



Nagesh Subrahmanyam

MQTT and Arduino Devices

The number of different types of devices that can benefit from MQTT implementations is growing every day. This paper shows how to use MQTT directly from the Arduino platform.

Original publication: This paper was originally published as a chapter in the IBM® Redbooks® publication, *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*, SG24-8054. The publication is available at the following website:

http://www.redbooks.ibm.com/abstracts/sg248054.html

Introducing Arduino

Arduino is an open source electronics platform that enables you to enhance the capabilities of sensors and actuators. Arduino is built on top of a micro-controller that has a number of digital and analogue pins. These pins allow the board to sense its environment and respond to it. After the wiring is done, the board can be programmed using the Arduino programming language.

The syntax of this programming language is similar to the C programming language. The similarity also extends to use of libraries in C programs. That is, a regularly used piece of code could be hived off as a library that other programmers could include in their subsequent programs.

Additional resource: You can learn more about the Arduino programming language at: http://arduino.cc/en/Reference/HomePage

Simple Arduino circuit

This scenario shows a simple example of using Arduino and is taken from the Arduino development IDE. It provides an illustration of the circuit (Figure 1) and includes the source code (Example 1). The board is an Arduino Uno. In the circuit shown in Figure 1, the LED has its cathode connected to GND and the anode to pin 13.



Figure 1 Illustration of the Arduino circuit

In general, the Arduino code runs such that the setup() is run once and only once, whereas the loop() function runs all the time unless the reset button on the board is pressed. When reset is pressed, the code execution begins again with setup() followed by looping the loop() function. To run the code, it is first compiled in the IDE and then uploaded to the board. The upload causes a reset automatically, and the execution begins as described before.

In Example 1, the LED connected to pin 13 is first designated as an OUTPUT pin. Then, in the loop() function, the LED is turned on using the built-in digitalWrite() function, where HIGH turns the LED ON and LOW turns it OFF. A delay of 1000 milliseconds (or a second) is introduced between these functions to give an impression of a blinking LED.



```
/
Blink
Turns on an LED on for one second, then off for one second, repeatedly.
This example code is in the public domain.
/
void setup() {
    // initialize the digital pin as an output.
    // Pin 13 has an LED connected on most Arduino boards:
    pinMode(13, OUTPUT);
}
void loop() {
    digitalWrite(13, HIGH); // set the LED on
```

```
delay(1000); // wait for a second
digitalWrite(13, LOW); // set the LED off
delay(1000); // wait for a second
```

MQTT support in Arduino

}

The MQTT support in Arduino is provided through a library that is an implementation of the MQTT protocol V3. You can download this library at:

http://knolleary.net/arduino-client-for-mqtt/

This library should be included in the Arduino code and then appropriate functions are called for CONNECT, PUBLISH, SUBSCRIBE, and so forth. In our simple example, the code for CONNECT is added in the setup() function because the connection is made only once. The SUBSCRIBE code is added in the setup() function, and the message that has arrived is handled separately in a callback() function. The PUBLISH code is added in the loop() function.

Arduino and MQTT with a publish scenario

The network support for the Arduino board is provided using a *shield*, such as the Arduino Ethernet shield, which is connected to a modem or a network port using a standard network cable. This shield sits on top of the Arduino Uno board or another board.

For more information about the Arduino Ethernet shield, see:

http://arduino.cc/en/Main/ArduinoEthernetShield

Description of the scenario

This scenario illustrates an Arduino-based device that publishes to a topic using the MQTT protocol. The device is a simple temperature sensor that measures the ambient temperature every two minutes and publishes to a topic on an MQTT server.

Figure 2 on page 4 illustrates the circuit for this scenario.



Figure 2 Publish scenario circuit

Example 2 lists the source code for this scenario.

