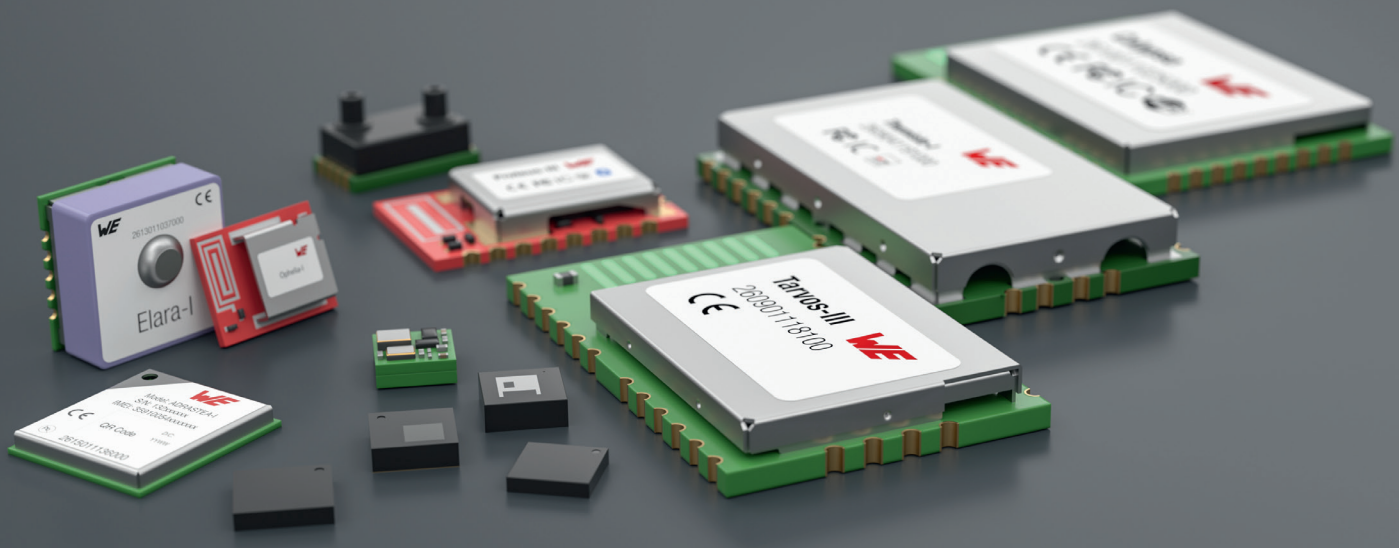

ANR007

IoT application based on Calypso module

Version 1.5

July 19, 2023

WÜRTH ELEKTRONIK MORE THAN YOU EXPECT

Revision history

Document version	Notes	Date
1.0	• Initial version	April 2019
1.1	• Updated file name to new AppNote name structure. Updated important notes, legal notice & license terms chapters.	June 2019
1.2	• Updated address of Division Wireless Connectivity & Sensors location	October 2019
1.3	• Updated reference 1 link (MQTT specification).	October 2019
1.4	• Repaired Raspian link	January 2021
1.5	• Updated `Important notes`, meta data and document style	July 2023

Contents

1 IoT application **3**
1.1 Introduction . . . 3
1.2 The IoT stack . . . 3
1.3 IoT application example . . . 5
1.3.1 Requirements . . . 5
1.3.2 System architecture . . . 6
1.4 System design using Calypso . . . 7
1.5 IoT application demo . . . 9

2 MQTT **10**
2.1 Publish/Subscribe . . . 10
2.2 Topics/Subscriptions . . . 10
2.3 QoS . . . 11
2.4 Message persistence . . . 11
2.5 Last Will and Testament . . . 11
2.6 Security . . . 11

3 IoT demo using Calypso evaluation board **12**
3.1 Prerequisites . . . 12
3.2 Demo set-up . . . 12
3.3 Eclipse Mosquitto . . . 13
3.4 Installing Eclipse Mosquitto MQTT broker on Raspberry Pi . . . 13
3.5 Setting-up Wi-Fi network . . . 14
3.6 Connecting Raspberry Pi to the network . . . 14
3.7 Connecting Calypso to the network . . . 14
3.7.1 Connecting Calypso to PC . . . 14
3.7.2 Connect to an access point . . . 15
3.8 Connecting smart phone to the network . . . 15
3.9 Starting MQTT broker on Raspberry Pi . . . 16
3.10 Starting MQTT client on Calypso . . . 16
3.11 Starting MQTT client on smart phone . . . 16
3.12 Data exchange . . . 18
3.12.1 Subscribe from Calypso evaluation board . . . 18
3.12.2 Subscribe from smart phone . . . 18
3.12.3 Publish from Calypso evaluation board . . . 19
3.12.4 Publish from smart phone . . . 20
3.13 Further enhancements . . . 21

4 Summary **22**

5 References **23**

6 Important notes **24**

1 IoT application

1.1 Introduction

The Internet of Things (IoT) can be broadly defined as an umbrella term for a range of technologies that enable devices to connect and interact with each other. Interacting devices and the data they generate, provide for a range of new applications. Industrial automation, healthcare, home automation, smart cities, smart grids and smart farming are some of the areas in which IoT provides substantial benefits.

Dubbed the "fourth industrial revolution" or Industry 4.0, the Industrial IoT (IIoT) is the digitization of industrial assets and processes that connects products, machines, services, locations/sites to workers, managers, suppliers, and partners. Closer networking of the digital world with the world of machines holds the potential for profound changes in global industry and, therefore, for many aspects of private and social life - to the way we work and live.

The core task of any IoT solution is to get data from the field to the cloud where analysis of the same generates the desired value proposition for the application. This application note aims to propose an elegant solution to achieve this task based on the Calypso Wi-Fi module.

This chapter begins by describing the parts of a typical IoT application. Further, an application scenario is discussed using a smart home example. Finally, a solution using the Calypso Wi-Fi module is presented.

1.2 The IoT stack

Irrespective of the area of application, an end-to-end IoT solution consists of the following components (figure 1). Explosive growth of IoT market has called for an industry standard that defines how various layers of the stack interact with each other. However, practical implementation of the same remains a challenge.

