

Chapter 8: IIoT (Industrial Internet of Things)

In this Chapter...

8.1	IIoT (Industrial Internet of Things)	276
8.1.1	MQTT	277
8.2	MQTT Essentials	280
8.2.1	Basic Concepts	280
8.2.2	MQTT More details and Examples	282
8.3	Basic MQTT Setup on EZLogix.....	294
8.4	Broker Setup	296
8.5	EZLogix IIoT (MQTT) Example	298
8.6	MQTT HIVEMQ Essentials.....	302
8.7	EZ-IIoT Subscriber Utility	307
8.7.1	Install EZ-IIoT Subscriber Utility.....	307
8.7.2	EZ-IIoT Subscriber Utility Setup	308
8.7.3	EZ-IIoT Subscriber Utility Function	310
8.7.4	EZ-IIoT Subscriber Utility Best Practices	319

8.1 IIOT (Industrial Internet of Things)

The EZLogix PLC supports the Industrial Internet of Things. The EZLogix PLC comes with a built in instruction that allows the user to publish data to secure offsite MQTT Cloud Broker. This capability allows the EZLogix PLC to provide data for analysis to improve efficiency, troubleshoot problems, and do preventative maintenance. This section explores in more depth what IIoT (Industrial Internet of Things) means, how the EZLogix supports IIoT through the MQTT protocol, and finally looks at how to setup the EZLogix to do MQTT communication.

What is IIoT (Industrial Internet of Things)?

The Industrial Internet of Things (IIoT) focuses on the interconnectivity and utilization of powerful data in a manufacturing environment. IIoT enables the acquisition and accessibility of important plant data at far greater speeds, security and reliability. IIoT incorporates machine learning and big data technology, harnessing the sensor data, machine-to-machine communication and automation technologies that have existed in industrial settings for years. The driving philosophy behind the IIoT is that smart machines are better than humans at accurately, consistently capturing and communicating data.

EZLogix built in IIoT and MQTT protocol support acts as a "bridge" between existing operational technology within a plant, for example factory machines, and plant database networks, so valuable data can be shared reliably and securely to improve plant productivity and efficiency.

How EZLogix Support “Edge-Gateway” Communications and IIoT?

The EZLogix PLC operates as an “Edge-of-Network” or “Edge-Gateway” device with direct connectivity to external devices such as sensors, RTDs, analog inputs, etc. and easy to setup secure communications with other networks such as Modbus TCP/IP. Through the use of the MQTT protocol it can publish up to 80 tags of data per EZLogix CPU, thus providing a subscriber pertinent real time data from these external devices. The use of the MQTT protocol allows for great interoperability since it is becoming an industry standard. It also allows for great security through the broker. It must also be noted that with the EZLogix PLC, a “security breach” to access the machine is not of any concern since there is no backwards flow of data. That is data is only ever published from the PLC. It will never accept any data or commands back from any server, broker or client.

8.1.1 MQTT

What is MQTT?

