WHITE PAPER

INTERNET OF THINGS AND UEI DATA ACQUISITION & CONTROL SYSTEMS

Protocols available for sharing data between data acquisition systems and other IoT objects and services

United Electronic Industries

www.ueidaq.com

Internet of Things and UEI Data Acquisition & Control Systems

Written by UEI Staff

Internet of Things Overview

The Internet of Things (IoT) is a networked system of interconnected physical objects that have the capability of sharing data with each other and also with a cloud service for archiving and analysis. With IoT, data producing devices can easily publish data, and multiple consumer clients can receive that data from any location in the world.

> Why IoT matters for industrial systems

Statistics for IoT connectivity list the current number of devices connected over the Internet as somewhere between 6 billion and 17.6 billion.[1] The number of connections and interconnections is expected to exponentially increase in the years to come as IoT evolves beyond consumer-driven applications that include fitness and entertainment products to industrial-driven solutions that result in improved efficiencies, customer satisfaction, and profit for corporations.

With the advent of interconnected “things,” devices that have not typically been accessible outside of a local network can become accessible on a world-wide scale. For commercial and industrial data acquisition (DAQ) systems, these devices include sensors, controllers, signal conditioning units, machines for connectivity and processing units. One of the strengths of IoT is that it allows multiple data producers and multiple data consumers to work concurrently. Once devices are interconnected and accessible, the opportunities for remote diagnostics, predictive maintenance systems, automation, and real-time data collection and analysis become broader-based and more globally viable.[2]

Predictive modeling, machine learning, and other big-data analysis tools for global industrial systems require high volumes of data from distributed environments. IoT is being looked upon to solve many data throughput and delivery concerns associated with industrial data acquisition systems.

> UEI and IoT Enablement

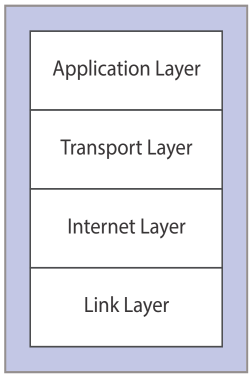
This white paper describes protocols available for sharing data between data acquisition systems and other IoT objects and services. The first sections of the paper discuss IoT connectivity and protocols. The last section provides three IoT tutorials using United Electronic Industries (UEI) DAQ hardware and software.

IoT Connectivity: How it works

IoT works by objects and services connecting and communicating over the Internet. IoT objects must conform to the Internet Protocol Suite to be considered “things” in terms of the Internet of Things.

> IoT Protocol Overview

The Internet Protocol Suite refers to the set of protocols used on the Internet. It models Internet communication and can be described in terms of four stacked protocol layers based on the Open Systems Interconnection (OSI) model.[3]



The lowest layer is the link layer, which defines communication locally between physical devices. The Internet Protocol (IP) layer is next. The IP layer identifies addressing through IP addresses and packet structuring. Example protocols of the IP layer are IPv4 and IPv6.

The next layer up is the transport layer. Protocols at the transport layer ensure host-to-host transmissions occurring over established ports. Example protocols of this layer are the Transport Control Protocol (TCP) and the User Datagram Protocol (UDP), where datagrams are another name for packets. The Internet Protocol Suite is often commonly referred to as TCP/IP, the acronyms for these two lower levels.

At the top of the Internet Protocol Suite stack are application layer protocols, which determine how applications communicate with each other. These protocols can be associated with distributed application structures, such as client-server or publish-subscribe applications.[4]

A popular protocol for industrial IoT (IIoT) applications is Message Queue Telemetry Transport (MQTT), a lightweight publish-subscribe messaging protocol. MQTT is an open source TCP-based protocol first authored in 1999 and designed for applications where hosting hardware may have limited on-board memory and for applications requiring remote communications.

> IoT Protocols Supported in UEI Hardware & Software

UEI hardware and software support several application level protocols that enable IoT.

MQTT UEI's Linux-based programmable automation controllers (UEIPACs) come pre-installed with Eclipse Mosquitto, which implements the MQTT machine-to-machine (M2M) messaging protocol. UEIPAC runs as a standalone data acquisition and control system and supports MQTT applications out-of-the-box.

VxWorks UEIPACs can alternatively run with Wind River's standard VxWorks real-time operating system (RTOS) kernel instead of the Linux kernel. Wind River offers their own IoT-enabled product portfolio, Wind River Helix, for IoT applications.[5]

OPC-UA Additionally, UEI also offers a UEIPAC that is compatible with the Open Platforms Communication (OPC) United Architecture (UA), a platform independent protocol for industrial M2M communication. The UEIOPC-UA controller is a UEIPAC that runs a standard OPC-UA server. In the spring of 2016, OPC Foundation announced that publish/subscribe functionality was being added to OPC-UA. UEIOPC-UA's IoT-enablement will continue to evolve as OPC-UA continues to develop infrastructure to achieve interoperability for IIoT, IoT, and industrie 4.0 applications and devices.[6]

About the UEIPAC

The UEIPAC is a standalone embedded controller.

Depending which chassis type you choose, the UEIPAC can be configured with 1 to 12 I/O boards, with a selection of over 60 I/O boards to choose from.

The UEIPAC uses a Linux or VxWorks kernel. The kernel comes preloaded into flash memory along with drivers for each of the I/O boards.

An SD card houses the root file system, which includes libraries, utilities, initialization scripts, daemons, as well as example code, documentation, and an assortment of third-party software.

The following sections of this white paper provide an MQTT overview and tutorials.

MQTT for IoT-Enabled Industrial Communication

UEI provides built-in MQTT support with the UEIPAC data acquisition and control system. A Mosquitto open-source message broker and C library for publishing and subscribing to MQTT messages is pre-installed on the PAC.

UEIPAC is also pre-loaded with example C code for connecting the UEIPAC to other MQTT clients. C code examples can be used as a basis for user code development. Additionally, the UEIPAC Software Development Kit (SDK) comes with the GNU toolchain compiled to run on a host PC and build binaries targeting the UEIPAC.

> MQTT connects many DAQs to many consumers

Implementations get interesting when the traditional DAQ signal chain used for industrial applications is integrated in the MQTT data flow model. For UEI DAQs, our API allows only one supervisory system to read data from DAQ hardware at a time. This is a serial signal chain from sensor to PC.

One of the strengths of using an MQTT data flow model for IoT is that when a DAQ system publishes data, multiple data consumers can subscribe to and consume the same data.

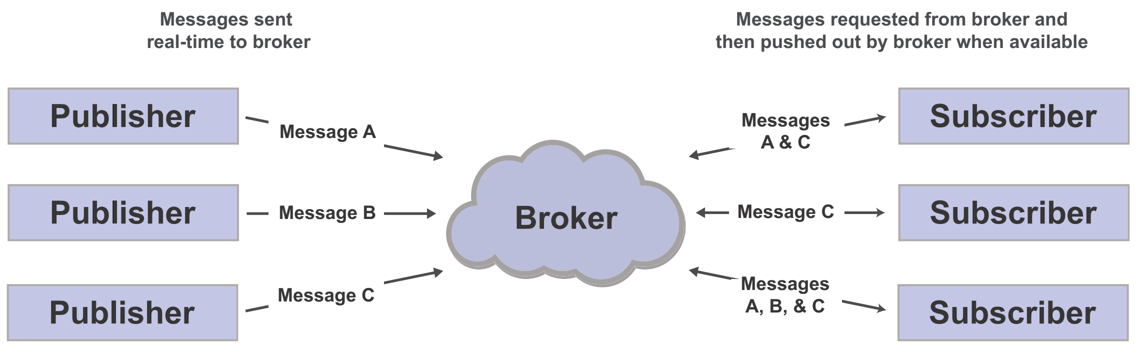


Figure 1: Subscriber/Publisher Data Transfers

MQTT Data Flow Model The MQTT publish/subscribe communication model is event driven. Messages are pushed to clients (as opposed to clients periodically requesting updates).

MQTT works by clients connecting to a broker to publish data messages or subscribe to them. The broker is the central communication point and mediator in charge of dispatching all messages among multiple publishers and multiple subscribers.

Clients publish messages with a topic name embedded in the message, and the broker uses that topic as routing information. Receiving clients register with the broker and subscribe to specific topics. The broker delivers all messages with matching topics to the clients. All data storage and transfer control is done by the broker.

Traditional DAQ Data Flow Model The traditional DAQ data flow model includes sensors and DAQ signal conditioning hardware co-located at the edge of a network. Sensors detect real-world physical conditions, which they convert into electrical representations. DAQ hardware samples the sensor output and produces digitized data through signal conditioning of the sensor samples. Digitized data is then acquired by computers for data collection, processing, and analysis.

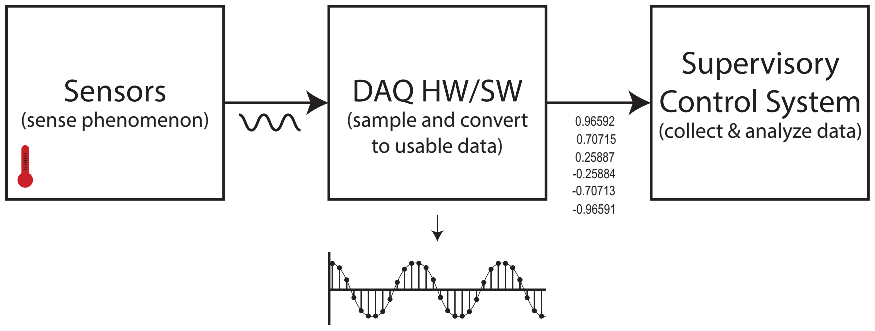


Figure 2: Traditional DAQ Data Flow Model

Sensors are wired to UEI DAQ hardware through 37- or 62-pin D-Sub connectors, and output data can be stored locally on an SD-card or other removable media or transferred to a host computer via a Wifi, copper, or fiber optic Ethernet connection.