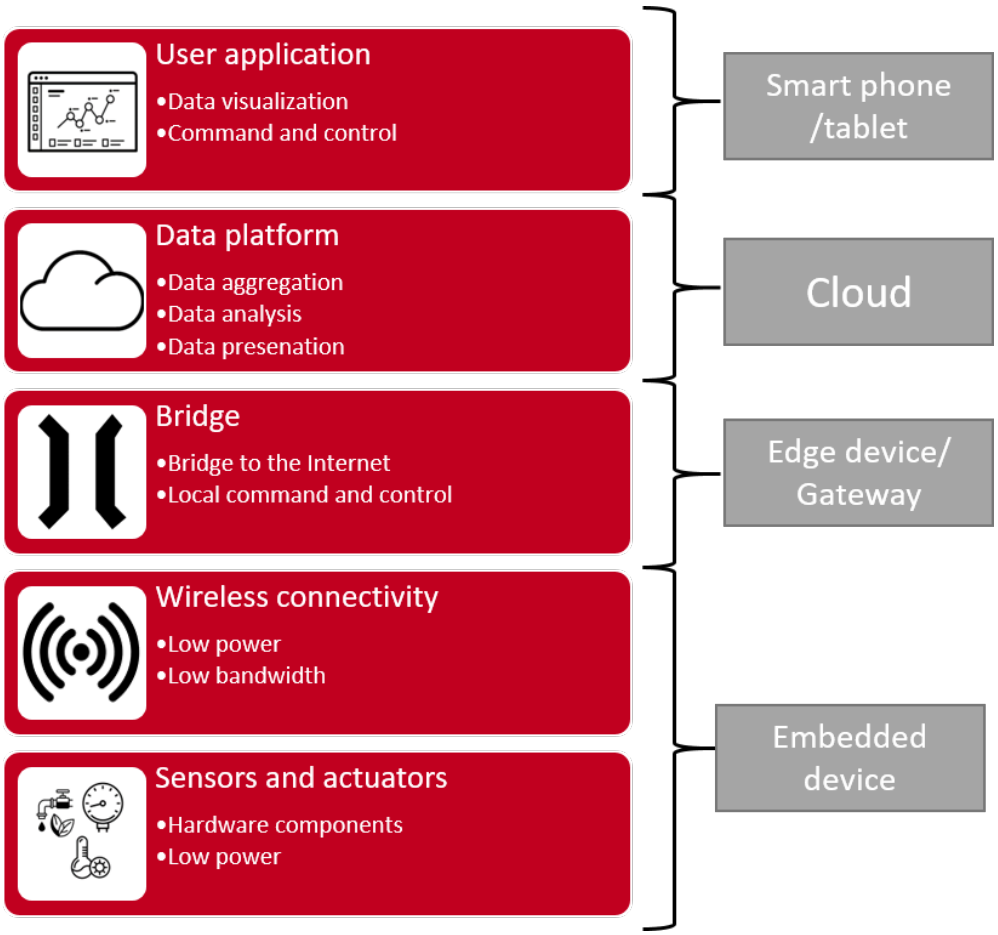

Figure 1: IoT application stack

- **Sensors and actuators:** This is a part of the system that directly interfaces with the physical environment. Sensors measure the state of the environment and interpret the same as digital data. On the other hand, actuators activate a physical change in the measured environment. Advances in the field of electronics in general and semiconductors in particular has led to the availability of a wide range of sensor and actuators which are highly efficient and yet very compact.
- **Wireless connectivity:** Sensors and actuators are typically installed in devices with limited access to the digital world. Consider, for example, a temperature sensor mounted inside an industrial boiler. Wireless connectivity provides in addition to many advantages the reachability necessary for such applications. A wide variety of standard as well proprietary wireless connectivity solutions are available today. A number of factors including range, throughput, spectrum regulations, local statutory requirements and power budget determine the choice of wireless connectivity solution.

  Modern embedded designs usually combine the above components into a single embedded device interacting with a gateway.
- **Gateway device:** A gateway device acts as a bridge between the physical and the digital worlds. It interprets the multitude of wireless connectivity protocols, collects the data and

forwards the same in a format understood by entities above. In certain applications the gateway device also performs basic analytics like threshold detection.

- **Data platform:** This is the platform where the data is finally stored and presented for further analysis. Options here can range from a local database to cloud server with redundancies. The data platform enables the use of technologies like Artificial Intelligence (AI) and Machine Learning (ML) to perform advanced data analytics that generates value additions to the application.
- **User application:** This is the interface between the human users and the digital world. Here the status of the observed environment is presented in human readable format. The user can take the necessary actions by interacting with this application.

Artificial intelligence and big data analytics creates a potential for huge advances in productivity, efficiency, and cost savings. The IIoT creates a universe of sensors that enables an accelerated deep learning of existing operations. These data tools allow for rapid contextualization, automatic pattern, and trend detection. Furthering this for manufacturing operations will finally allow for true quantitative capture of qualitative operations.
In the next section a sample IoT application is considered.

1.3 IoT application example

"Smart home" or "connected home" is a typical use case for IoT. Networking individual devices (sensors and actuators) used in a common household can result in an intelligent system which provides several benefits including energy efficiency, cost savings and general well-being.

1.3.1 Requirements

A typical smart home application has the following key features.

- Heating ventilation and air conditioning (HVAC).
- Access control (door and windows).
- Security and surveillance.
- Lighting control.
- Control home appliances or consumer electronics.
- Energy management.

The preferred ways of interaction with humans would be

- Computer via web-interface.
- Mobile applications.
- Smart voice assistants.