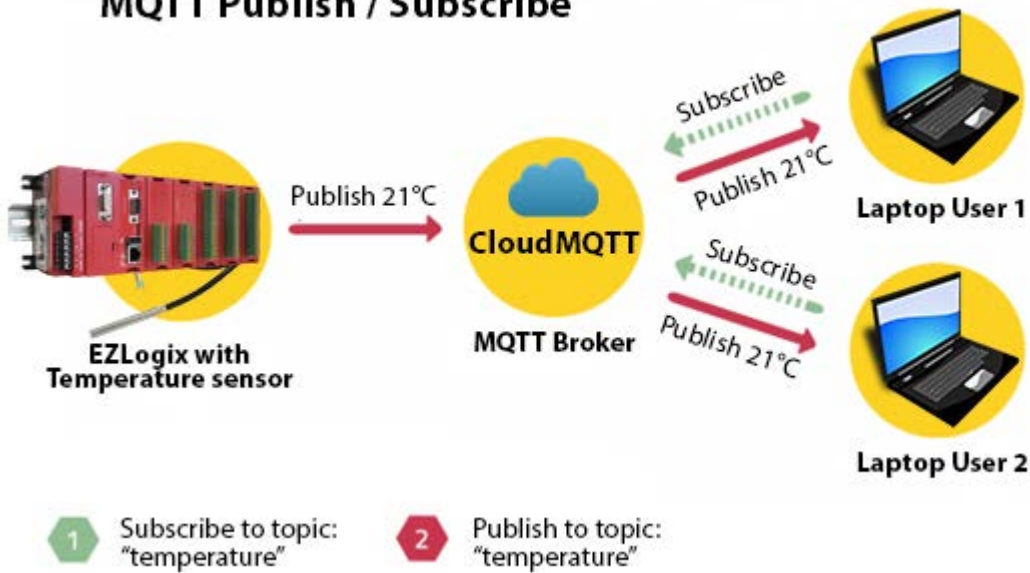
MQTT which stands for message queuing telemetry transport, is a standard Client Server publish/subscribe messaging transport protocol that is quickly becoming the leading messaging protocol for the Industrial Internet of Things (IIoT).

How does MQTT work?

The MQTT protocol works on a publish/subscribe (pub/sub) pattern. This is different from a traditional client-server model in that the machine (PLC) does not directly communicate to the server. The Pub/Sub pattern decouples a client that is publishing data (sending messages) from the client that is subscribing to the data (receiving messages). For this pattern the sender of messages is called the publisher and the receiver of messages is called the subscriber.

This Pub/Sub pattern essentially creates a barrier between the publisher and subscriber in that they do not know about the existence of the other. The broker who is known by both is the link between them. The broker can filter all the messages and distribute them to the subscriber that is supposed to receive them. Multiple subscribers can be receiving messages from the broker at the same time but getting different data. This allows for separating out access so only pertinent data is received to selected individuals or “subscribers”. The graphic below shows how Pub/Sub works.

MQTT Publish / Subscribe



What is the current functionality of EZLogix?

The EZLogix PLC currently works as a publisher of data in the MQTT pattern. It is very flexible in that it works with any broker that the customer would like. The EZLogix PLC connects to the broker with a username and password for security and then can publish up to 80 tags, also known as topics, at settable intervals.

Why does EZLogix as an Edge-Gateway device use MQTT for its communication?

The MQTT protocol is becoming the industry standard communication protocol for IIoT. More importantly, the MQTT protocol, provides a "bridge" between existing operational technology within a plant, for example factory machines, and plant database networks so valuable data can be shared reliably and securely to improve plant productivity and efficiency.

Misconceptions of IIoT and MQTT.

1. Implementation is extremely costly.

The EZLogix PLC, with base rack, CPU and power supply all included is at an extremely attractive price of \$248 and has IIoT MQTT protocol built in, among many other features including data-logging, ladder logic and function blocks, auto-tuned PID and much more...

2. Better to wait for an Industry Consensus.

MQTT is becoming the industry standard therefore a consensus around it is developing in the manufacturing and process sectors. But even if that wasn't true, MQTT is a light and versatile protocol which can allow for communication with many different machines and plant devices.

3. Is implementing IIoT really worth it?

IIoT connectivity allows "management" to see plant performance thereby allowing the plant to optimize and track their production and efficiency. Furthermore, the implementation of IIoT can help with offsite troubleshooting and offsite analysis of production data.

4. Adding IIoT will be complicated.

The EZLogix PLC has IIoT MQTT protocol built in and therefore it is a very easy setup process. Also if in the future you wish to add IIoT to an existing system no major changes are needed. Please see the MQTT Essentials section to understand how simple IIoT MQTT protocol is.

8.2 MQTT Essentials

This section explores the basics of MQTT and how it functions. For easy setup guide please see *Section 8.3, 8.4 and 8.5*.