IoT Data Flow Model For IoT systems, the traditional DAQ data flow model changes such that DAQ hardware acts as client to a broker instead of transferring data directly to a supervisory system. DAQ hardware still collects raw data from sensors at the point of creation, but through a pre-established authenticated instance with a cloud service, the digitized data is transferred to a broker using the MQTT protocol. Once data is secured by the broker, any authenticated subscriber can access the data, asynchronously and concurrently with other subscribers.

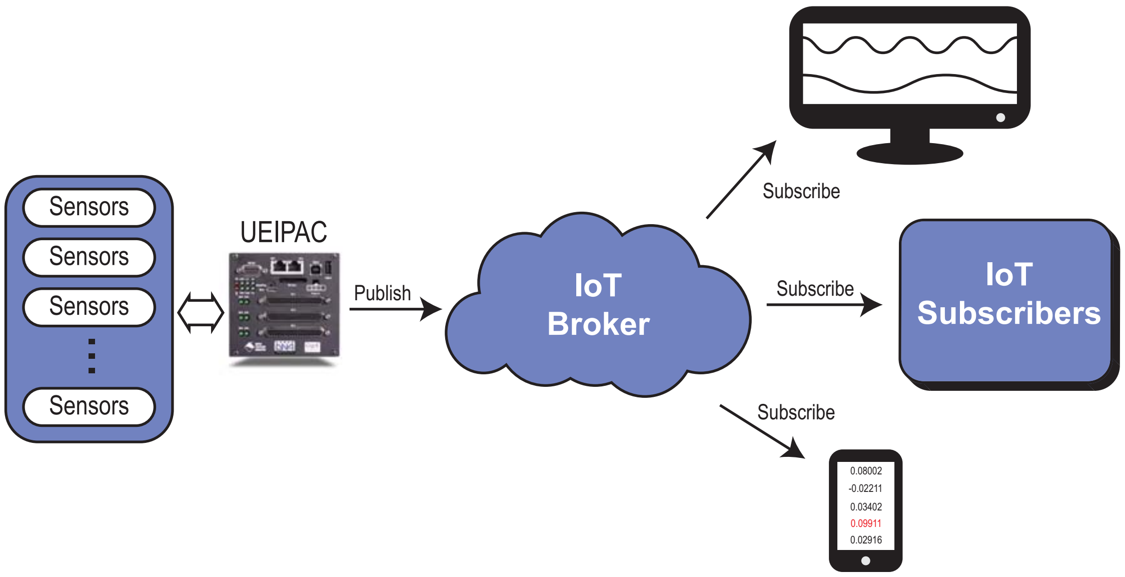


Figure 3: Example of Enabling IoT Implementations with UEIPAC

UEI IoT Connectivity Tutorials

The tutorials in the following sections are based on running UEI example code on a UEIPAC and using the pre-installed Mosquitto sub/pub/broker software. Other configurations are supported; however, all configurations require an MQTT broker, either running on a network accessible host or running the Mosquitto broker on the UEIPAC itself.

This tutorial includes the following information:

- A Quick Overview of Example Code and Formatting (page 6)
- Setting Up UEIPAC for MQTT Publishing (page 8)
- Using the Local Mosquitto MQTT Service on the UEIPAC (page 8)
- Using the Local MQTT Publisher & Broker and Remote MQTT Subscribe Service (page 10)
- Using the Local MQTT Publisher, a Remote Cloud Broker, & Remote MQTT Subscription Applications (page 13)

> A Quick Overview of Example Code and Formatting

Example Code Overview

The sample program, “SampleDMAP_MQTT,” comes pre-loaded on the UEIPAC and is available with the UEIPAC SDK download. Sample code runs directly on the UEIPAC and acquires and publishes data from specified analog input, digital input, or counter/timer DAQ channels. The example can also subscribe to external data and transfer it to analog output and digital output DAQ channels.

Formatting Overview

Text formatting Text formatted in `fixed typeface` generally represents source code or other text that should be entered verbatim at the command line or into the source code, initialization, or other file. These commands should be typed exactly as shown.

Topic formatting MQTT clients publish messages with a topic name embedded in the message, and the broker uses that topic as routing information. One MQTT topic is mapped to each specified UEIPAC channel.

Topic names in this tutorial consist of three topic levels with each level separated by a forward slash: <**Channel type**>/<**Device position**>/<**Channel number**>.

- **Channel type** is the type of I/O board being referenced. For example, AI is an analog input board, DI is a digital input board, and CT is a counter board.
- **Device position** is the board position in the chassis. For example, Dev0 is the first or top DAQ I/O board installed in the UEIPAC chassis, Dev1 is the second board, Dev2 is the third, etc.
- **Channel number** is the channel number of the board. For example, the DIO-449 board is a 48 channel digital input board. The topic name can reference any channel from “Ch0” to “Ch47” to specify the input.
- Wild cards are also permitted in the topic name: # is the multi-level wildcard, and + is the single level wildcard.

The following are examples of topic names for UEIPAC devices:

- `AI/Dev0/Ch0`: Channel 0 of the analog input board in the first chassis position
- `AI/Dev1/Ch0`: Channel 0 of the analog input board in the second chassis position
- `DI/Dev2/Ch1`: Channel 1 of the digital input board in the third chassis position
- `AI/#`: All channels of all analog input boards in any chassis position

Sample code parameter formatting The SampleDMAP_MQTT example code collects input samples from the specified channels and publishes those samples to the specified host/broker.

The following are supported command line parameters:

Parameter	Description
`-d <device list>`	The list of devices (or board positions) to include in the data acquisition (DMAP) (e.g., `-d 0,1` will include I/O boards in the chassis at positions 0 and 1)
`-c <channel per device>`	The list of channels to configure for each device (e.g., `-c 4,8` configures 4 channels on device 0 (channels 0 thru 3), and 8 channels on device 1 (channels 0 thru 7)
`-f <refresh rate>`	The rate at which data is published (in hertz)
`-h <MQTT host>`	The IP address of the MQTT broker. If using DHCP, this can be the hostname
`-p <MQTT port>`	The IP port used to connect to the MQTT broker
`-u <username>`	The user name for brokers that require authentication
`-P <password>`	The password for brokers that require authentication

> Setting Up UEIPAC for MQTT Publishing

1. Access the Linux command line prompt on the UEIPAC

To access the Linux command line prompt, install a serial cable between your host PC and the UEIPAC to connect to the PAC file system via the serial port (57600/8/1/no parity).

Alternatively, you can remote shell (`ssh`, `telnet`, etc.) into the UEIPAC from a PC running Linux OS, from a PuTTY telnet or SSH session from a Windows PC, or from a Cygwin shell.

The default IP address on NIC 1 of the UEIPAC is 192.168.100.2.

2. Add “mosquitto” as a user

Before using the Mosquitto service, “mosquitto” must be added as a user.

Type the following at the UEIPAC Linux prompt (~#), and then enter and reenter the password (the default password is `powerdna`):

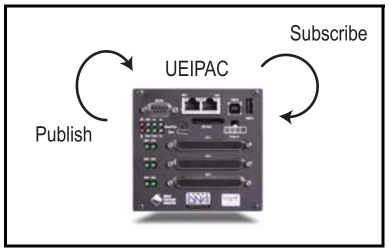
- `~# adduser mosquitto`

3. Start the Mosquitto service on the UEIPAC

Type the following command at the UEIPAC Linux prompt to start the Mosquitto service:

- `~# mosquitto –c /etc/mosquitto/mosquitto.conf &`

> Using the Local Mosquitto MQTT Service on the UEIPAC



The following tutorial is an introduction to using the MQTT protocol on the UEIPAC. This example uses the local Mosquitto MQTT service that is pre-installed on the UEIPAC.

The tutorial starts the sample code running on the UEIPAC, which acquires and publishes data from an analog input board. Then the local MQTT command line subscriber is used to request the analog input data from the on-board Mosquitto broker.

If the Mosquitto service hasn’t already been started, follow the steps in “Setting Up UEIPAC for MQTT Publishing” above.

1. Start the sample code on the UEIPAC

Type the following command at the UEIPAC Linux prompt:

- `~# /usr/local/examples/SampleDMAP_MQTT –d 0,1 –c 4,8 –f 10 –h 127.0.0.1 –p 1883`

This command starts the sample running on the UEIPAC. The publisher publishes analog input samples acquired from the first 4 channels on board 0 (Dev0) and the first 8 channels on board 1 (Dev1) to the broker local on the UEIPAC at a frequency of 10 Hz.

In this example, the host IP address of the broker is the loopback IP address (`127.0.0.1`), and the port is 1883.

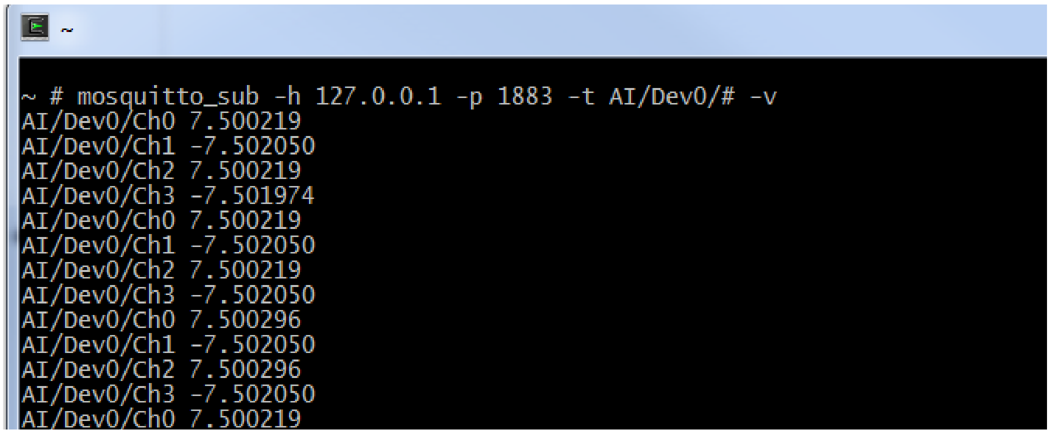
2. **Verify MQTT connectivity with the Mosquitto subscriber on the UEIPAC**

   Open a second Linux terminal window to run `mosquitto_sub` while the sample code is still running in the first Linux terminal window to verify that MQTT messages are published.