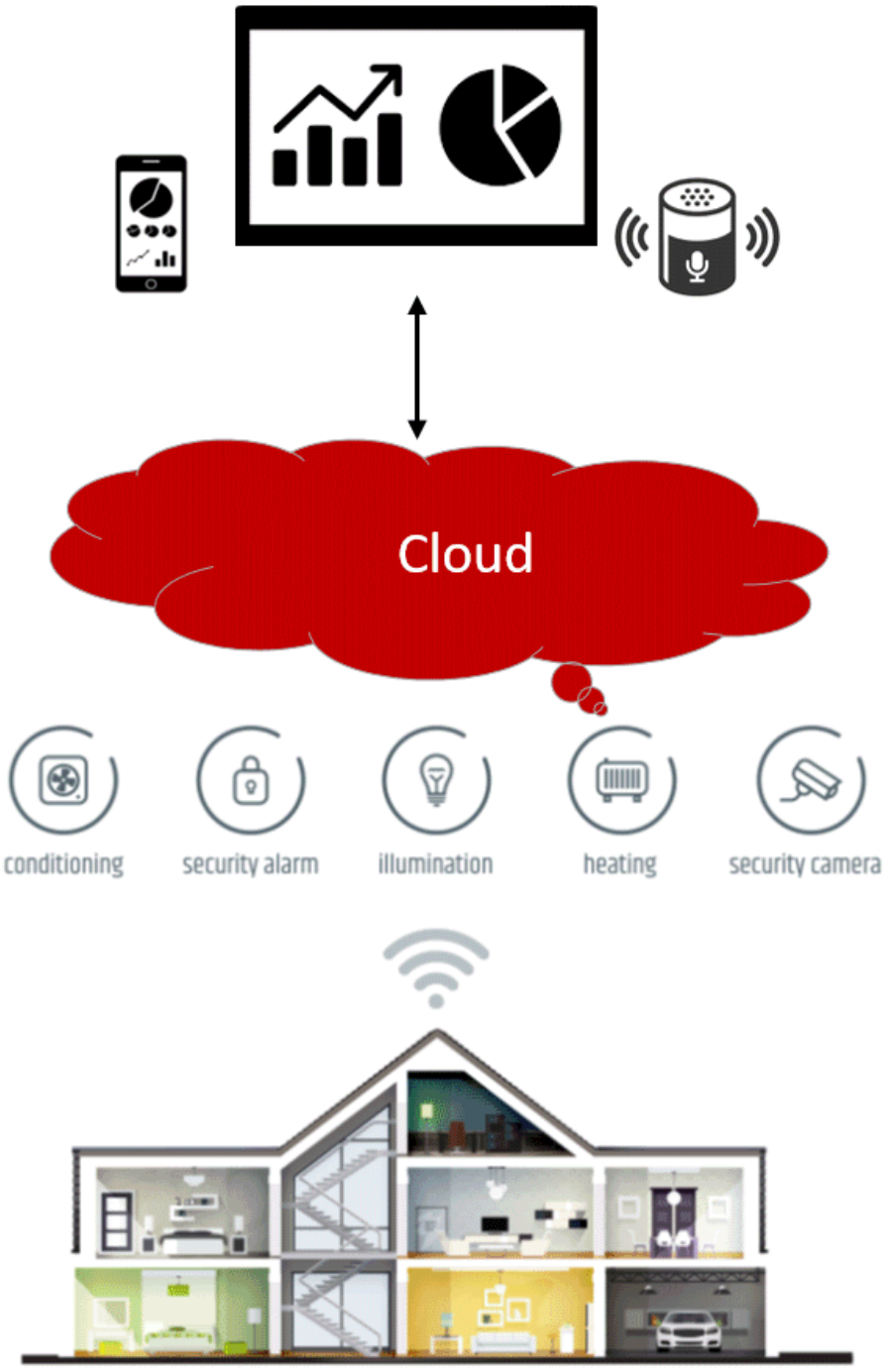

Figure 2: Smart home example

1.3.2 System architecture

Each of the above-mentioned tasks require sensors/actuators to interact with the environment, a wireless connectivity method, a gateway to collect the data, a cloud platform to store data and a user interface to enable human interaction. The architecture of such a system is as shown in figure 3.

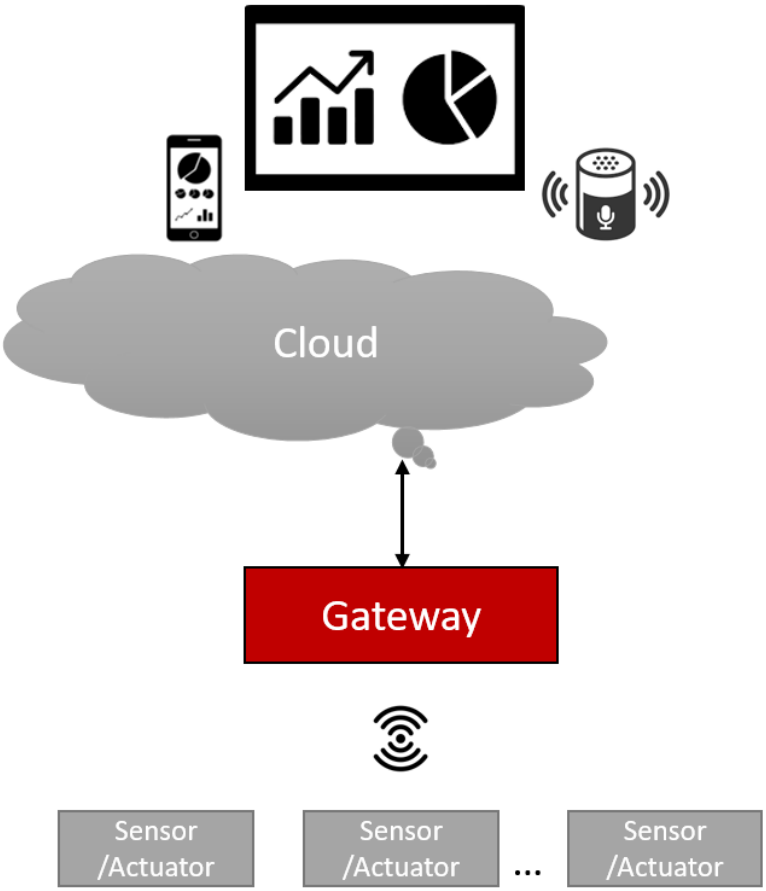

Figure 3: System architecture

1.4 System design using Calypso

The Calypso is a compact Wi-Fi radio module based on IEEE 802.11 b/g/n (2.4 GHz). With an on-board TCP/IP stack and MQTT protocol implemented out-of-the-box, Calypso acts the perfect building block for an IoT application. Figure 4 illustrates the design that realizes the architecture described in 1.3.2.

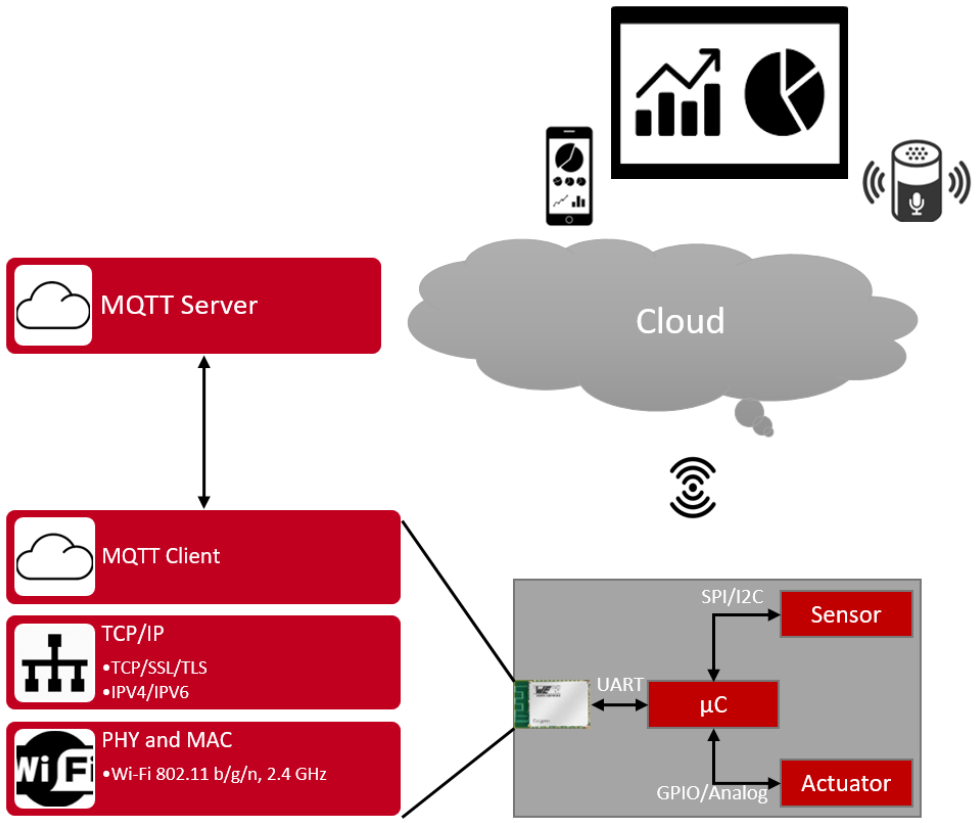

Figure 4: System design

At the hardware level, a host microcontroller connects and controls sensors/actuators over standard interfaces like I2C, SPI, GPIO or analog. The Calypso uses UART as the primary interface to the host. Being Wi-Fi compliant, the module can be configured to connect to the local infrastructure network (via AP).