8.2.1 Basic Concepts

Referenced from <http://mqtt.org/> and <http://mosquitto.org/man/mqtt-7.html>

Publish/Subscribe

The MQTT protocol is based on the principle of publishing messages and subscribing to topics, or "pub/sub". Multiple clients connect to a broker and subscribe to topics that they are interested in. Clients also connect to the broker and publish messages to topics. Many clients may subscribe to the same topics and do with the information as they please. The broker and MQTT act as a simple, common interface for everything to connect to. This means that if you have clients that dump subscribed messages to a database, for example Twitter, or even a simple text file, then it becomes very simple to add new sensors or other data input to a database, Twitter or so on.

Topics/Subscriptions

Messages in MQTT are published on topics. There is no need to configure a topic, publishing on it is enough. Topics are treated as a hierarchy, using a slash (/) as a separator. This allows sensible arrangement of common themes to be created, much in the same way as a filesystem. For example, multiple computers may all publish their hard drive temperature information on the following topic, with their own computer and hard drive name being replaced as appropriate:

```
sensors/COMPUTER_NAME/temperature/HARDDRIVE_NAME
```

Clients can receive messages by creating subscriptions. A subscription may be to an explicit topic, in which case only messages to that topic will be received, or it may include wildcards.

Quality of Service

MQTT defines three levels of Quality of Service (QoS). The QoS defines how hard the broker/client will try to ensure that a message is received. Messages may be sent at any QoS level, and clients may attempt to subscribe to topics at any QoS level.

Higher levels of QoS are more reliable, but involve higher latency and have higher bandwidth requirements.

0: The broker/client will deliver the message once, with no confirmation.

1: The broker/client will deliver the message at least once, with confirmation required.

2: The broker/client will deliver the message exactly once by using a four step handshake.

Retained Messages

All messages may be set to be retained. This means that the broker will keep the message even after sending it to all current subscribers. If a new subscription is made that matches the topic of the retained message, then the message will be sent to the client. This is useful as a "last known good" mechanism. If a topic is only updated infrequently, then without a retained message, a newly subscribed client may have to wait a long time to receive an update. With a retained message, the client will receive an instant update.

8.2.2 MQTT More Details and Examples

Referenced from <http://www.hivemq.com/blog/mqtt-essentials/> and <http://mosquitto.org/>. A full explanations of how MQTT function can be found in *section 8.6*.

MQTT History

MQTT was invented by Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, now Cirrus Link) back in 1999, when their use case was to create a protocol for minimal battery loss and minimal bandwidth connecting oil pipelines over satellite connection. They specified the following goals, which the future protocol should have:

- Simple to implement
- Provide a Quality of Service Data Delivery
- Lightweight and Bandwidth Efficient
- Data Agnostic
- Continuous Session Awareness

These goals are still the core of MQTT, while the focus has changed from proprietary embedded systems to open Internet of Things use cases. Another thing that is often confused about MQTT is the appropriate meaning of the abbreviation MQTT. It's a long story, the short answer is that MQTT officially does not have an acronym anymore, it's just MQTT.

OASIS Standard

Around 3 years after the initial publication, it was announced that MQTT should be standardized under the wings of OASIS, an open organization with the purpose of advancing standards. On October 29th 2014 [MQTT was officially approved as OASIS Standard](#). MQTT 3.1.1 is now the newest version of the protocol.

Definition of Client/Broker

Client

When talking about a client it almost always means an MQTT client. This includes publisher or subscribers, both of them label an MQTT client that is only doing publishing or subscribing. (In general a MQTT client can be both a publisher & subscriber at the same time). A MQTT client is any device from a micro controller up to a full-fledged server that has a MQTT library running and is connecting to an MQTT broker over any kind of network. This could be a really small and resource constrained device that is connected over a wireless network and has a library strapped to the minimum or a typical computer running a graphical MQTT client for testing purposes, basically any device that has a TCP/IP stack and speaks MQTT over it.