   NOTE The `mosquitto_sub` command invokes a simple MQTT client that will subscribe to a single topic and print all messages it receives.

   Type the following command at the UEIPAC Linux prompt to see subscribed data messages:

   - `~# mosquitto_sub -h 127.0.0.1 -p 1883 -t AI/Dev0/# -v`



```
~ # mosquitto_sub -h 127.0.0.1 -p 1883 -t AI/Dev0/# -v
AI/Dev0/Ch0 7.500219
AI/Dev0/Ch1 -7.502050
AI/Dev0/Ch2 7.500219
AI/Dev0/Ch3 -7.501974
AI/Dev0/Ch0 7.500219
AI/Dev0/Ch1 -7.502050
AI/Dev0/Ch2 7.500219
AI/Dev0/Ch3 -7.502050
AI/Dev0/Ch0 7.500296
AI/Dev0/Ch1 -7.502050
AI/Dev0/Ch2 7.500296
AI/Dev0/Ch3 -7.502050
AI/Dev0/Ch0 7.500219
```

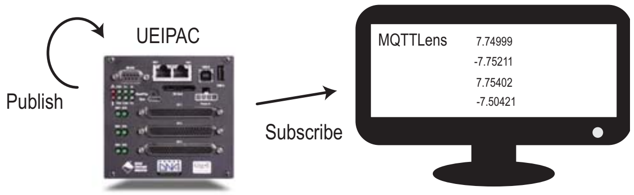
Figure 4: Analog Input Values Pushed to Local Subscriber from Local Broker

The subscriber requests `AI/Dev0/#` topics. The `-t` option specifies which topics are requested, and `-v` specifies verbose printing.

In this example, the `mosquitto_sub` service subscribes to receive all channels from the first analog input board in the UEIPAC chassis. The analog input board in this example has even channels tied to +7.5 volts and odd channels tied to -7.5 volts.

> Using the Local MQTT Publisher & Broker and Remote MQTT Subscribe Service

The following tutorial uses a remote MQTT subscription service to request and view topics from the local Mosquitto broker on the UEIPAC.



This tutorial sets up running sample code on the UEIPAC the same way as in the previous example; however, the MQTT subscriber is provided by the Chrome application, MQTTLens.

If the Mosquitto service hasn't already been started, follow the steps in "Setting Up UEIPAC for MQTT Publishing" on page 8.

1. **Start the sample code on the UEIPAC**

   Type the following command at the UEIPAC Linux prompt:

   - ~# /usr/local/examples/SampleDMAP_MQTT -d 0,1 -c 4,8 -f 10 -h 127.0.0.1 -p 1883

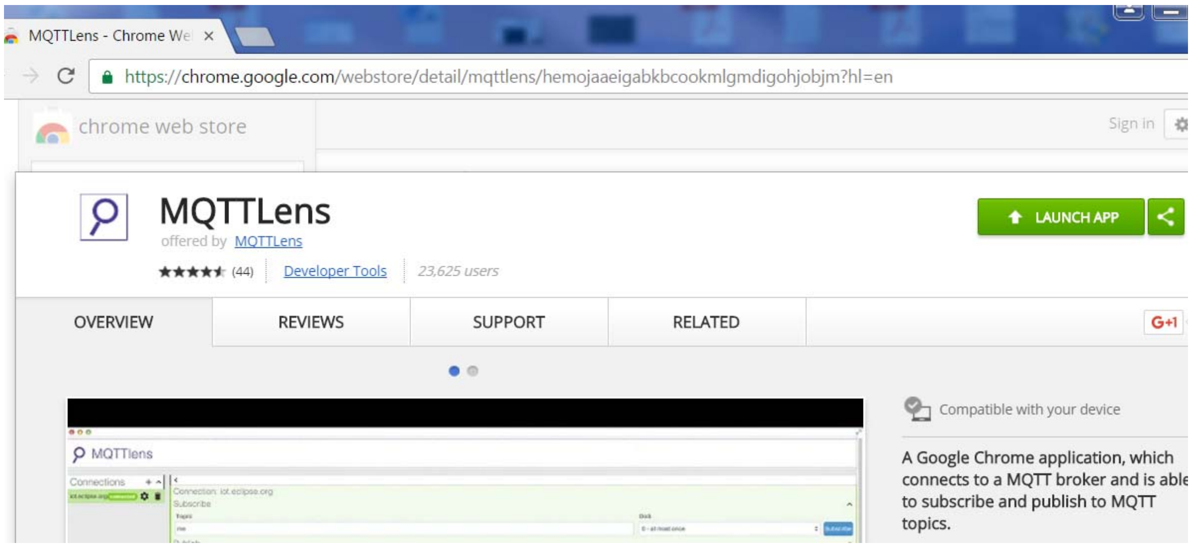
   As in the previous example, the sample is running on the UEIPAC, and the publisher is publishing analog input samples acquired from the first 4 channels on board 0 (Dev0) and the first 8 channels on board 1 (Dev1) to the Mosquitto broker local on the UEIPAC at a frequency of 10 Hz.

   The host IP address of the broker is the loopback IP address (`127.0.0.1`), and the port is 1883.

2. **Install an MQTT client on your PC**

   This tutorial uses MQTTLens, a free Google Chrome App. MQTTLens connects to any MQTT broker and can subscribe to MQTT topics.

   Launch MQTTLens:



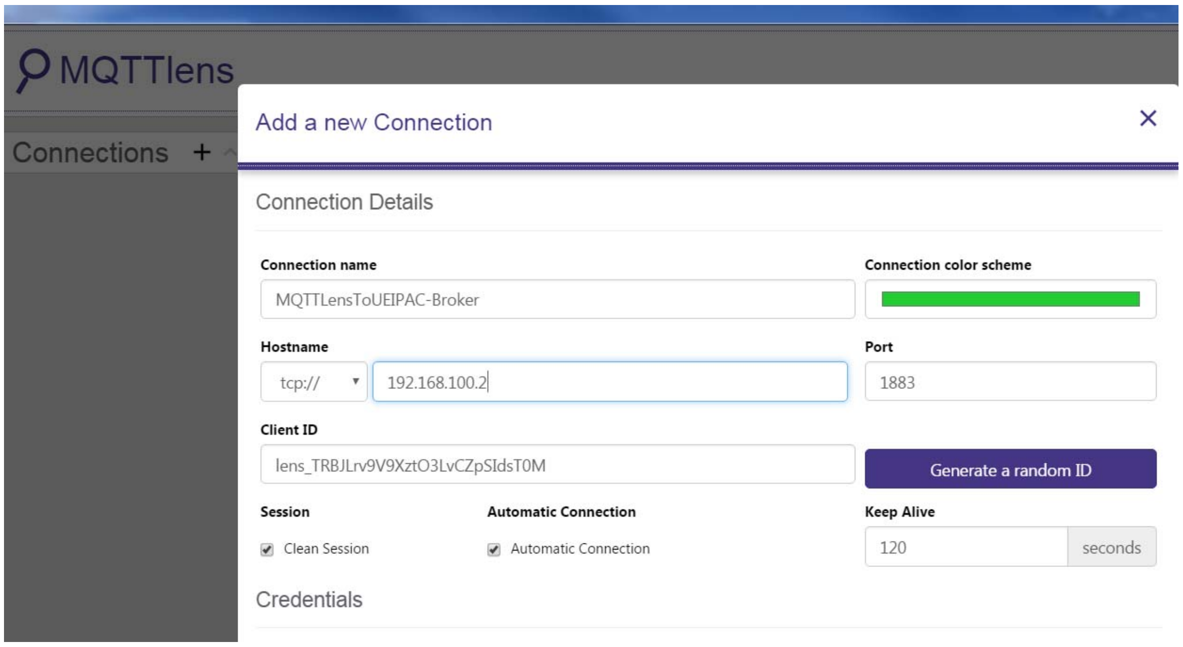
A Google Chrome application, which connects to a MQTT broker and is able to subscribe and publish to MQTT topics.

Figure 5: Launch MQTTLens Subscription Client

3. **Identify the broker connection for MQTT subscription client**

   In this example, the broker is the local Mosquitto broker on the UEIPAC. The broker IP address is the default IP address of the UEIPAC, which is the 198.168.100.2, and the port is 1883. Note that port 1883 is an assigned MQTT TCP port.

   Click **Connections +** to add a connection, and then enter a connection name in the **Connection name** field and the UEIPAC IP address under **Hostname**:



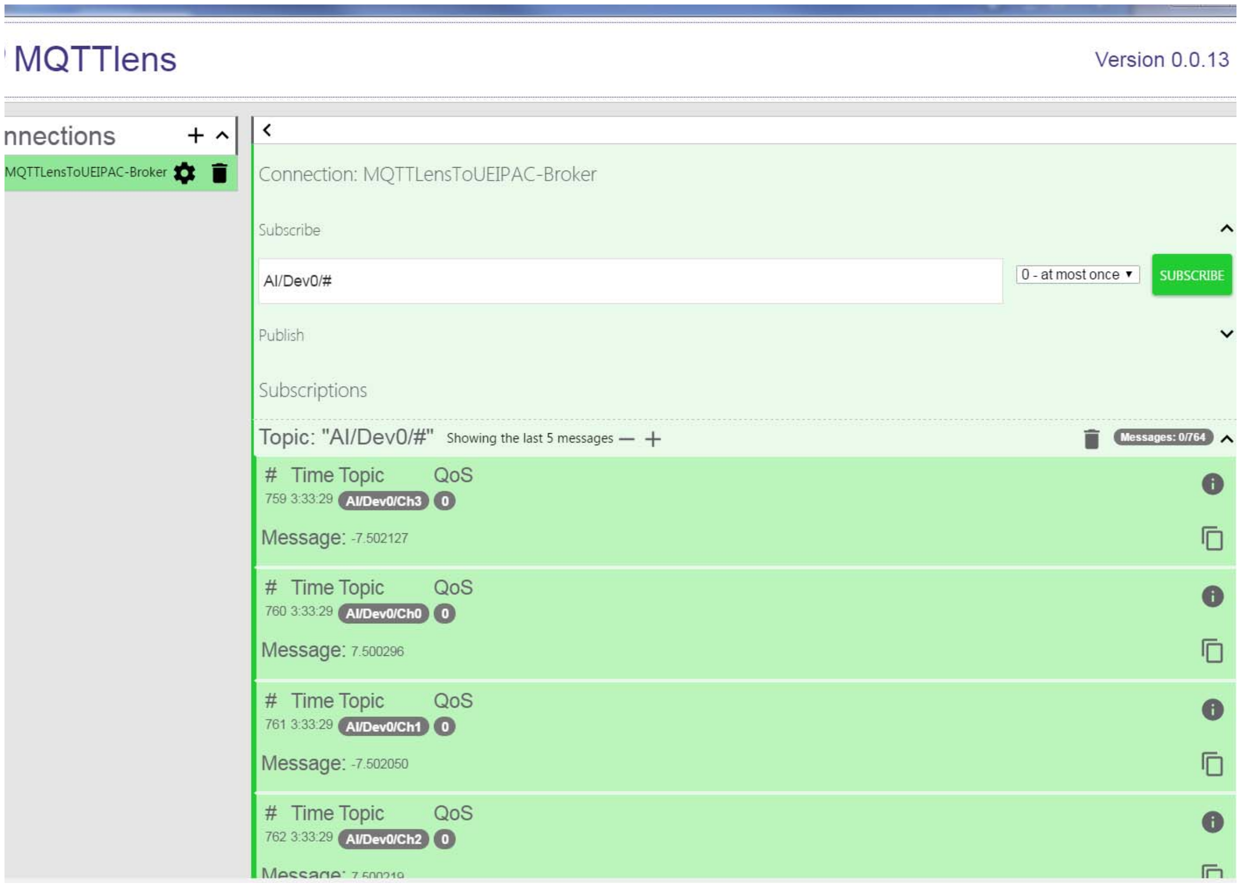
Connection name: MQTTLensToUEIPAC-Broker
Connection color scheme: 🟩
Hostname: tcp:// 192.168.100.2
Port: 1883
Client ID: lens_TRBJLrv9V9XztO3LvCZpSIdsT0M
Generate a random ID
Session: ✓ Clean Session
Automatic Connection: ✓ Automatic Connection
Keep Alive: 120 seconds

Figure 6: Enter Broker Connection Information for MQTTLens Subscription Client

Click **Create Connection** to save changes.

4. **Subscribe to topics using the MQTT subscription client**

   Enter `AI/Dev0/#` in the **Subscribe** field to acquire data messages from all channels on the analog input board at device 0, and then click **SUBSCRIBE**.



AI/Dev0/#

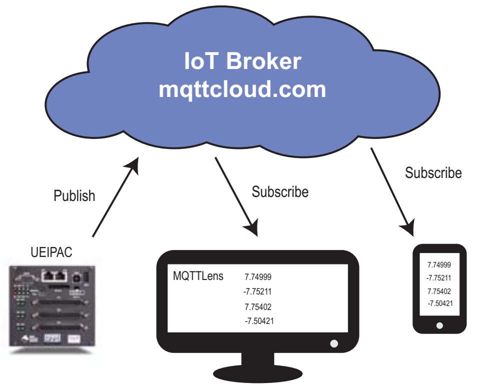
Topic: "AI/Dev0/#"

```
# Time Topic QoS
759 3:33:29 AI/Dev0/Ch3 0
Message: -7.502127

# Time Topic QoS
760 3:33:29 AI/Dev0/Ch0 0
Message: 7.500296

# Time Topic QoS
761 3:33:29 AI/Dev0/Ch1 0
Message: -7.502050

# Time Topic QoS
762 3:33:29 AI/Dev0/Ch2 0
Message: 7.500219
```

Figure 7: MQTTLens Displays Subscription Topics

Topics print in the GUI display.

> Using the Local MQTT Publisher, a Remote Cloud Broker, & Remote MQTT Subscription Applications



The following tutorial uses the local Mosquitto publisher on the UEIPAC, a remote MQTT broker cloud service, and remote MQTT subscription services (a Chrome app and a hand-held device app).

Several companies offer hosted MQTT services as part of their cloud offerings, and a variety of MQTT client apps exist. This white paper uses cloudmqtt.com because it offers a free plan (with limited bandwidth) and uses MQTTLens and ICPDAS MQTT because they are free applications available for the Chrome browser and iPhone, respectively.

This example goes through the steps of creating an account on cloudmqtt.com, creating an MQTT instance, and setting up MQTTLens and the hand-held MQTT app to simultaneously view samples on both.

This example also provides the steps to set up the UEIPAC to use a DHCP assigned IP address. Using DHCP will configure the gateway and DNS properly, so the “cloudmqtt.com” hostname can be entered instead of a semi-static IP address. The dynamically allocated IP address for the UEIPAC will be needed when we setup the MQTT instance.

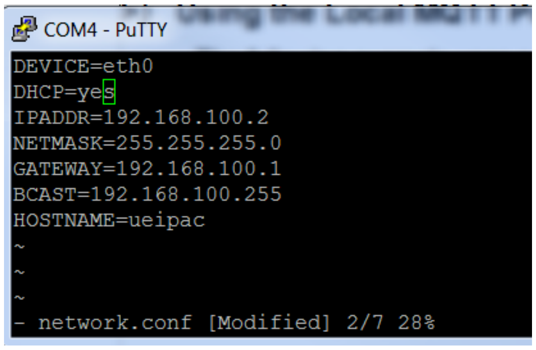
If the Mosquitto service hasn’t already been started, follow the steps in “Setting Up UEIPAC for MQTT Publishing” on page 8.

1. Setup the UEIPAC to use a dynamically hosted IP address with DHCP

Connect to the UEIPAC over the serial port (57600/8/1/no parity).

Edit the `/etc/network.conf` file, and change “`DHCP=no`” to “`DHCP=yes`”.

Restart the UEIPAC.



```
DEVICE=eth0
DHCP=yes
IPADDR=192.168.100.2
NETMASK=255.255.255.0
GATEWAY=192.168.100.1
BCAST=192.168.100.255
HOSTNAME=ueipac
~
~
~
- network.conf [Modified] 2/7 28%
```

In the serial terminal window, type `ifconfig` to see the dynamically assigned IP address (the assigned DHCP address shows as 192.168.0.91 in the following):

```
~ # ifconfig
      eth0
         Link encap:Ethernet  HWaddr 00:0C:94:00:D0:D0
         inet addr:192.168.0.91 Bcast:192.168.0.255
               Mask:255.255.255.0
```

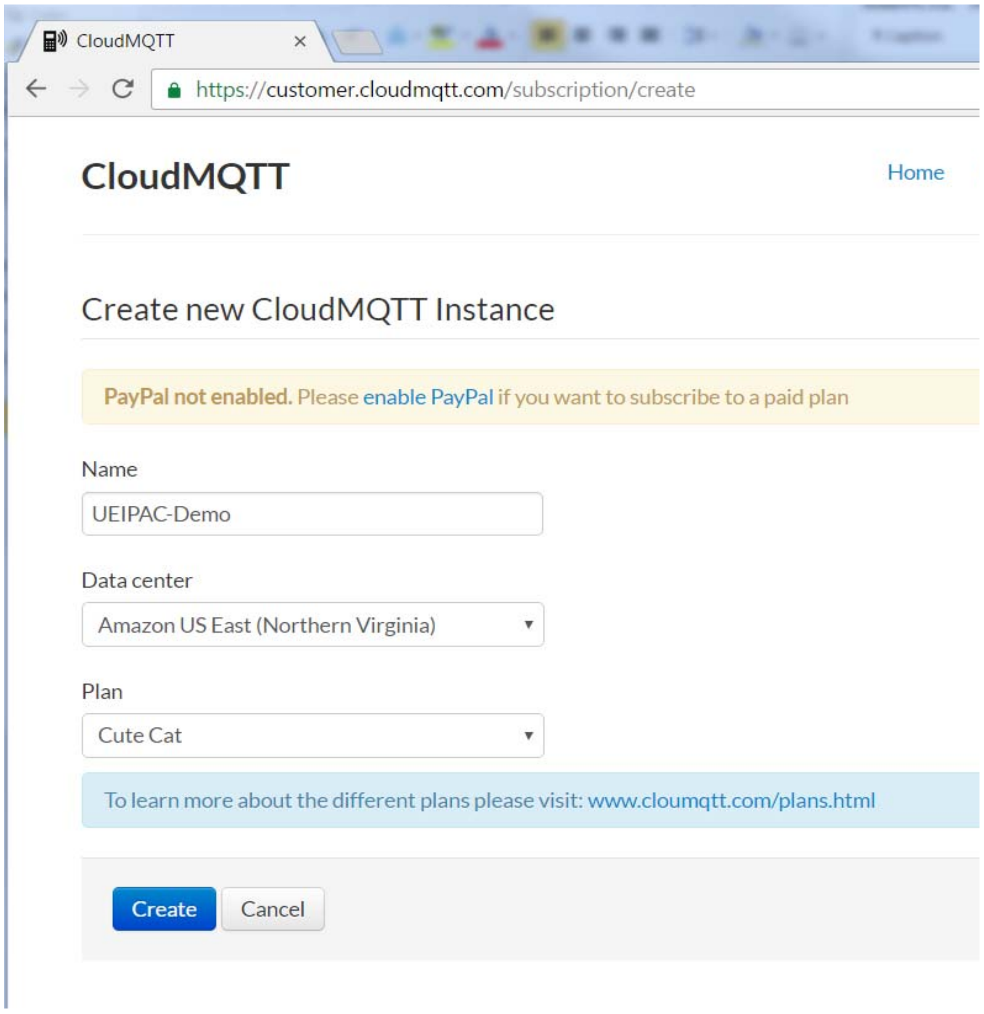
Note this IP address for later reference.

2. **Create an account with the cloud broker**

   In this example, we are using cloudmqtt.com as our hosted broker. Follow the instructions on the site to set up a login ID and password for the account.