Example 2 Source code for the publish scenario

```
#include <SPI.h>
#include <PubSubClient.h>
#include <DallasTemperature.h>
#include <OneWire.h>
#include <Ethernet.h>
#include <util.h>
#include <ctype.h>
/
OneWire - http://www.arduino.cc/playground/Learning/OneWire
Dallas Temperature Sensor -
https://code.google.com/p/dallas-temperature-control-library/source/checkout
MQTT - http://knolleary.net/arduino-client-for-mqtt/
/
#define CLIENTID "ArduinoSensor"
#define TOPICNAME "sensor/temperature"
#define POLLINTERVAL 120000
#define DS18S20Pin 2
//Some MAC - DEAD BEEF FEED !
byte mac [] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
//Connect to test.mosquitto.org server
byte server [] = { 85, 119, 83, 194 };
//Our MQTT client
PubSubClient arduinoClient(server, 1883, callback) ;
//Board on time
```

```
unsigned long boardTime = 0 ;
//Sensed temperature
float sensedTemperature = 0.0 ;
//Temperature in character format for publishing
char charTemperature [20]
//Temperature chip i/o
OneWire oneWire(DS18S20Pin); // on digital pin 2
DallasTemperature sensors(&oneWire);
//Handle message from mosquitto and set LEDs
void callback(char topic, byte payload, unsigned int length){
//Do nothing as we are publishing ONLY.
}
void setup(void) {
Serial.begin(9600);
// Start the sensor
sensors.begin();
//Connect to the MQTT server - test.mosquitto.org
beginConnection() ;
}
//Initialise MQTT connection
void beginConnection() {
Serial.begin(9600);
Ethernet.begin(mac) ;
int connRC = arduinoClient.connect(CLIENTID) ;
if (!connRC) {
Serial.println(connRC) ;
Serial.println("Could not connect to MQTT Server");
Serial.println("Please reset the arduino to try again");
delay(100);
exit(-1);
}
else {
Serial.println("Connected to MQTT Server...");
}
}
void loop(void) {
boardTime = millis();
if ((boardTime % POLLINTERVAL) == 0) {
getTemp() ;
dtostrf(sensedTemperature,5,2,charTemperature) ;
arduinoClient.publish(TOPICNAME, charTemperature);
}
void getTemp() {
// Send the command to get temperatures
sensors.requestTemperatures();
delay(100);
sensedTemperature = sensors.getTempCByIndex(0);
delay(150);
ł
```

The description of the source code is as follows:

- 1. Add the PubSubClient.h, OneWire.h, and DallasTemperature.h header files and ensure that the associated .cpp files are available to the IDE.
- The server to be connected to is defined as a byte array where each number represents the octets of an IP address.
- Define the function callback() that receives control when a message for the subscribed topic arrives.
- 4. Define the client arduinoClient using the server IP address in the form of byte array, port number and the callback() function as the arguments.
- 5. Define the name of the client and the topic to be subscribed as constants.
- 6. As part of setup(), initialize the pins and initiate the connection. To initiate the connection, define (randomly) the MAC address for the device as 0xDEADBEEFFEED. Change it to the MAC address as applicable to your installation.
- 7. As part of the setup() function, initialize the temperature sensor.
- 8. In the loop() function, obtain the temperature every 2.0 minutes, and publish it to an MQTT topic.

Running the scenario

To run the scenario, follow these steps:

- 1. Start an instance of the WMQTT utility.
- 2. Enter the IP address and port for the MQTT broker.
- 3. Click Connect.
- 4. Enter a topic for the subscription as sensor/temperature/.
- 5. Turn on the board.
- Click Subscribe on the WMQTT utility.

Temperatures appear in the WMQTT utility.

Arduino and MQTT with a subscribe scenario

The network support for the Arduino board is provided using a *shield*, such as the Arduino Ethernet shield. This shield is connected to a modem or a network port using a standard network cable. This shield sits on top of the Arduino Uno board or another board.