Broker

The counterpart to a MQTT client is the MQTT broker, which is the heart of any publish/subscribe protocol. Depending on the concrete implementation, a broker can handle up to thousands of concurrently connected MQTT clients. The broker is primarily responsible for receiving all messages, filtering them, decide who is interested in it and then sending the message to all subscribed clients. Another responsibility of the broker is the authentication and authorization of clients. And at most of the times a broker is also extensible, which allows to easily integrate custom authentication, authorization and integration into backend systems. Especially the integration is an important aspect, because often the broker is the component, which is directly exposed on the internet and handles a lot of clients and then passes messages along to downstream analyzing and processing systems. All in all the broker is the central hub, which every message needs to pass.

Note: A broker has only 1 message per topic therefore for data acquisition a server client (cloud storage) or any such devices with data storage capability needs to be used. They will subscribe to the broker and store all the messages seen.

Quality of Service Expanded

Higher levels of QoS are more reliable, but involve higher latency and have higher bandwidth requirements.

0: The broker/client will deliver the message once, with no confirmation. Since there is no confirmation the message might not be delivered if connection is bad. This is often called “fire and forget” and provides the same guarantee as the underlying TCP protocol.

1: The broker/client will deliver the message at least once, with confirmation required. Will send message till confirmation received so possible that multiples of the message can exist.

2: The broker/client will deliver the message exactly once by using a four step handshake. Will always have exactly one of the message delivered. It is the slowest quality of service level. Currently not supported by the EZLogix PLC.

The client chooses the maximum QoS it will receive. For example, if a message is published at QoS 2 and a client is subscribed with QoS 0, the message will be delivered to that client with QoS 0. If a second client is also subscribed to the same topic, but with QoS 2, then it will receive the same message but with QoS 2. For a second example, if a client is subscribed with QoS 2 and a message is published on QoS 0, the client will receive it on QoS 0.

Best Practice

The following should provide you some guidance if you are also confronted with this decision. Often this is heavily depending on your use case.

Use QoS 0 when ...

- You have a complete or almost stable connection between sender and receiver. A classic use case is when connecting a test client or a front end application to a MQTT broker over a wired connection.
- You don't care if one or more messages are lost once a while. That is sometimes the case if the data is not that important or will be send at short intervals, where it is okay that messages might get lost.
- You don't need any message queuing. Messages are only queued for disconnected clients if they have QoS 1 or 2 and a persistent session.

Use QoS 1 when ...

- You need to get every message and your use case can handle duplicates. The most often used QoS is level 1, because it guarantees the message arrives at least once. Of course your application must be tolerating duplicates and process them accordingly.
- You can't bear the overhead of QoS 2. Of course QoS 1 is a lot faster in delivering messages without the guarantee of level 2.

Use QoS 2 when ...

- It is critical to your application to receive all messages exactly once. This is often the case if a duplicate delivery would do harm to application users or subscribing clients. You should be aware of the overhead and that it takes a bit longer to complete the QoS 2 flow. Currently not supported by the EZLogix PLC.

Queuing of QoS 1 and 2 messages

All messages sent with QoS 1 and 2 will also be queued for offline clients, until they are available again. But queuing is only happening, if the client has a persistent session (durable connection).

Topics

A topic is a UTF-8 string, which is used by the broker to filter messages for each connected client. A topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).

WildCards (Topics)

For topic navigation there exist wildcards. Two wildcards are available, + or # for use in topics. These allow for easier access to different ranges of topics.

+ can be used as a wildcard for a single level of hierarchy. An example of its use:

sensors+/temperature/+

can be used as a wildcard for all remaining levels of hierarchy. This means that it must be the final character in a subscription. An example of its use:

sensors/machine/temperature/#

Example +

As another example, for a topic of "a/b/c/d", the following example subscriptions will match:

a/b/c/d +/b/c/d a/+/c/d a/+/+/d +/+/+/+

The following subscriptions will not match:

a/b/c b/+/c/d +/+/+

Example #

With a topic of "a/b/c/d", the following example subscriptions will match:

a/# a/b/# a/b/c/# +/b/c/#

Topic Best Practices

So these were the basics about MQTT message topics. As you can see, MQTT topics are dynamically and give great flexibility to its creator. But when using these in real world applications there are some challenges you should be aware of.