3. **Create an instance with the cloud broker**

   Enter an instance name, and click **Create**:



Create new CloudMQTT Instance

PayPal not enabled. Please enable PayPal if you want to subscribe to a paid plan

Name	UEIPAC-Demo
Data center	Amazon US East (Northern Virginia)
Plan	Cute Cat

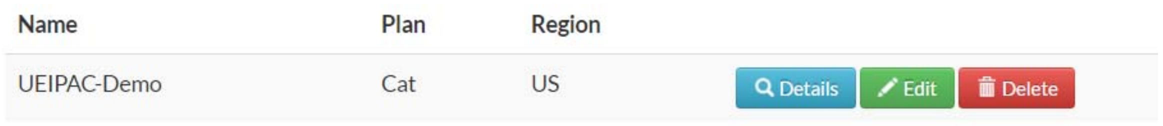
To learn more about the different plans please visit: www.cloumqtt.com/plans.html

Figure 8: Create an Instance on the hosted broker

The instance we created is "UEIPAC-Demo".

4. **Note host information for the instance**

   Click **Details** to access the **Instance Info** page:

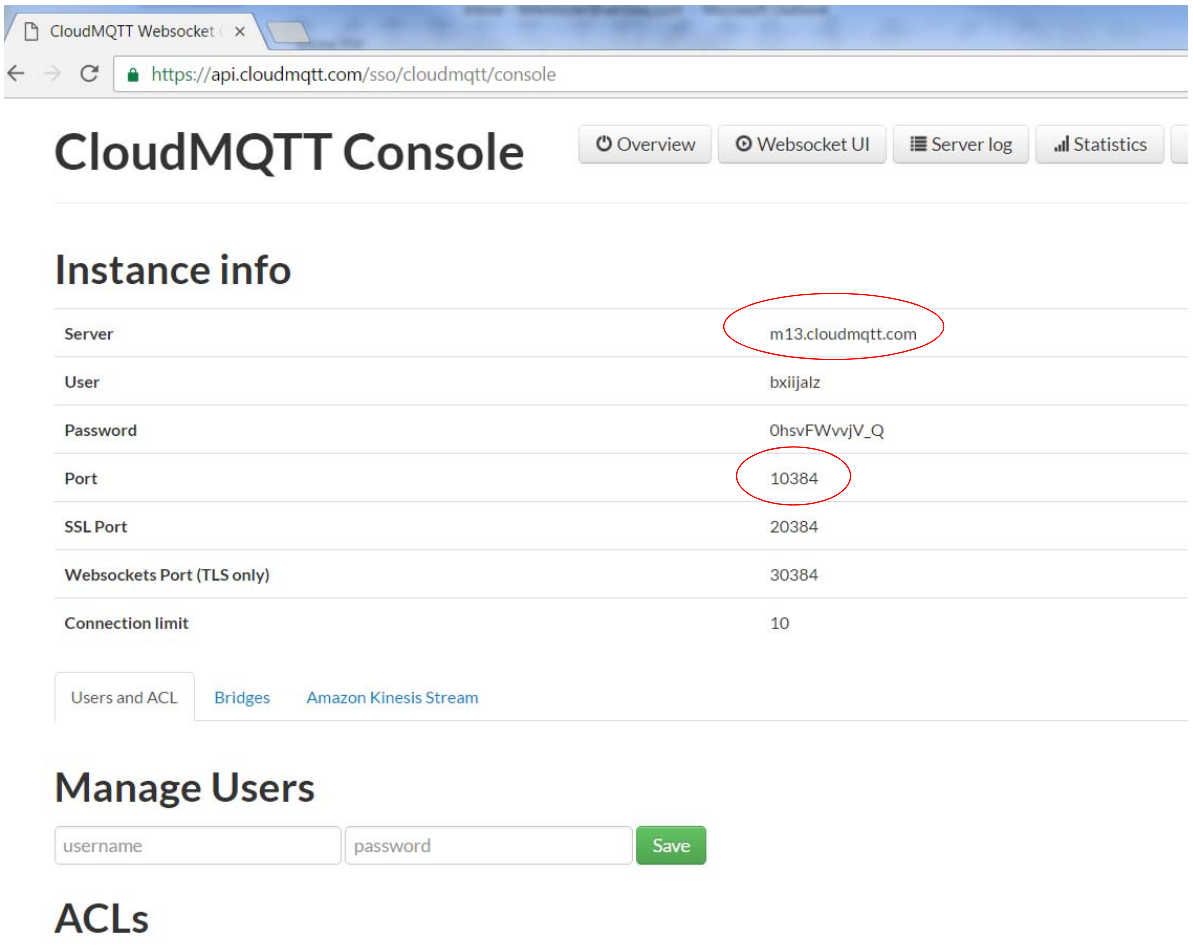


Name	Plan	Region	
UEIPAC-Demo	Cat	US	

Figure 9: Instance details on the hosted broker

The **CloudMQTT Console** will open.

Note the **Server** and **Port** information. These will be used later to connect the Mosquitto publication service on the UEIPAC to the cloudMQTT broker:



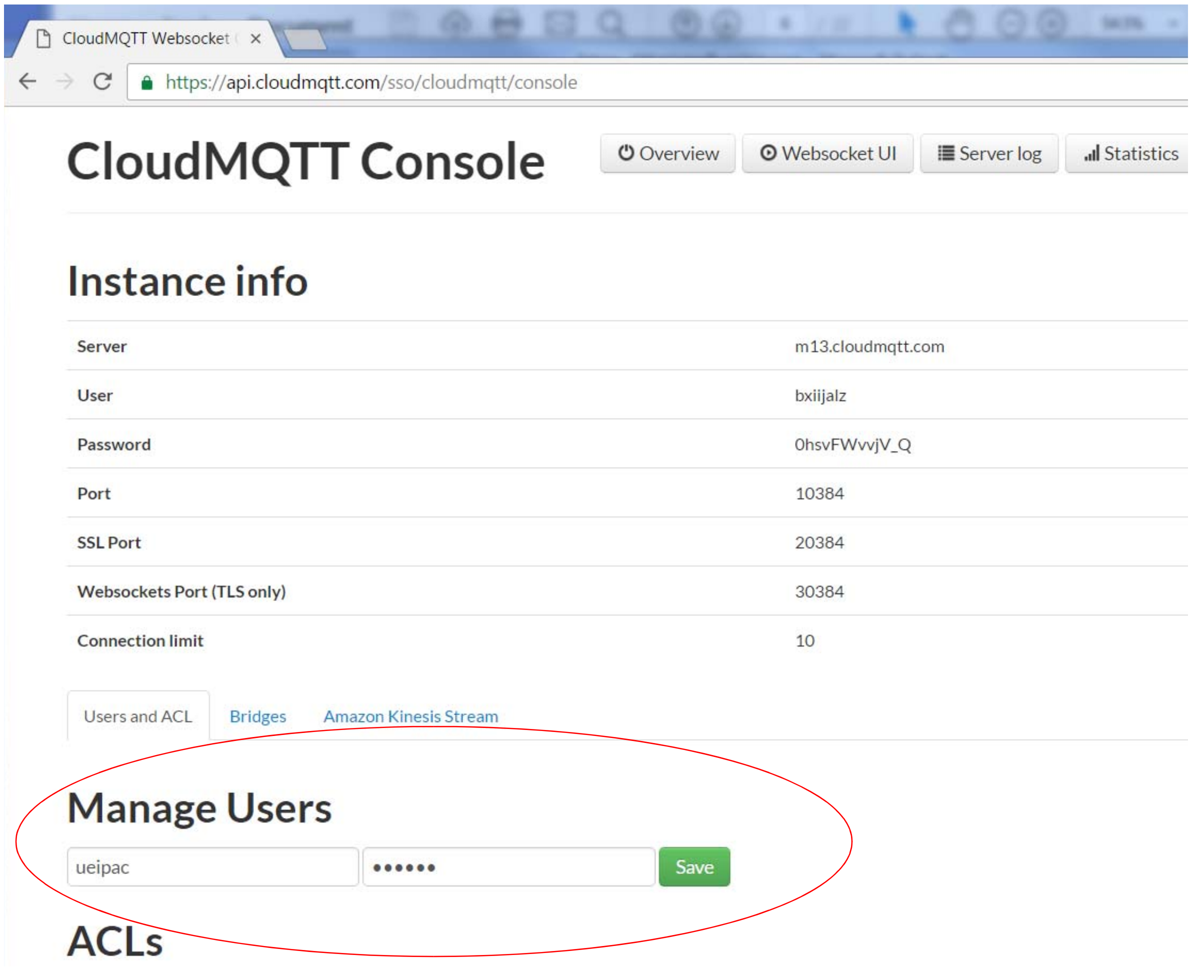
Instance info

Server	m13.cloudmqtt.com
User	bxiijalz
Password	0hsvFWvvjV_Q
Port	10384
SSL Port	20384
Websockets Port (TLS only)	30384
Connection limit	10

Figure 10: Note the `Server hostname` and `Port` on the hosted broker

5. Set up credentials for users of the instance

Enter a username and password under the **Manage Users** section, and click **Save**:



CloudMQTT Console

Instance info

Server	m13.cloudmqtt.com
User	bxiijalz
Password	0hsvFWvvjV_Q
Port	10384
SSL Port	20384
Websockets Port (TLS only)	30384
Connection limit	10

