself (“hardcoded”) and the Firmware rebuilt.

3) « Display MQTT client information »: Displays many items relative the MQTT client. One important information is toward the end of the display and gives the current connect status, indicating if the MQTT client (the “Pico-MQTT-Example” Firmware) is currently connected to the MQTT broker. Note that the client will automatically connect to the broker when the Firmware is launched.

4) « MQTT connect »: Force a connection to the MQTT broker.

5) « Subscribe to a MQTT topic »: Subscribe the MQTT client (the “Pico-MQTT-Example” Firmware) to a particular topic.

6) « Publish on a MQTT topic »: The MQTT client (the “Pico-MQTT-Example” Firmware) publishes the specified payload on the specified topic.

7) « Unsubscribe from a MQTT topic »: Unsubscribe the MQTT client (the “Pico-MQTT-Example” Firmware) from the specified topic.

8) « Disconnect client from MQTT broker »: Disconnect the MQTT client (the "Pico-MQTT-Example" Firmware) from the MQTT broker.

9) « Find memory pattern for a given number »: Probably not useful for you. I used this Firmware to replace / change memory on a specific device. This function allowed me, for example, to find what is the memory representation of the number "unsigned int_16 56382" (or unsigned int_64). I could then send an MQTT command to replace a specific memory location (in RAM) by the given memory pattern.

10) « Restart the Firmware »: Not much to say about that option !

11) « Switch Pico in upload mode »: Allows the user to upload a new firmware to the Pico without having to make a power cycle while holding down the « BootSel » Pico button (menu choice number 99).