Don't use a leading forward slash

It is allowed to use a leading forward slash in MQTT, for example `/myhome/groundfloor/livingroom`. But that introduces an unnecessary topic level with a zero character at the front. That should be avoided, because it doesn't provide any benefit and often leads to confusion.

Don't use spaces in a topic

A space is the natural enemy of each programmer, they often make it much harder to read and debug topics, when things are not going the way, they should be. So similar to the first one, only because something is allowed doesn't mean it should be used. UTF-8 knows many different white space types, it's pretty obvious that such uncommon characters should be avoided.

Keep the topic short and concise

Each topic will be included in every message it is used in, so you should think about making them short and concise. When it comes to small devices, each byte counts and makes really a difference.

Use only ASCII characters, avoid non printable characters

Using non-ASCII UTF-8 character makes it really hard to find typos or issues related to the character set, because often they cannot be displayed correctly. Unless it is really necessary we recommend avoid using non ASCII character in a topic.

Embed a unique identifier or the ClientId into the topic

In some cases it is very helpful, when the topic contains a unique identifier of the client the publish is coming from. This helps identifying, who send the message. Another advantage is the enforcement of authorization, so that only a client with the same ClientId as contained in the topic is allowed to publish to that topic. So a client with the id `client1` is allowed to publish to `client1/status`, but not permitted to publish to `client2/status`.

Don't subscribe to #

Sometimes it is necessary to subscribe to all messages, which are transferred over the broker, for example when persisting all of them into a database. This should not be done by using a MQTT client and subscribing to the multi level wildcard. The reason is that often the subscribing client is not able to process the load of messages that is coming its way. Especially if you have a massive throughput. The recommended solution is to implement an extension in the MQTT broker.

Don't forget extensibility

Topics are a flexible concept and there is no need to preallocate them in any kind of way, regardless both the publisher and subscriber need to be aware of the topic. So it is important to think about how they can be extended in case you are adding new features to your product. For example when your smart home solution is extended by some new sensors, it should be possible to add these to your topic tree without changing the whole topic hierarchy.

Use specific topics, instead of general ones

When naming topics it is important not to use them like a queue, for example using only one topic for all messages is an anti pattern. You should use as specific topics as possible. So if you have three sensors in your living room, you should use topics `myhome/livingroom/temperature`, `myhome/livingroom/brightness` and `myhome/livingroom/humidity`, instead of sending all values over `myhome/livingroom`.

Persistent session / Durable connections

When a client connects to a MQTT broker, it needs to create subscriptions for all topics that it is interested in in order to receive messages from the broker. On a reconnect these topics are lost and the client needs to subscribe again. This is the normal behavior with no persistent session. But for constrained clients with limited resources it would be a burden to subscribe again each time they lose the connection. So a persistent session saves all information relevant for the client on the broker. The session is identified by the clientId provided by the client on connection establishment (more details).

So what will be stored in the session?

- Existence of a session, even if there are no subscriptions
- All subscriptions
- All messages in a Quality of Service (QoS) 1 or 2 flow, which are not confirmed by the client
- All new QoS 1 or 2 messages, which the client missed while it was offline
- All received QoS 2 messages, which are not yet confirmed to the client

That means even if the client is offline all the above will be stored by the broker and are available right after the client reconnects.

How to start/end a persistent session?

A persistent session can be requested by the client on connection establishment with the broker. The client can control, if the broker stores the session using the clean Session flag. If the clean session is set to true then the client does not have a persistent session and all information are lost when the client disconnects for any reason. When clean session is set to false, a persistent session is created and it will be preserved until the client requests a clean session again. If there is already a session available then it is used and queued messages will be delivered to the client if available.

Best practices

When you should use a persistent session and when a clean session?