Manage Users

ueipac ••••••

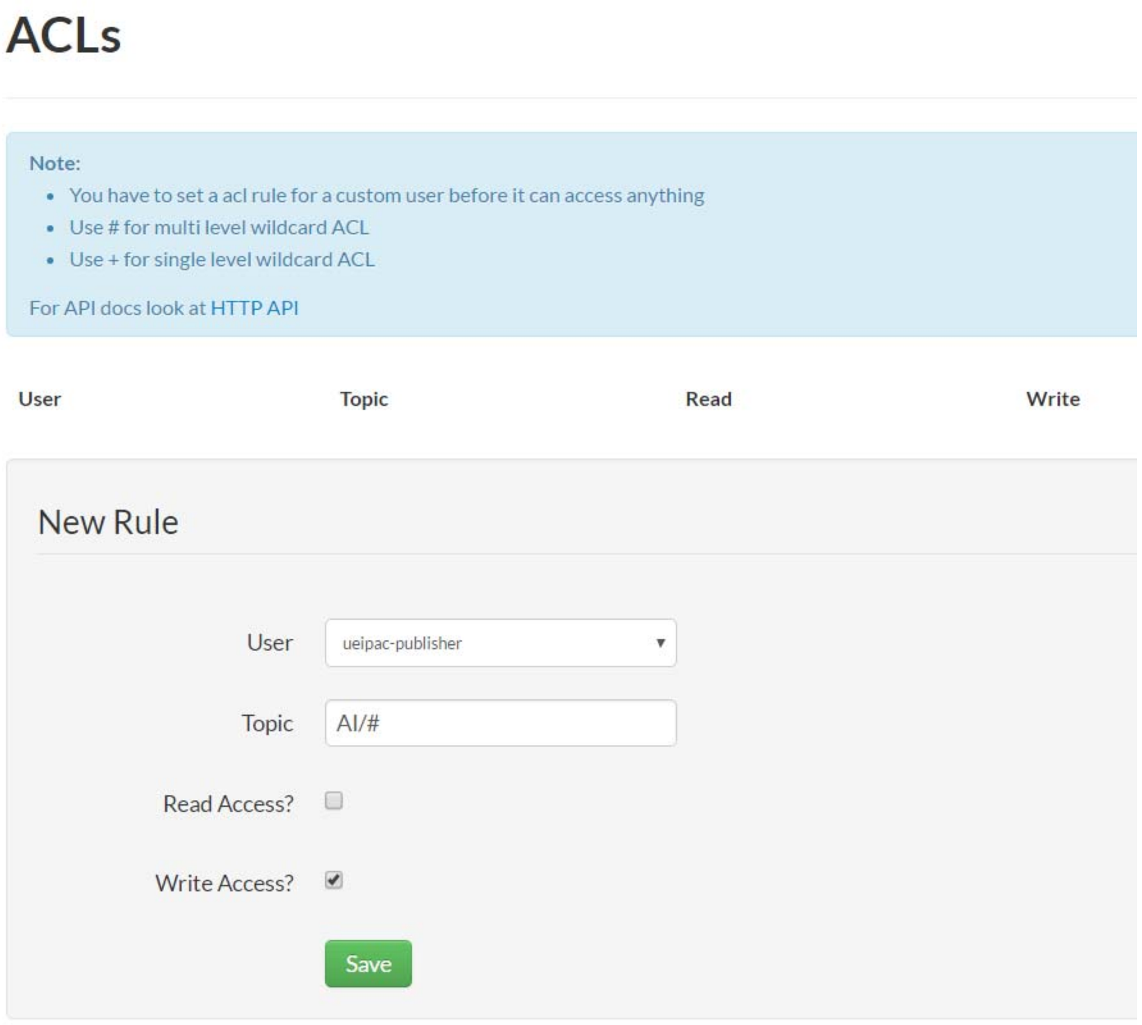
ACLs

Figure 11: Enter authentication credentials on the hosted broker

For this example, we set up two usernames: one for the UEIPAC as a publisher (`ueipac-publisher`), and one for the subscriber (`ueipac`).

6. **Set up Access Control List rules for publishing messages**

   Create a new Access Control List with a rule allowing the UEIPAC to publish all samples from analog input boards (`AI/#`) to this broker:



ACLs

Note:

- You have to set a acl rule for a custom user before it can access anything
- Use # for multi level wildcard ACL
- Use + for single level wildcard ACL

For API docs look at HTTP API

User	Topic	Read	Write

New Rule

User	ueipac-publisher
Topic	AI/#
Read Access?	☐
Write Access?	✓

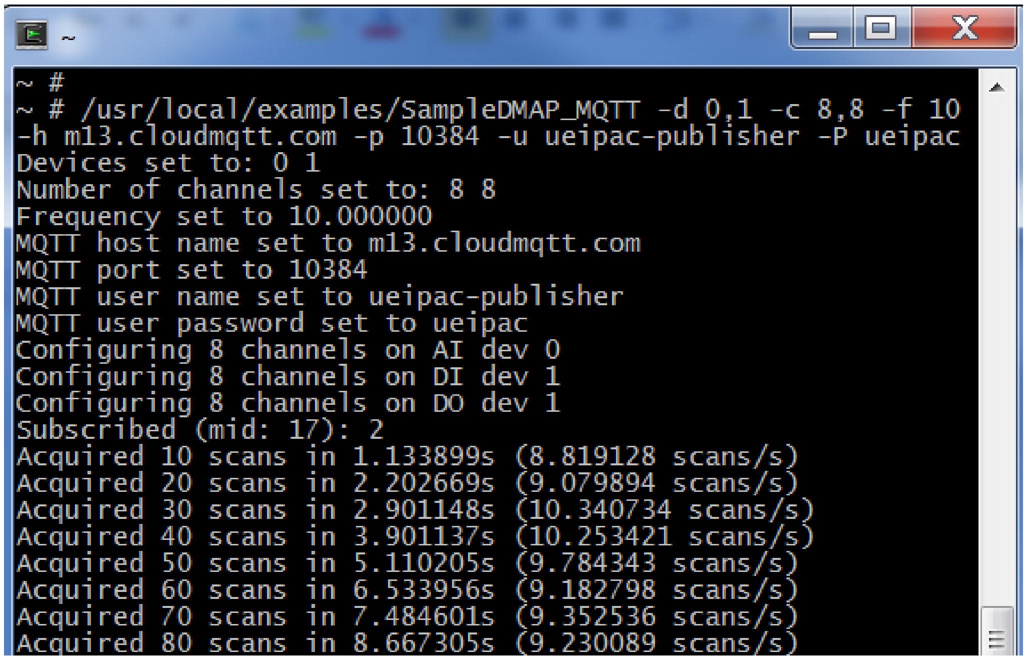
Figure 12: Enter ACL rules for publisher

In this example, we set the **User** to the credentials set up for the UEIPAC as a publisher (`ueipac-publisher`) and specify published topics from the UEIPAC will be the **Topic** `AI/#`. Publishers must have **Write Access** checked, and subscribers must have **Read Access** checked.

7. **Start the sample code on the UEIPAC**

   Type the following command at the UEIPAC Linux prompt:

```
~# /usr/local/examples/SampleDMAP_MQTT –d 0,1 –c 8,8 –f 10 –h
m13.cloudmqtt.com –p 10384 -u ueipac-publisher -P ueipac:
```



```
~ #
~ # /usr/local/examples/SampleDMAP_MQTT -d 0,1 -c 8,8 -f 10
-h m13.cloudmqtt.com -p 10384 -u ueipac-publisher -P ueipac
Devices set to: 0 1
Number of channels set to: 8 8
Frequency set to 10.000000
MQTT host name set to m13.cloudmqtt.com
MQTT port set to 10384
MQTT user name set to ueipac-publisher
MQTT user password set to ueipac
Configuring 8 channels on AI dev 0
Configuring 8 channels on DI dev 1
Configuring 8 channels on DO dev 1
Subscribed (mid: 17): 2
Acquired 10 scans in 1.133899s (8.819128 scans/s)
Acquired 20 scans in 2.202669s (9.079894 scans/s)
Acquired 30 scans in 2.901148s (10.340734 scans/s)
Acquired 40 scans in 3.901137s (10.253421 scans/s)
Acquired 50 scans in 5.110205s (9.784343 scans/s)
Acquired 60 scans in 6.533956s (9.182798 scans/s)
Acquired 70 scans in 7.484601s (9.352536 scans/s)
Acquired 80 scans in 8.667305s (9.230089 scans/s)
```

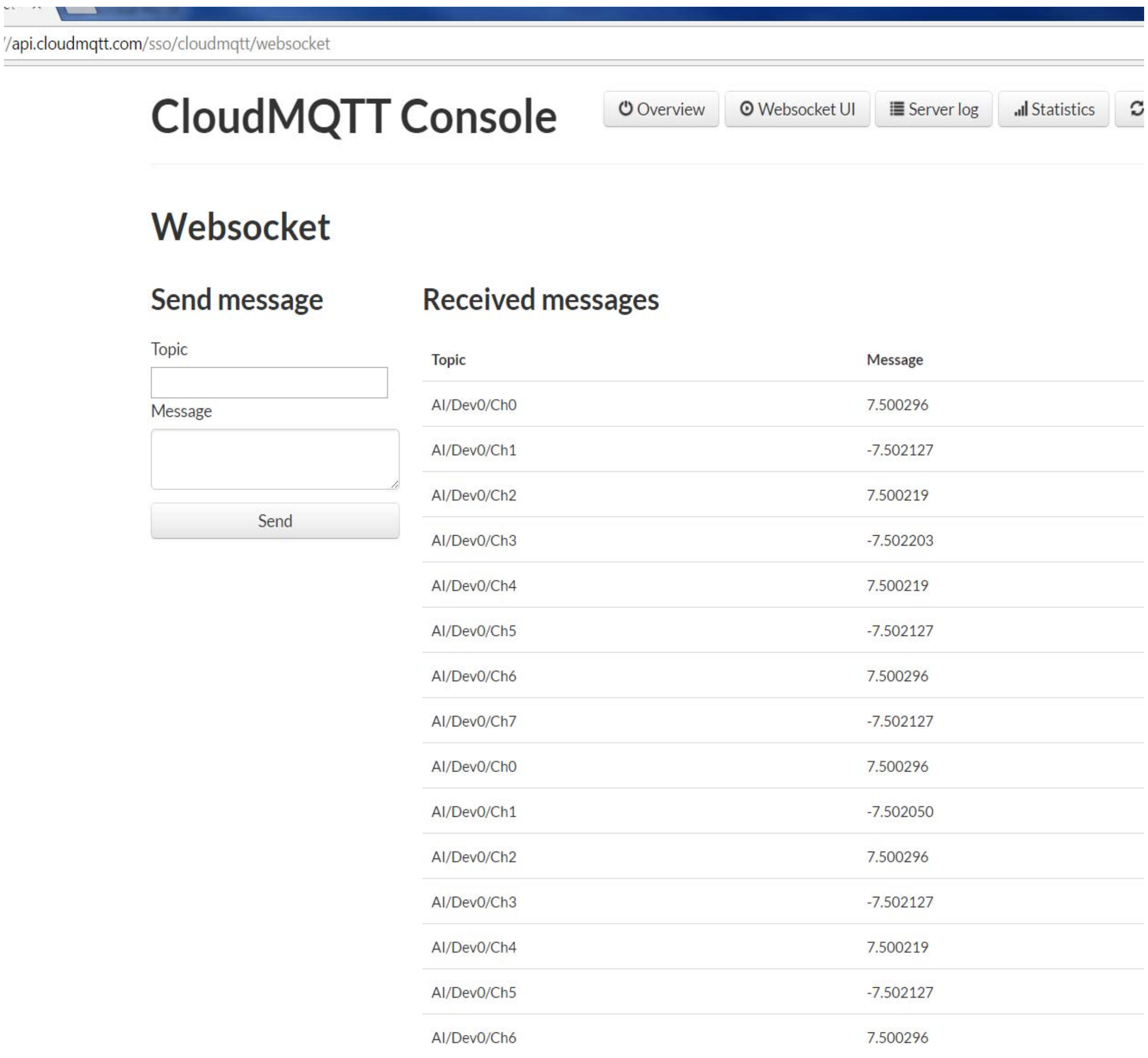
Figure 13: Launch MQTTLens Subscription Client