Further, the MQTT-client implemented on-board the Calypso can be configured to connect to a broker/server running in the network. Most of the commonly used cloud platforms today use MQTT protocol to connect devices and exchange data. Hence, Calypso provides a direct communication link to the cloud without the need of a gateway device in between.
Here are a few advantages provided by this design approach:

- **Easy hardware integration:** Calypso's compact form factor, edge castellated connection and smart antenna selection allows easy integration into any hardware design. The smart antenna configuration enables the user to either use the on-board PCB antenna or an external antenna. With UART as the standard interface, the Calypso can be interfaced with most of the standard host microcontrollers.
- **Connection to existing network:** Wi-Fi is one of the most commonly used wireless connectivity technologies. Today, most homes already have Wi-Fi infrastructure in use. Hence, integration into the existing wireless network is easier and does not require an extra bridge device.
- **Provisioning into the network:** Any device on deployment needs to be configured to connect to the local network. The Calypso offers a provisioning mode which allows the

user to connect via a smart phone/tablet and configure the device via web pages running on-board.

- **Easy software integration:** Calypso comes with an intuitive AT-style command interface over UART. This allows the host microcontroller to send ASCII based commands to the module to initiate the necessary actions. Additionally, a fully featured TCP/IP stack with several network applications implemented allows the host controller to delegate the complete network handling to the Calypso module.
- **Low-power operation:** Often, the devices deployed in IoT applications are battery operated. Calypso's low power mode allows the module to consume very little current (< 10 μA) when the device is not in use. However, features like fast connect and auto connect ensure that the module is up and running in a very short time after wake-up.
- **Security:** One of the major challenges in designing any IoT application is security. Calypso deals with this challenge at several levels. The module itself has a secure-bootloader implemented to detect firmware tampering. Wi-Fi compliance ensures conformity to standard Wi-Fi security feature like WPA2 and WPA2 enterprise. Further, at the transport layer, SSL/TLS allow authentication as well as end to end encryption of data. Any modern application requires secure storage to store sensitive information like credentials, certificate-key pairs etc. Calypso also has an encrypted file system with limited space to enable secure storage. Thus, Calypso provides a very good basis for secure IoT application development.

1.5 IoT application demo

The rest of this application note is intended to demonstrate the capabilities of the Calypso Wi-Fi module in an IoT application example. Chapter 2 gives a brief introduction to the MQTT protocol and Chapter 3 provides a step-by-step demo of data transfer between a MQTT client and a MQTT broker using the Calypso evaluation board.

2 MQTT

MQTT - MQ Telemetry Transport is a light weight application layer protocol based on a publish/subscribe messaging mechanism. This protocol was designed for resource constrained and unreliable networks with limited bandwidth and high latency. These characteristics make MQTT suitable for low-power, low-bandwidth IoT applications. Inherently, the MQTT protocol offers some degree of assurance of delivery thereby offering the robustness necessary for industrial machine-to-machine communication.

MQTT was originally developed by IBM and the version 3.1.1 is an OASIS standard that is open and royalty-free [4]. It is based on a client-server architecture where every client connects to a server (broker) over TCP resulting in a star topology. Once connected, the clients send and receive messages using the publish/subscribe mechanism.

2.1 Publish/Subscribe

Data transfer in MQTT takes place based on a publish/subscribe mechanism. The clients connected to the broker can publish messages under certain topics. The clients can also subscribe to topics that are of interest to them. When a client publishes a message on a topic, the broker forwards the message to any client that has subscribed to the topic. This mechanism enables bi-directional communication with an extremely low overhead (2-byte header).

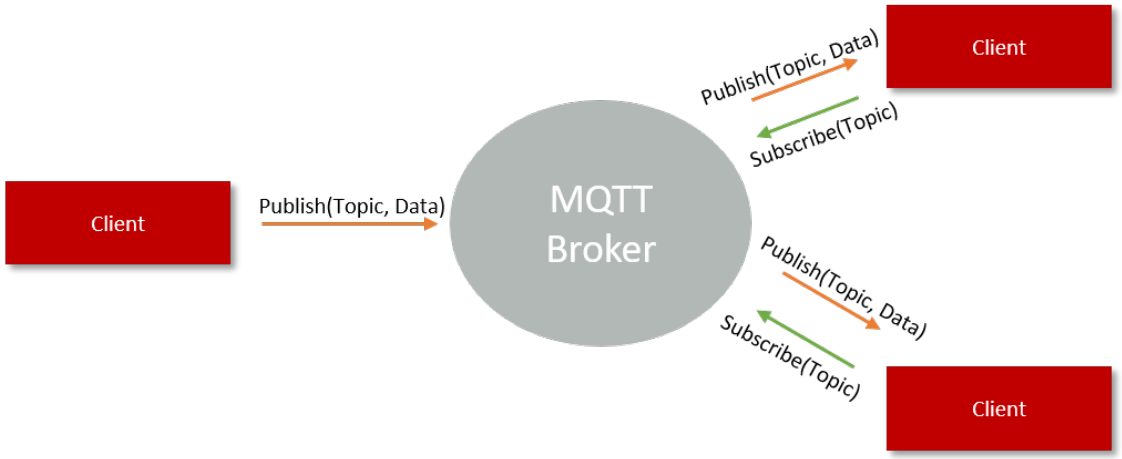

Figure 5: Publish/Subscribe mechanism

2.2 Topics/Subscriptions

Messages in MQTT are always published on topics. A hierarchy can be created in topics using the '/' character. For example, the status of a smart light in the living room can have a topic "home/lighting/living_room/light_index".

Clients can create subscriptions on a topic by explicitly mentioning the topics or by using wildcard characters. There are two wildcards available, '+' and '#'. '+' matches any topics on a single hierarchical level whereas '#' matches any number of levels. For example, subscribing

to "home/lighting/+/light_index" would result in getting status change messages of all lights with "light_index" from all rooms of the house. Subscribing to "home/lighting/#", on the other hand, results in messages with all lights (all light indices) from all rooms.