Persistent Session

- A client must get all messages from a certain topic, even if it is offline. The broker should queue the messages for the client and deliver them as soon as the client is online again.
- A client has limited resources and the broker should hold its subscription, so the communication can be restored quickly after it got interrupted.
- The client should resume all QoS 1 and 2 publish messages after a reconnect.

Clean session

- A client is not subscribing, but only publishing messages to topics. It doesn't need any session information to be stored on the broker and publishing messages with QoS 1 and 2 should not be retried.
- A client should explicitly not get messages for the time it is offline.

How long are messages stored on the broker?

An often asked question is how long is a session stored on the broker. The easy answer is until the clients comes back online and receives the message. But what happens if a client does not come online for a long time? The constraint for storing messages is often the memory limit of the operating system. There is no standard way on what to do in this scenario. It totally depends on the use case and the broker.

Retained Messages

A retained message is a normal MQTT message with the retained flag set to true. The broker will store the last retained message and the corresponding QoS for that topic. Each client that subscribes to a topic pattern, which matches the topic of the retained message, will receive the message immediately after subscribing. For each topic only one retained message will be stored by the broker.

The subscribing client can identify if a received message was a retained message or not, because the broker sends out retained messages with the retained flag still set to true. A client can then decide on how to process the message.

So retained messages can help newly subscribed clients to get a status update immediately after subscribing to a topic and don't have to wait until a publishing clients send the next update.

In other words a retained message on a topic is the last known good value, because it doesn't have to be the last value, but it certainly is the last message with the retained flag set to true.

It is important to understand that a retained message has nothing to do with a persistent session of any client. Once a retained message is stored by the broker, the only way to remove it is explained below.

Send a retained message

Sending a retained message from the perspective of a developer is quite simple and straightforward. You just need to set the retained flag of a MQTT publish message to true. Each client library typically provides an easy way to do that.

Delete a retained message

There is also a very simple way for deleting a retained message on a topic: Just send a retained message with a zero byte payload on that topic where the previous retained message should be deleted. The broker deletes the retained message and all new subscribers won't get a retained message for that topic anymore. Often deleting is not necessary, because each new retained message will overwrite the last one.

Why and when you should use Retained Messages?

A retained message makes sense, when newly connected subscribers should receive messages immediately and shouldn't have to wait until a publishing client sends the next message. This is extremely helpful when for status updates of components or devices on individual topics. For example the status of device1 is on the topic myhome/devices/device1/status, a new subscriber to the topic will get the status (online/offline) of the device immediately after subscribing when retained messages are used. The same is true for clients, which send data in intervals, temperature, GPS coordinates and other data. Without retained messages new subscribers are kept in the dark between publish intervals. So using retained messages helps to provide the last good value to a connecting client immediately.

Last Will and Testament

When a client connects to a broker, it may inform the broker that it has a will. This is a message that it wishes the broker to send when the client disconnects unexpectedly. The will message has a topic, QoS and retain status just the same as any other message. EZLogix PLC currently does not support Wills.

When will a broker send the LWT message?

According to the MQTT 3.1.1 specification the broker will distribute the LWT of a client in the following cases:

- An I/O error or network failure is detected by the server.
- The client fails to communicate within the Keep Alive time.
- The client closes the network connection without sending a DISCONNECT packet first.
- The server closes the network connection because of a protocol error.
- We will hear more about the Keep Alive time in the next post.

Best Practices – When should you use LWT?

LWT is ideal for notifying other interested clients about the connection loss. In real world scenarios LWT is often used together with retained messages, in order to store the state of a client on a specific topic. For example after a client has connected to a broker, it will send a retained message to the topic client1/status with the payload “online”. When connecting to the broker, the client sets the LWT message on the same topic to the payload “offline” and marks this LWT message as a retained message. If the client now disconnects ungracefully, the broker will publish the retained message with the content “offline”. This pattern allows for other clients to observe the status of the client on a single topic and due to the retained message even newly connected client now immediately the current status.