In this example, the host is the mqttcloud broker. The hostname is “m13.cloudmqtt.com” as noted when the instance was created in Step 4 on page 12. The port and credentials are also what were setup in the previous two steps.

Once we start the SampleDMAP_MQTT code running, the local Mosquitto publisher will publish analog input samples acquired from the first 8 channels on board 0 (Dev0) and the first 8 channels on board 1 (Dev1) this time to the cloud based MQTT broker.

8. **Verify the cloud broker is receiving published data**

   Look on the cloud broker website to verify published data is being received. To check on cloudmqtt, click **Websocket UI** in the **CloudMQTT Console** window:



Topic	Message
AI/Dev0/Ch0	7.500296
AI/Dev0/Ch1	-7.502127
AI/Dev0/Ch2	7.500219
AI/Dev0/Ch3	-7.502203
AI/Dev0/Ch4	7.500219
AI/Dev0/Ch5	-7.502127
AI/Dev0/Ch6	7.500296
AI/Dev0/Ch7	-7.502127
AI/Dev0/Ch0	7.500296
AI/Dev0/Ch1	-7.502050
AI/Dev0/Ch2	7.500296
AI/Dev0/Ch3	-7.502127
AI/Dev0/Ch4	7.500219
AI/Dev0/Ch5	-7.502127
AI/Dev0/Ch6	7.500296

Figure 14: Verify the broker is receiving published messages

In this example, we have an analog input board (AI) as the first board in the UEIPAC chassis (Dev0), and we are publishing data from channels 0 through 7.

At this point, any MQTT client from any location in the world can subscribe to this data.

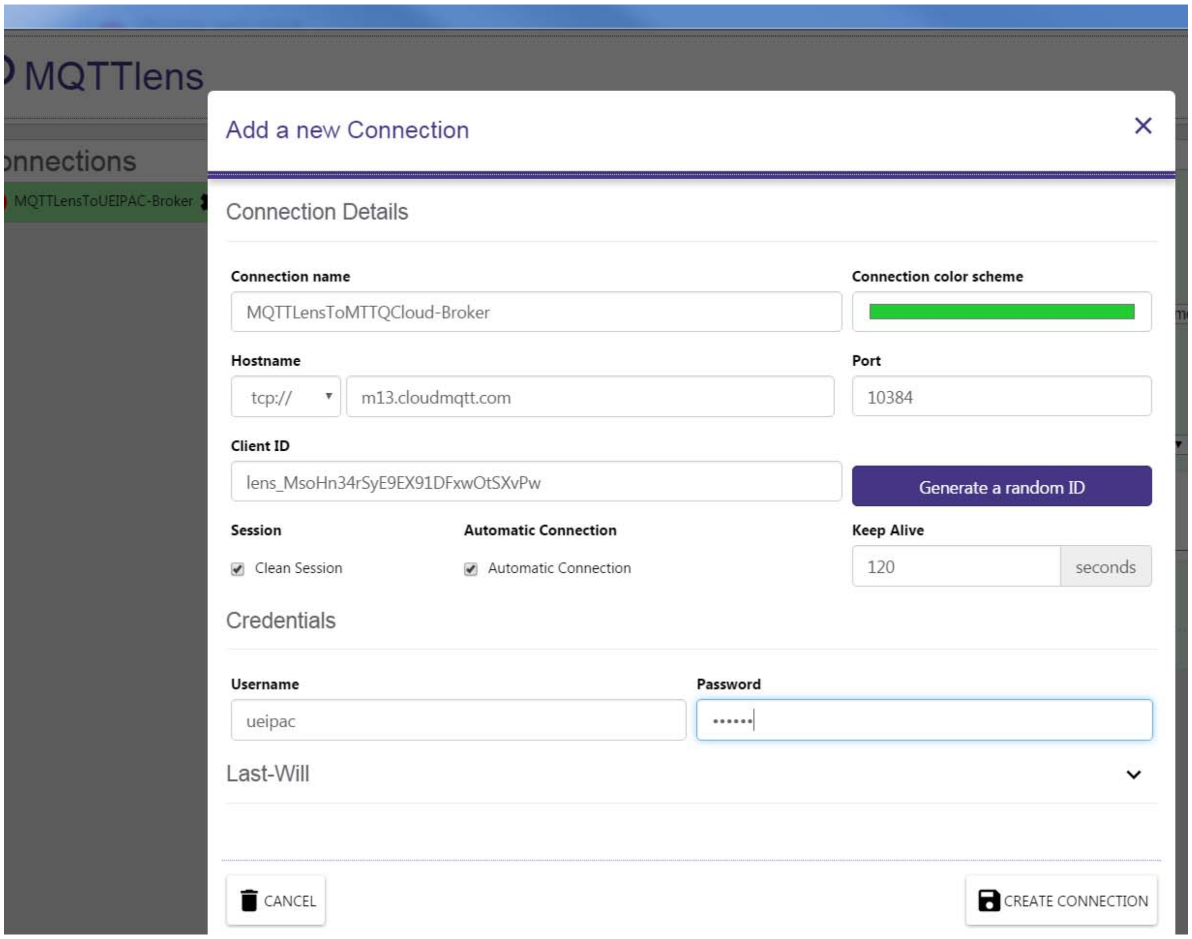
The following steps set up Google Chrome's MQTTLens (used in the previous tutorial) and the iPhone app, ICPDAS MQTT, to subscribe to the mqttcloud.com broker and view samples.

9. **Launch MQTT Lens**

   If MQTTLens is not already installed, follow set up instructions in the previous tutorial.

10. **Setup mqttcloud as the broker connection for MQTTLens subscription**

    Click **Connections +** to add a connection to the cloud broker, and enter **Connection name**, **Port**, and **Username** and **Password** credentials:



Connection name: MQTTLensToMTTQCloud-Broker
Connection color scheme: 🟩
Hostname: tcp:// m13.cloudmqtt.com
Port: 10384
Client ID: lens_MsoHn34rSyE9EX91DFxwOtSXvPw
Session: ✓ Clean Session
Automatic Connection: ✓ Automatic Connection
Keep Alive: 120 seconds
Username: ueipac
Password: ••••••

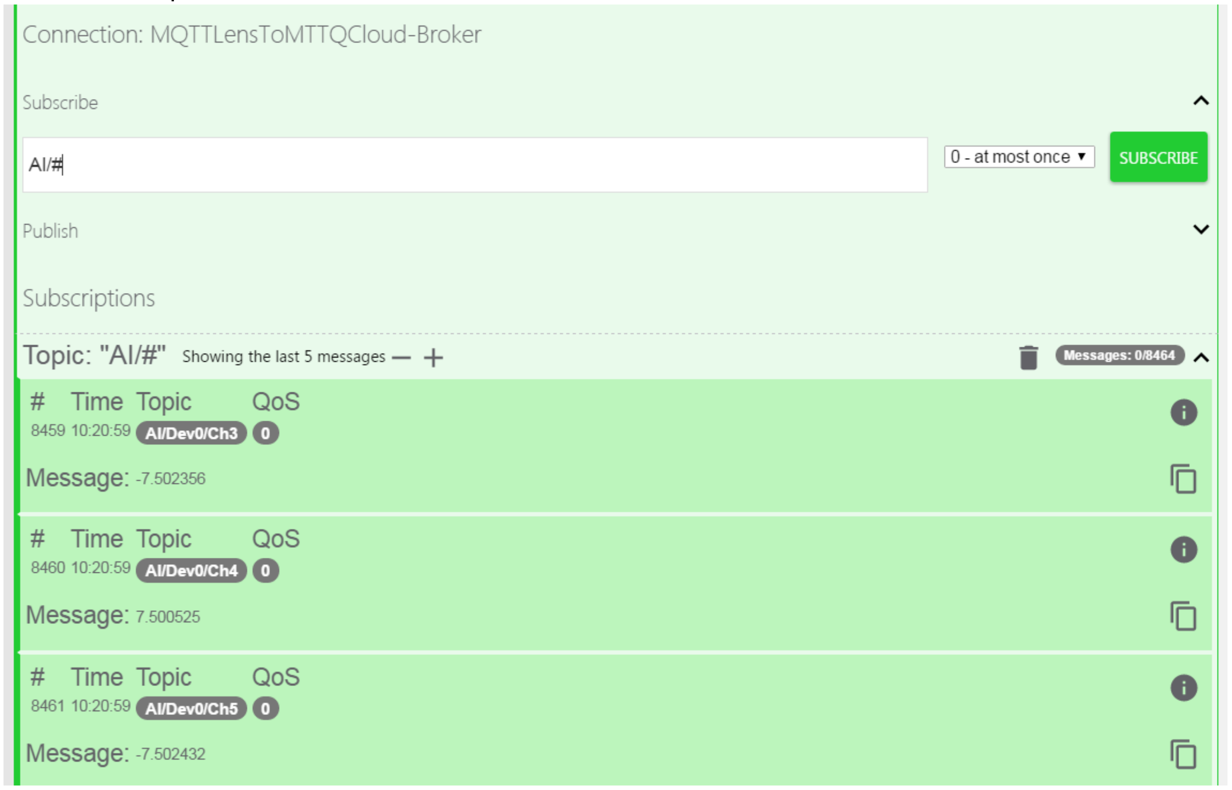
Figure 15: Enter Broker Connection Information for MQTTLens Subscription Client

Click **Save Changes** to save setup.