This feature makes MQTT modular and highly scalable. Inserting a new node into an existing network does not require a lot of configuration.

2.3 QoS

Based on the requirement of the application one of the following levels of Quality-of-Service (QoS) can be chosen.

- **QoS level 0:** The broker/client delivers the message only once without acknowledgement of reception. The reliability in this case is same as that of the underlying TCP.
- **QoS level 1:** The broker/client delivers the message at least once. In this case an acknowledgement of the received packet is done. This case however does not handle duplicate packets.
- **QoS level 2:** The broker/client delivers the message exactly once using a four step handshake. This offers higher reliability at the cost of lower throughput.

2.4 Message persistence

The publisher can specify if a message published to a topic has to be retained. If marked as retained, the broker retains the message and sends it to all new subscribers. This acts as the "last known good" mechanism where nodes that come into the network do not have to wait long to get the first message.

2.5 Last Will and Testament

This mechanism enables the client to publish one last message to all subscribed clients when it abruptly disconnects from the network. Clients can send a last will and testament message to the broker at any point. If the broker detects that a client has gone offline without a disconnect message, it sends the LWT message on the specified topic. This mechanism is very helpful to detect node failures due to battery failure or network outages.

2.6 Security

MQTT offers basic authentication where the client has to send a username and password with the connect message. The broker can authenticate the connect and allow or disallow a client connection. However, the user has to run MQTT over TLS/SSL to enable end-to-end encryption and advanced client authentication.

3 IoT demo using Calypso evaluation board

In this chapter, a demonstration of a smart home IoT application is presented using the Calypso evaluation board. The intention here is to go through the process step-by-step so as to enable the user to implement the same without much effort. Instead of using a specific cloud platform, a more generic approach is taken here by using an open source Mosquitto broker. However, the same concept can be extended to work with any cloud based MQTT broker.

3.1 Prerequisites

The following hardware is necessary to go though this demo.

1. One Calypso evaluation board.
2. An IEEE 802.11 b/g/n compatible access point in the 2.4 GHz band.
3. Computer with a serial terminal emulator like Tera Term.
4. Raspberry Pi 3 Model B or Raspberry Pi 3 Model B+ with latest version of raspbian OS installed [5].
5. Internet connectivity for Raspberry Pi to download and install packages.
6. Smart phone with MQTT app installed. MyMQTT [3] and MQTTTool [2] applications on android and iOS platforms respectively are used in this example.

3.2 Demo set-up

The main objective of this demo is to create a quick prototype of the system designed in section 1.4.

- **Host processor:** For simplicity, a PC/laptop running a serial UART emulator acts as the host processor that controls the actions of the Calypso module.
- **Sensors and actuators:** Simulated sensor data is input into the console on the PC. No actual sensors are used here.
- **Wireless connectivity and gateway:** Calypso provides Wi-Fi connectivity as well as MQTT client for information exchange.
- **webpage_MQTT broker:** Mosquitto broker running on the Raspberry Pi.
- **User interface:** The user interaction is demonstrated by using MQTT apps installed on smart phones.

The set-up consists of a Wi-Fi network (simulating the smart home network) with an access point and a DHCP-server. The Raspberry Pi, the Calypso and the smart phone are connected to this network. The MQTT-broker is running on the Raspberry Pi while MQTT clients run on the smart phone and the Calypso. With this configuration, data exchange can take place between the smart phone (user interface) and Calypso (sensor/actuator) over Raspberry Pi (cloud platform). In the following sections, step-by-step instructions to achieve the same are presented.

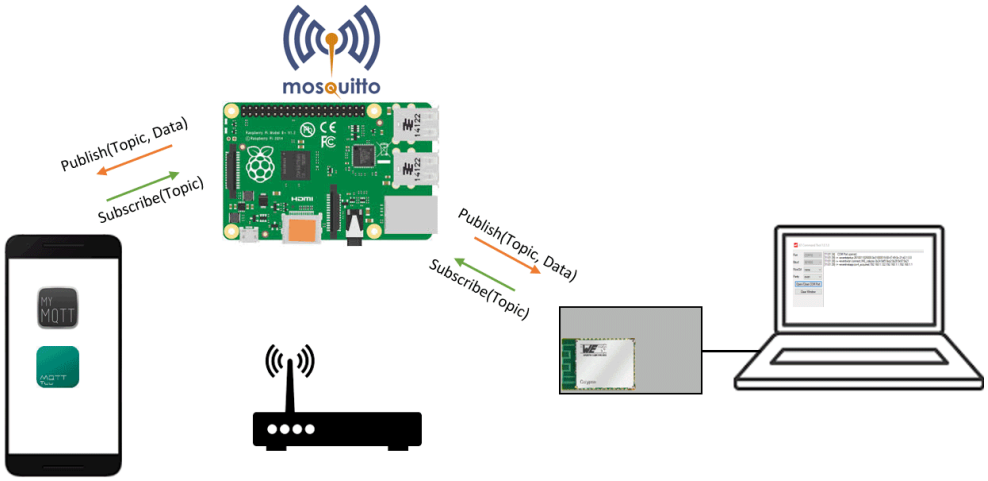

Figure 6: Set up

3.3 Eclipse Mosquitto

Eclipse Mosquitto is an open source light weight message broker that implements MQTT protocol v3.1 and v3.1.1. Mosquitto is a part of the Eclipse foundation. In this particular use case Mosquitto on Raspberry Pi is used as message broker. Mosquitto is available either as executable and/or source on most of the commonly used platforms [1].

3.4 Installing Eclipse Mosquitto MQTT broker on Raspberry Pi

Go through the following steps to install the MQTT broker on the Raspberry Pi.

1. Make sure that the Raspberry Pi is connected to the internet (see [6]).
2. Update the Raspbian OS by typing in the following commands into the terminal on Raspberry Pi.

```
$sudo apt-get update
$sudo apt-get upgrade
```

3. In order to get the latest version of the Mosquitto repository, the key to the repository has to be imported.

```
$sudo wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key
$sudo apt-key add mosquitto-repo.gpg.key
```

4. Make the repository available to apt.

```
$cd /etc/apt/sources.list.d/
```

5. Now get the correct repository based on the corresponding Debian distribution.

```
$sudo wget http://repo.mosquitto.org/debian/mosquitto-stretch.list
```

6. Updating the apt information to access the correct repository.

```
$sudo apt-get update
```

7. Install mosquitto.

```
$sudo apt-get install  mosquitto
```

8. Reboot the system after installation.

3.5 Setting-up Wi-Fi network

An access point compatible with IEEE 802.11b/g/n operating in the 2.4 GHz frequency band is necessary for this application. In this particular example, an access point setup with WPA2, SSID "WE_calypso" and pre-shared key "calypsowlan" is used. The DHCP server on-board the AP is configured to lease IPv4 addresses to the clients. As all network applications are local, no internet connection is necessary.