For more information about the Arduino Ethernet shield, see:

http://arduino.cc/en/Main/ArduinoEthernetShield

Description of the scenario

This scenario shows a device that subscribes to a topic using the MQTT protocol. In this example, an application reads in the temperature that is published by the device described in "Arduino and MQTT with a publish scenario" on page 3. This application has logic in it to determine whether the temperature is within predetermined limits. If the temperature is within limits, the message YES,YES is published. If the temperature is not within limits, a message

NO,YES or YES,NO is published, depending on whether the lower limit or higher limit, respectively, was breached.

Figure 3 shows the circuit for this scenario.



Figure 3 Subscribe scenario circuit

In this circuit, three LEDs have anodes connected to pins 7, 8, and 9, respectively. Their cathodes are grounded. The resistors are 220 Ω to avoid damaging the LED. These three resistors approximately replicate an RGB LED behavior. That is, at any point, only one of the LEDs will be glowing to indicate NORMAL, HIGH, or LOW situations.

Example 3 shows the source code for this scenario.

Example 3 Source code

```
#include <Ethernet.h>
#include <EthernetUdp.h>
#include <EthernetServer.h>
#include <EthernetClient.h>
#include <Util.h>
#include <Dns.h>
#include <Dhcp.h>
#include <SPI.h>
#include "PubSubClient.h"
//Pin definition
#define NORMALPIN 7
#define LOWPIN 8
#define HIGHPIN 9
//MQTT definition
#define CLIENTID "ArduinoActuator"
```

```
#define TOPICNAME "sensor/temperature/monitor"
By default Pin 7 will be ON to denote that the temperature is between HIGH and LOW
limits.
When the limits are breached, then Pin 7 goes OFF and either Pin 8 or Pin 9 goes
ON depending on which limit was breached.
Pin 7 Pin 8 Pin 9
NO,NO 0 1 1 Meaningless - this should not happen !
NO,YES 0 1 0 LOW limit breached
YES,NO 0 0 1 HIGH limit breached
YES,YES 1 0 0 NORMAL - No limits are breached.
/
// CHANGE MAC ADDRESS TO THAT APPEARS ON THE ETHERNET SHIELD
byte mac [] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
//
//Connect to test.mosquitto.org server
byte server [] = { 85, 119, 83, 194 };
//Handle message from mosquitto and set LEDs
void callback(char topic, byte payload, unsigned int length) {
int i=0; char buffer [length] ;
for(i=0;i<length;i++) {</pre>
buffer [i] = char((payload + i));
String payLoadData = buffer ;
setPins(payLoadData) ;
ł
//Our MQTT client
PubSubClient arduinoClient(server, 1883, callback) ;
void setup() {
//Set pins to indicate normal
beginPins();
//Connect to the MQTT server - test.mosquitto.org
beginConnection() ;
}
//Initialise LED pins
void beginPins() {
pinMode(NORMALPIN, OUTPUT);
pinMode(LOWPIN, OUTPUT);
pinMode(HIGHPIN, OUTPUT);
digitalWrite(NORMALPIN, HIGH);
digitalWrite(LOWPIN, LOW);
digitalWrite(HIGHPIN, LOW) ;
}
//Initialise MQTT connection
void beginConnection() {
//Serial.begin(9600);
Ethernet.begin(mac) ;
int connRC = arduinoClient.connect(CLIENTID) ;
if (connRC) {
arduinoClient.subscribe(TOPICNAME) ;
} else {
Serial.println(connRC) ;
}
void loop() {
```

```
arduinoClient.loop() ;
}
void setPins(String s) {
1
The code between BEGIN and END turns ON all the LED for a second before turning ON
or OFF the LED
to denote the temperature range. If the temperature source does not fluctuate too
much, then this
piece of code can give an impression that the Arduino is "working".
/
// BEGIN
digitalWrite(NORMALPIN, HIGH);
digitalWrite(LOWPIN, HIGH) ;
digitalWrite(HIGHPIN, HIGH) ;
delay(1000);
// END
s.trim();
int temperatureLimits [] = \{1,1\};
if (s.substring(0,s.indexOf(',')) == "YES") { temperatureLimits [0] = 1; }
if (s.substring(0,s.indexOf(',')) == "NO") { temperatureLimits [0] = 0 ; }
if (s.substring(s.indexOf(',')+1) == "YES") { temperatureLimits [1] = 1 ;}
if (s.substring(s.indexOf(',')+1) == "NO") { temperatureLimits [1] = 0 ;}
int ledPins [] = {NORMALPIN, LOWPIN, HIGHPIN};
int ledStatus [] = {1, 0, 0} ; //Default is NORMAL, so NORMAL pin is ON
for (int i=0; i<2; i++) {
if (temperatureLimits [i] == 0) { ledStatus [0] = 0 ; ledStatus [i+1] = 1 ; }
}
for (int j=0; j<3; j++) {
digitalWrite(ledPins[j],ledStatus[j]
);
}
}
```