8.3 Basic MQTT Setup on EZLogix

The EZLogix PLC MQTT Publish instructions is looked at in *Section 3.3.16*. But before the instruction can be used the MQTT Broker information needs to be configured. To do this please go to **Setup > MQTT Setup....** The needed information for this setup is:

Information Type	Description	Example
Domain Name	This is the broker URL. Used to find your broker that you have configured.	m12.cloudmqtt.com
Port Number	Port number that your broker uses.	16581
Client ID	Individual connection ID. Needs to be different for every client otherwise will encounter problems. Can be random.	ee097f5c-fa36-4929-9414-fad17b3df3bd
User Name	Your configured username for EZLogix connection to broker. Should be different for every client.	
Password	Your configured password for EZLogix connection to broker. Should be different for every client.	

Instruction to setup MQTT:

10. Go to **Setup > MQTT Setup....** You will see the following dialog box appear.

11. Use the Domain Name Lookup with the Domain Name from the broker to find the Broker IP Address.

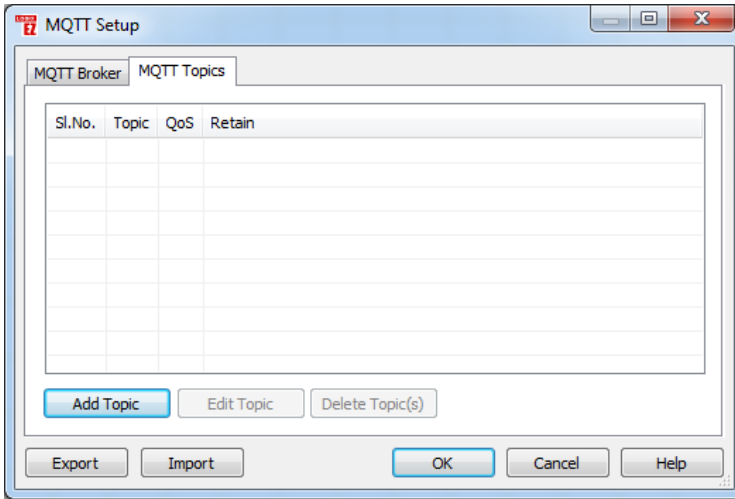
12. Enter the port number from the broker.

13. Select your keep alive interval if wanted. See *section 8.6* for more information.

14. Enter a unique client ID or generate one using the Generate Unique Id button.

15. Enter the user name and password for your broker.

16. Go to the MQTT topics.



17. In the MQTT Topics use the Add Topic button to create the prefixes for your tags. The publish instruction will publish the tagname as a topic but if you want to have more topic information create the prefix here. For example:

Note: After this topic an "/" is appended

Topic: EZLogixPLC/Machine1

TagName: Speed

Published Topic: EZLogixPLC/Machine1/Speed

18. Now in your ladder logic add the IIoT (MQTT) Publish instruction and configure it. For configuration options please see Section 3.3.16.

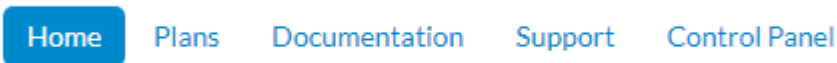
8.4 Broker Setup

The EZLogix PLC can work with any third party broker. It has been tested and used extensively with the CloudMQTT broker. This section will go through some important information about setup of your broker.

CloudMQTT has a free plan for testing purposes. Please see below for setup instructions.