3.6 Connecting Raspberry Pi to the network

First, the Raspberry Pi is connected to the AP by selecting the correct SSID and entering the pre-shared key [6]. Check the IP address acquired by entering the following command,

```
$ifconfig

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet  192.168.1.126 netmask 255.255.255.0 broadcast 192.168.1.255
inet6  fe80::fe36:49e4:3851:ea47 prefixlen  64   scopeid 0x20<link>
ether  b8:27:eb:ad:ec:72  txqueuelen 1000 (Ethernet)
RX packets 27 bytes 3167 (3.0 KiB)
RX errors 0   dropped 0  overruns 0   frame 0
TX packets 87  bytes 13191 (12.8 KiB)
TX errors 0   dropped 0 overruns 0  carrier  0    collisions   0
```

3.7 Connecting Calypso to the network

3.7.1 Connecting Calypso to PC

1. Connect the Calypso evaluation boards to the laptop/PC via USB.
2. The power LED indicates that supply voltage is active.

The FTDI driver for the converter IC on the evaluation board has to be installed and/or updated to the latest version. On correct driver installation, the evaluation board appears as a virtual COM port.

3. Open an instance of the serial port emulator with COM port settings 921600 Baud, 8e1 for the Wi-Fi module connected to the PC via USB.
4. On pressing the reset button, the start-up message appears on the terminal with the product article number, chipID, MAC address and the current software version. It has to be noted that the MAC address is unique to every module.

```
+eventstartup:2610011025000,0x31000019,c8:fd:19:05:54:b4,1.0.0
```

3.7.2 Connect to an access point

1. In this example an access point with the following settings is used.

   SSID : WE_calypso

   Security method : WPA2_PSK

   Key : calypsowlan

2. Type in the following command into the terminal to connect to the access point.

```
AT+wlanconnect=WE_calypso,,WPA_WPA2,calypsowlan,,,
OK
+eventwlan:connect,WE_calypso,0x24:0xf5:0xa2:0x28:0x97:0x21
+eventnetapp:ipv4_acquired,192.168.1.169,192.168.1.1,192.168.1.1
```

3. The above log indicate a successful Wi-Fi connect and subsequent IP acquisition. The Wi-Fi connection process typically takes a few seconds to complete.

3.8 Connecting smart phone to the network

Make sure that the smart phone/tablet is also connected to the same Wi-Fi network.

In the current example, the addresses assigned are as shown below,

	IP Address	Role
Raspberry Pi	192.168.1.126	MQTT broker
Calypso evaluation board	192.168.1.169	MQTT client
Smart phone	192.168.1.161	MQTT client

Table 1: IP addresses and roles

3.9 Starting MQTT broker on Raspberry Pi

Once mosquitto is installed as described in section 3.4, the broker starts as a background process on boot-up. The following commands can be used to start and stop the background process

```
$sudo /etc/ init .d/mosquitto stop
$sudo /etc/ init .d/mosquitto start
```

The broker can be started in verbose mode to get information regarding MQTT events.

```
$sudo mosquitto -v
```

On starting the Mosquitto broker, a server listens for connections on port 1883.

3.10 Starting MQTT client on Calypso

1. Firstly, an MQTT client is created using the following command. In this case the client ID is chosen to be "kitchen_temp".

   ```
   AT+mqttcreate=kitchen_temp,ip4,192.168.1.126,1883,,,,,,,v3_1,0,1
   +mqttcreate:0
   OK
   ```

2. The next step is to establish a connection with the broker using

   ```
   AT+mqttconnect=0
   +eventmqtt:operation,connack,0
   OK
   ```

   If the broker is opened in verbose mode, the following logs indicate a successful connection.

   ```
   61548680537: New connection from 192.168.1.169 on port 1883.
   1548680537: New client connected from 192.168.1.169 as kitchen_temp (c1, k25).
   1548680537: Sending CONNACK to kitchen_temp (0, 0)
   1548680562: Received PINGREQ from kitchen_temp
   1548680562: Sending PINGRESP to kitchen_temp
   ```

3.11 Starting MQTT client on smart phone

1. Open the smart phone application and enter the broker address and port.
2. Initiate a connection to the broker.

   If the broker is opened in verbose mode, the following logs indicate a successful connection.

```
1548680731: New client connected from 192.168.1.161 as root.1548680731721 (c1
     , k60).
1548680731: Sending CONNACK to root.1548680731721 (0, 0)
```

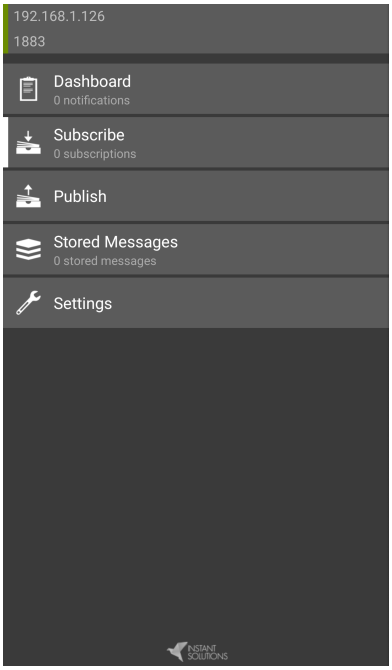

Figure 7: Connect on android device

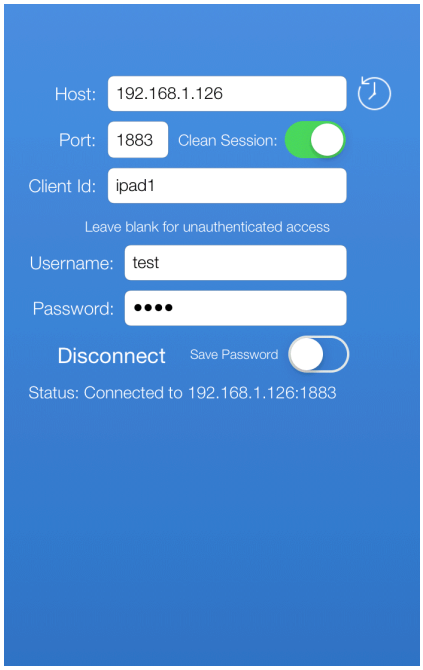

Figure 8: Connect on iOS device

3.12 Data exchange

The final step in the process is to exchange data between the two clients using the previously explained publish/subscribe mechanism. In order to simulate a smart home application, the Calypso sends simulated temperature data under the topic "home/kitchen/temperature". The smart phone on the other hand controls a simulated cooling system by publishing under the topic "home/kitchen/cooler". Both modules have to subscribe to the other topic to receive messages.