The description of the source code is as follows:

- 1. Add the PubSubClient.h header file, and ensure that this header file and the associated .cpp files are available to the IDE.
- 2. Define the server to be connected to as a byte array, where each number represents the octets of an IP address.
- 3. Define the function callback() that receives control when a message for the subscribed topic arrives.
- 4. Define the arduinoClient client using the server IP address in the form of byte array, port number, and the callback() function as the arguments.
- 5. Define the name of the client and the topic to be subscribed as constants.
- 6. As part of the setup() function, initialize the pins and initiate the connection. To initiate the connection, define (randomly) the MAC address for the device as 0xDEADBEEFFEED. Change it to the MAC address that is burnt onto the device.
- 7. After initiating the connection, subscribe to the topic name in which you are interested.

The callback() function calls the setPins() function to turn ON or OFF the LEDs as described previously.

Running the scenario

To run the scenario, follow these steps:

- 1. Start an instance of the WMQTT utility.
- 2. Enter the IP address and port for MQTT broker.
- 3. Click Connect.
- 4. Enter a topic for publication as sensor/temperature/monitor.
- 5. Type in the message YES, YES and click **Publish**.
- 6. Watch the LEDs turn ON or OFF.

To run this scenario, set up an MQTT server and a client. Then, publish to the sensor/temperature/monitor topic the message YES,YES, NO,YES, or another setting.

The author who wrote this paper

This paper was produced by a specialist working at the International Technical Support Organization, Raleigh Center.

Nagesh Subrahmanyam is an IT Specialist in India. He has over 10 years of experience in the field spanning IBM z/OS®, WebSphere® MQ, WebSphere Message Broker and XML technologies. He holds a Bachelor's degree in Mechanical Engineering from National Institute of Technology, Jamshedpur. His areas of expertise include Java and lately the Arduino platform. He has contributed a couple of articles on the IBM developerWorks® website, and co-authored IBM Redbooks publications about XML and z/OS. He has also submitted a WebSphere MQ SupportPac.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Stay connected to IBM Redbooks publications

Find us on Facebook:

http://www.facebook.com/IBMRedbooks

 Follow us on Twitter: http://twitter.com/ibmredbooks Look for us on LinkedIn:

http://www.linkedin.com/groups?home=&gid=2130806

Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

► Stay current on recent Redbooks publications with RSS Feeds:

http://www.redbooks.ibm.com/rss.html

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright International Business Machines Corporation 2012. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This document REDP-4929-00 was created or updated on October 12, 2012.



Send us your comments in one of the following ways:

- Use the online Contact us review Redbooks form found at: ibm.com/redbooks
- Send your comments in an email to: redbooks@us.ibm.com
- Mail your comments to: IBM Corporation, International Technical Support Organization Dept. HYTD Mail Station P099 2455 South Road Poughkeepsie, NY 12601-5400 U.S.A.



Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

WebSphere® z/OS®

developerWorks®	Redbooks®
IBM®	Redbooks (logo) 🧬 🛛

The following terms are trademarks of other companies:

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.