The broker is the mqttcloud service. Refer to the instance setup in previous steps for the **Server**, **Port**, and user credentials.

11. **Subscribe to topics using the MQTT subscription client**

Enter `AI/#` in the **Subscribe** field to acquire data messages from all channels on all analog input boards, and then click **SUBSCRIBE**.


Connection: MQTTLensToMTTQCloud-Broker

Subscribe

AI/#

0 - at most once

Subscriptions

Topic: "AI/#" Showing the last 5 messages

Messages: 0/8464

#	Time	Topic	QoS
8459	10:20:59	AI/Dev0/Ch3	0

Message: -7.502356

#	Time	Topic	QoS
8460	10:20:59	AI/Dev0/Ch4	0

Message: 7.500525

#	Time	Topic	QoS
8461	10:20:59	AI/Dev0/Ch5	0

Message: -7.502432

Figure 16: MQTTLens Displays Subscription Topics

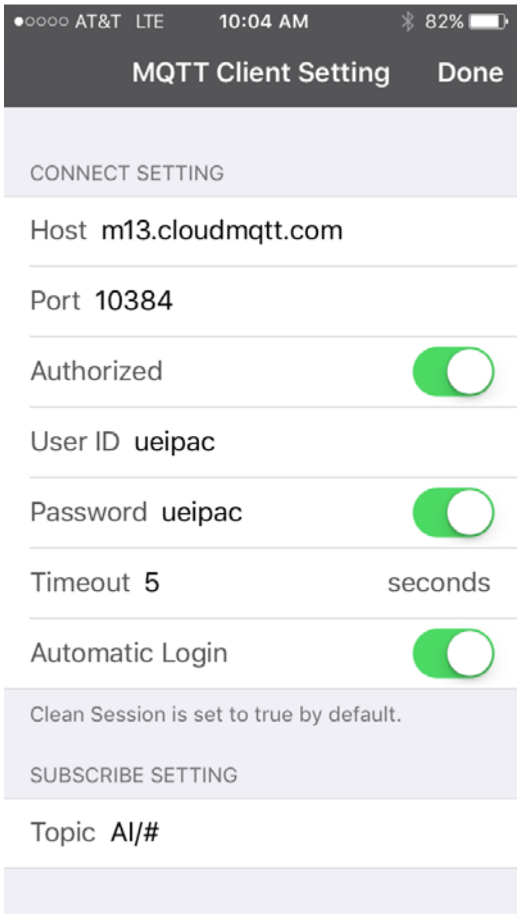
Topics print in the GUI display.

12. Open an MQTT client on your hand-held device

The following steps use the free iOS MQTT client app, ICPDAS MQTT.

13. Setup mqttcloud as the broker connection for the MQTT client on the hand-held device

Enter connection information for the cloud broker: **Host**, **Port**, **User ID** and **Password** credentials, and **Topic** settings can be found in step 4 through step 6:



CONNECT SETTING

Host m13.cloudmqtt.com

Port 10384

Authorized

User ID ueipac

Password ueipac

Timeout 5 seconds

Automatic Login

Clean Session is set to true by default.

SUBSCRIBE SETTING

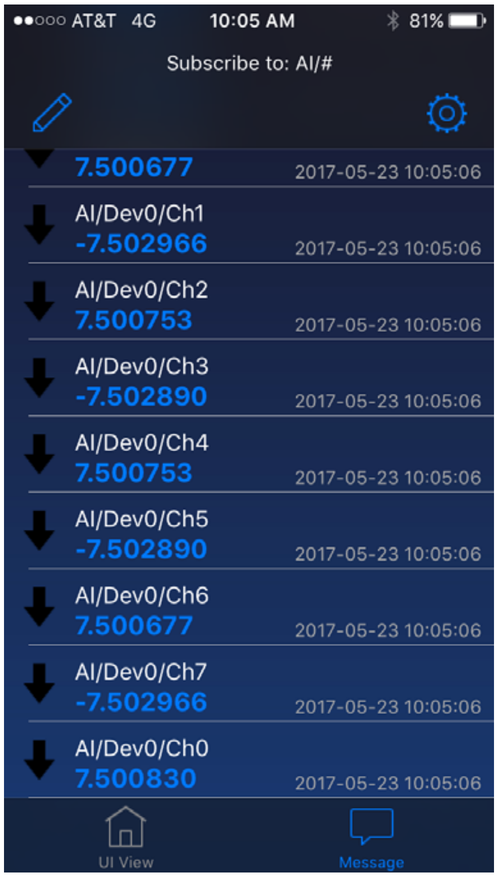
Topic AI/#

Figure 17: Enter Broker Credentials into iOS Subscription Client

Click **Done** to finish setup. The hand-held device will connect to the broker.

14. View real-time messages

For the app in this example, press the **Messages** button in the lower right corner to view real-time updates of the analog input sample data.



7.500677 2017-05-23 10:05:06

AI/Dev0/Ch1
-7.502966 2017-05-23 10:05:06

AI/Dev0/Ch2
7.500753 2017-05-23 10:05:06

AI/Dev0/Ch3
-7.502890 2017-05-23 10:05:06

AI/Dev0/Ch4
7.500753 2017-05-23 10:05:06

AI/Dev0/Ch5
-7.502890 2017-05-23 10:05:06

AI/Dev0/Ch6
7.500677 2017-05-23 10:05:06

AI/Dev0/Ch7
-7.502966 2017-05-23 10:05:06

AI/Dev0/Ch0
7.500830 2017-05-23 10:05:06

Figure 18: Hand-held Display of Analog Input Data

Topics print in the GUI display.

In summary

As industries progress to streamlining or automating operations, interconnected “things” that are able to communicate in real-time with data processing equipment become increasingly important. As an example, one big-data analytic scenario for the airline industry estimates the considerable amount of flight operation data from onboard sensors that will need to be transferred to processing stations on the ground for collection, storage and processing of data may be in the terabyte range.[7]

UEI provides rugged and flexible solutions that support IoT applications, with IoT hardware and software configurations offered through our programmable automation controllers (UEIPACs). UEIPACs are installed with any combination of UEI’s extensive selection of input/output boards to meet the needs of an application, providing a modular design that facilitates communication and interoperability with other devices, networks, and systems.

© 2016 United Electronic Industries, Inc. All Rights Reserved.

Endnotes

1. Source: IEEE Spectrum.
   Nordrum, Amy, “Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated” (Tech Talk, Aug 2016), spectrum.ieee.org/tech-talk/telecom/internet.
2. Source: Industrial Internet Consortium.
   Industrial Internet Consortium, Manufacturing & Smart Factory, www.iiconsortium.org/vertical-markets/manufacturing.htm.
3. Source: LinkedIn article.
   Stansberry, James, “The IoT communication protocols” (October 2015), www.linkedin.com/pulse/iot-communication-protocols-james-stansberry.
4. Source: SlideShare.
   Patierno, Paolo, “IoT protocols landscape” (June 2014), www.slideshare.net/paolopat/io-t-protocols-landscape.
5. Source: Wind River Press Release.
   Wind River, “Wind River Unveils IoT-Enabled Product Portfoilio” (February 2015), https://www.windriver.com/news/press/pr.html?ID=13288.
6. Source: OPC Foundation Press Release.
   OPC Foundation, “OPC Foundation Announces support of Publish / Subscribe for OPC UA” (May 2016), opcfoundation.org/news/opc-foundation-news.
7. Source: Sampathkumar, Arun Kumar, “Commercial Aviation, Big Data and What I Think About Them” (September 2014),
   www.linkedin.com/pulse/20140923105538-238203970-commercial-aviation-big-data-and-what-i-think-about-them?trk=pulse_spock-articles.

Evans, Dave, “How the Next Evolution of the Internet Is Changing Everything” (April 2011), www.cisco.com/c/dam/en_us/about/ac79/docs/innov.

Heimes,Felix, “Fleet Health Monitoring and Machine Learning Technology for CBM+” (October 2009), www.phmsociety.org/sites/phmsociety.org/files/FieldedSystems_Dresch.pdf.

Keranen, Ari and Bormann, Carsten, “Internet of Things: Standards and Guidance from the IETF” (April 2016). www.internetsociety.org/publications/ietf-journal-april-2016.

Schneider, Stan, “Understanding the Protocols Behind the Internet of Things” (Oct 2013), electronicdesign.com/iot/understanding-protocols-behind-internet-things

About United Electronic Industries United Electronic Industries (UEI) is a leader in the data acquisition and control, data logging, and embedded automation controller markets. Our cube, rack and military chassis provide a rugged and powerful platform that is ideal for many applications in the industrial, aerospace, military, and simulation fields – plus many more! Our chassis are uniquely flexible, capable of being deployed as a remote Ethernet I/O node, a standalone data logger, or an embedded controller. We offer incredible I/O flexibility with over 60 I/O boards available which include analog/digital I/O, avionics (including 1553, 429, AFDX™, 664, 708/453, 825), speed/freq input, CAN-bus, serial I/O and more. UEI supports all popular OSs such as Windows, Linux, VxWorks, QNX and other RTOSs as well as all programming languages. We also offer complete and seamless support of all major application packages, including LabVIEW™, MATLAB® and Simulink™.