3.12.1 Subscribe from Calypso evaluation board

The Calypso evaluation board can be made to subscribe to a topic by sending the following command:

```
AT+mqttsubscribe=0,1,home/kitchen/cooler,QOS0,
OK
```

If the broker is opened in verbose mode, the following logs indicate a successful subscribe:

```
1548682694: Received SUBSCRIBE from kitchen_temp
1548682694:         home/kitchen/cooler (QoS 0)
1548682694: kitchen_temp 0 home/kitchen/cooler
1548682694: Sending SUBACK to kitchen_temp
```

3.12.2 Subscribe from smart phone

Subscribe to the topic "home/kitchen/temperature" from the smart phone app.
If the broker is opened in verbose mode, the following logs indicate a successful subscribe:

```
1548682694: Received SUBSCRIBE from kitchen_temp
1548682694:         home/kitchen/cooler (QoS 0)
1548682694: kitchen_temp 0 home/kitchen/cooler
1548682694: Sending SUBACK to kitchen_temp
```

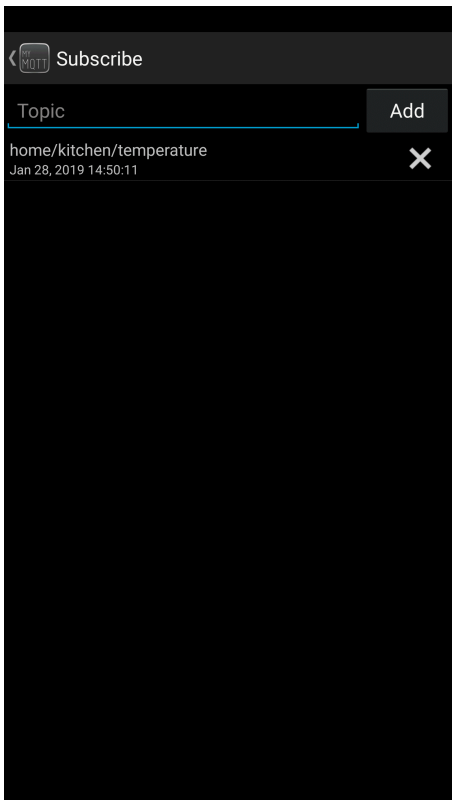

Figure 9: Subscribe on android device

3.12.3 Publish from Calypso evaluation board

The Calypso evaluation board can be made to publish the string "temp=15deg" ("dGVtcD0xNWRlZw==" base64 encoded) by sending the command

```
AT+mqttpublish=0,home/kitchen/temperature,QOS0,0,16,dGVtcD0xNWRlZw==
OK
```

If the broker is opened in verbose mode, the following logs indicate a successful publish:

```
1548683799: Received PUBLISH from kitchen_temp (d0, q0, r0, m0, 'home/kitchen/
    temperature', ... (10 bytes))
1548683799: Sending PUBLISH to root.1548683772471 (d0, q0, r0, m0, 'home/kitchen/
    temperature', ... (10 bytes))
```

The smartphone having subscribed to the topic receives a notification of the same.

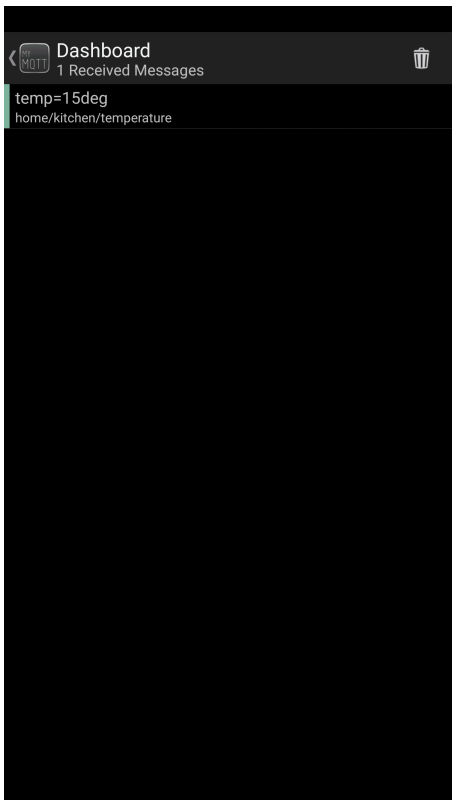

Figure 10: Receive on android device

3.12.4 Publish from smart phone

On initiating a publish from the smart phone on the topic "home/kitchen/cooler" with data "switch_off", the following message appears on the Calypso.

```
+eventmqtt:recv,home/kitchen/cooler,qos0,0,0,1,16,c3dpdGNoX29mZg==
```

Here the publish is notified as an event in the command interface of the Calypso.
If the broker is opened in verbose mode, the following logs indicate a successful publish:

```
1548683524: Received PUBLISH from root.1548683393664 (d0, q0, r0, m0, 'home/
    kitchen/cooler', ... (10 bytes))
1548683524: Sending PUBLISH to kitchen_temp (d0, q0, r0, m0, 'home/kitchen/cooler',
    ... (10 bytes))
```

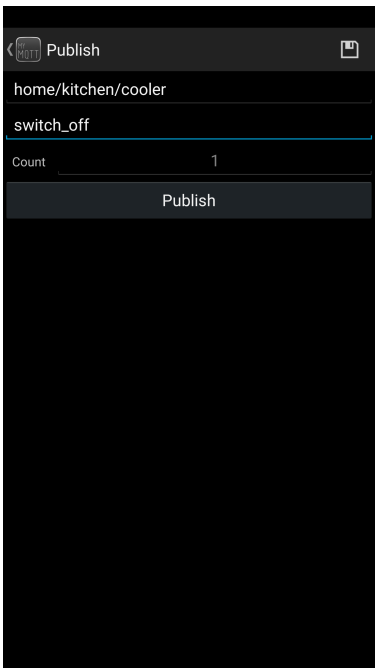

Figure 11: Publish from android device

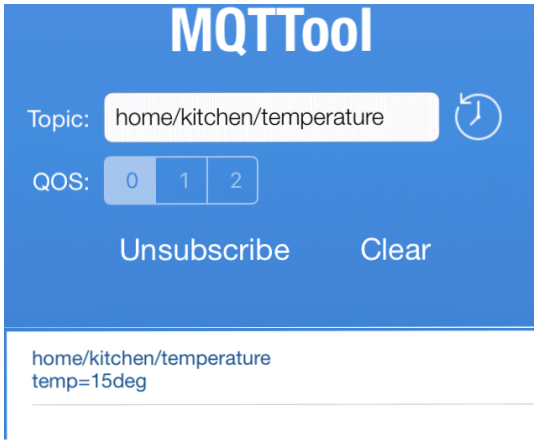

Figure 12: Publish/Subscribe iOS device

3.13 Further enhancements

This example is intended to build a quick prototype to demonstrate the system design. However, it can be easily extended to include more advanced features. Here is a list of possible enhancements.

- Use user authentication at the MQTT broker.
- TLS instead of TCP to enable end-to-end-encryption.
- Implement sleep-modes for battery operated applications.

4 Summary

Designing an IoT solution brings with it a number of challenges. Being multifaceted, IoT applications demand a lot of competence from hardware design to UI development. Under such circumstances, it is prudent to take a modular approach. This means breaking down the architecture into functionally independent blocks. One such essential building-block for an IoT application is wireless connectivity. Integrating a radio module such as Calypso enables the designer to completely delegate the task of wireless connectivity which saves a lot of effort enabling quicker time-to-market.
In this application note, a simple IoT technology stack that represents the principle behind any IoT solution is presented. Further, with the help of a smart home example, a solution using the Calypso Wi-Fi module is described. Finally, a step-by-step demo of the proposed solution is presented. This application note illustrates the ease of integration of the Calypso into any embedded application.

5 References

[1] Eclipse Mosquitto. https://mosquitto.org/download/.

[2] MQTTTool iOS application. https://itunes.apple.com/us/app/mqttool/id1085976398?mt=8.

[3] MyMQTT Android application. https://play.google.com/store/apps/details?id=at.tripwire.mqtt.client.

[4] MQTT specification (V3.1.1). https://mqtt.org/mqtt-specification/.

[5] Raspbian download page. https://www.raspberrypi.org/.

[6] Wireless connectivity in Raspberry Pi Desktop. https://www.raspberrypi.org/documentation/configuration/wireless/desktop.md.

6 Important notes

The Application Note and its containing information ("Information") is based on Würth Elektronik eiSos GmbH & Co. KG and its subsidiaries and affiliates ("WE eiSos") knowledge and experience of typical requirements concerning these areas. It serves as general guidance and shall not be construed as a commitment for the suitability for customer applications by WE eiSos. While WE eiSos has used reasonable efforts to ensure the accuracy of the Information, WE eiSos does not guarantee that the Information is error-free, nor makes any other representation, warranty or guarantee that the Information is completely accurate or up-to-date. The Information is subject to change without notice. To the extent permitted by law, the Information shall not be reproduced or copied without WE eiSos' prior written permission. In any case, the Information, in full or in parts, may not be altered, falsified or distorted nor be used for any unauthorized purpose.

WE eiSos is not liable for application assistance of any kind. Customer may use WE eiSos' assistance and product recommendations for customer's applications and design. No oral or written Information given by WE eiSos or its distributors, agents or employees will operate to create any warranty or guarantee or vary any official documentation of the product e.g. data sheets and user manuals towards customer and customer shall not rely on any provided Information. THE INFORMATION IS PROVIDED "AS IS". CUSTOMER ACKNOWLEDGES THAT WE EISOS MAKES NO REPRESENTATIONS AND WARRANTIES OF ANY KIND RELATED TO, BUT NOT LIMITED TO THE NON-INFRINGEMENT OF THIRD PARTIES' INTELLECTUAL PROPERTY RIGHTS OR THE MERCHANTABILITY OR FITNESS FOR A PURPOSE OR USAGE. WE EISOS DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT RELATING TO ANY COMBINATION, MACHINE, OR PROCESS IN WHICH WE EISOS INFORMATION IS USED. INFORMATION PUBLISHED BY WE EISOS REGARDING THIRD-PARTY PRODUCTS OR SERVICES DOES NOT CONSTITUTE A LICENSE FROM WE eiSos TO USE SUCH PRODUCTS OR SERVICES OR A WARRANTY OR ENDORSEMENT THEREOF.

The responsibility for the applicability and use of WE eiSos' components in a particular customer design is always solely within the authority of the customer. Due to this fact it is up to the customer to evaluate and investigate, where appropriate, and decide whether the device with the specific characteristics described in the specification is valid and suitable for the respective customer application or not. The technical specifications are stated in the current data sheet and user manual of the component. Therefore the customers shall use the data sheets and user manuals and are cautioned to verify that they are current. The data sheets and user manuals can be downloaded at *www.we-online.com*. Customers shall strictly observe any product-specific notes, cautions and warnings. WE eiSos reserves the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time without notice.

WE eiSos will in no case be liable for customer's use, or the results of the use, of the components or any accompanying written materials. IT IS CUSTOMER'S RESPONSIBILITY TO VERIFY THE RESULTS OF THE USE OF THIS INFORMATION IN IT'S OWN PARTICULAR ENGINEERING AND PRODUCT ENVIRONMENT AND CUSTOMER ASSUMES THE ENTIRE RISK OF DOING SO OR FAILING TO DO SO. IN NO CASE WILL WE EISOS BE LIABLE FOR

CUSTOMER'S USE, OR THE RESULTS OF IT'S USE OF THE COMPONENTS OR ANY ACCOMPANYING WRITTEN MATERIAL IF CUSTOMER TRANSLATES, ALTERS, ARRANGES, TRANSFORMS, OR OTHERWISE MODIFIES THE INFORMATION IN ANY WAY, SHAPE OR FORM.

If customer determines that the components are valid and suitable for a particular design and wants to order the corresponding components, customer acknowledges to minimize the risk of loss and harm to individuals and bears the risk for failure leading to personal injury or death due to customers usage of the components. The components have been designed and developed for usage in general electronic equipment only. The components are not authorized for use in equipment where a higher safety standard and reliability standard is especially required or where a failure of the components is reasonably expected to cause severe personal injury or death, unless WE eiSos and customer have executed an agreement specifically governing such use. Moreover WE eiSos components are neither designed nor intended for use in areas such as military, aerospace, aviation, nuclear control, submarine, transportation, transportation signal, disaster prevention, medical, public information network etc. WE eiSos must be informed about the intent of such usage before the design-in stage. In addition, sufficient reliability evaluation checks for safety must be performed on every component which is used in electrical circuits that require high safety and reliability functions or performance. COSTUMER SHALL INDEMNIFY WE EISOS AGAINST ANY DAMAGES ARISING OUT OF THE USE OF THE COMPONENTS IN SUCH SAFETY-CRITICAL APPLICATIONS.

List of Figures

1 IoT application stack 4
2 Smart home example 6
3 System architecture 7
4 System design 8
5 Publish/Subscribe mechanism 10
6 Set up 13
7 Connect on android device 17
8 Connect on iOS device 17
9 Subscribe on android device 19
10 Receive on android device 20
11 Publish from android device 21
12 Publish/Subscribe iOS device 21

List of Tables

1 IP addresses and roles 15

Contact
Würth Elektronik eiSos GmbH & Co. KG
Division Wireless Connectivity & Sensors

Max-Eyth-Straße 1
74638 Waldenburg
Germany

Tel.: +49 651 99355-0
Fax.: +49 651 99355-69
www.we-online.com/wireless-connectivity