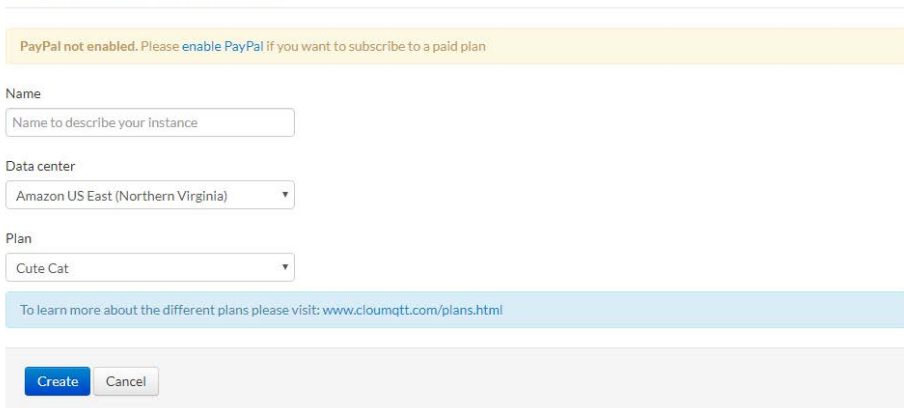
Broker Setup Basics

1. For any broker you can go to their website and create an account. For the CloudMQTT broker you go to <https://www.cloudmqtt.com/>.



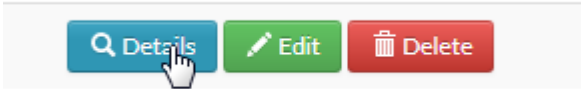
2. Then the plans section will give you information on the different plans available and their cost. The documentation provides information about how MQTT works. Support is the CloudMQTT Tech Support. Finally the Control Panel is what you use to create the MQTT connection.
3. After going to Control Panel, please create an account or login to an account.
4. In the account create a new CloudMQTT Instance.

Create new CloudMQTT Instance

A form for creating a new CloudMQTT instance. At the top, there is a yellow warning banner that says "PayPal not enabled. Please enable PayPal if you want to subscribe to a paid plan". Below this, there are three input fields: "Name" (with placeholder text "Name to describe your instance"), "Data center" (a dropdown menu currently showing "Amazon US East (Northern Virginia)"), and "Plan" (a dropdown menu currently showing "Cute Cat"). Below the form fields is a light blue banner with the text "To learn more about the different plans please visit: www.cloudmqtt.com/plans.html". At the bottom of the form are two buttons: "Create" (in blue) and "Cancel" (in light gray).

5. Enter a Name, select the Data Center and then for the free plan use the Cute Cat plan.

6. Once the Instance is create click on details to find the information needed to subscribe to this broker.



7. The Instance Info is the information that is needed for both the EZLogix Designer Pro and EZ-IIoT Subscriber Utility.

Instance info

Server

User

Password

Port

8. This information provides the details for this connections where:

EZLogix	Instance Info
Domain Name	Server
Port Number	Port
Client ID	N.A.
User Name	User
Password	Password

9. You can also add more users in the Manage Users section. You just need to provide the username and password.

Manage Users

username password

10. Finally you can create ACL rules which govern what each user can access. This allows for management and distribution of topics to the correct people.

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](#)

11. You have now configured your broker and it can be used with the EZLogix PLC and the EZ-IIoT Subscriber Utility.

8.5 EZLogix IIoT (MQTT) Example

This sections shows the creation of an IIoT (MQTT) Publish instruction from start to finish in a project. It requires that the user has created a broker and has broker information.

Used Broker Information:

Information Type	Information
Domain Name	m12.cloudmqtt.com
Port Number	16581
Client ID	Test-ID0001
User Name	TEST
Password	AVG123

1. In a open project go to **Setup > MQTT Setup...**

2. Click on Domain Name Lookup.

3. Enter the domain name and press Lookup. This will find the domain's IP address. Once found press Use Select IP.

4. The Broker IP will now have been entered.
5. Next input the port number (16581).
6. For this example we keep the Keep Alive Interval at 0.
7. Enter the Client ID or generate an Unique one.
8. Finally add your broker username and password.

9. The final result should look something like this.

The screenshot shows the 'MQTT Setup' dialog box with the 'MQTT Broker' tab selected. The fields are filled with the following values:

- Broker IP: 52 . 3 . 184 . 147
- Port Number: 0 (Default: 1883)
- Keep Alive Interval: 16581 Seconds
- Client Id: Client-ID0001
- User Name: TEST
- Password: AVG123

Buttons visible include 'Domain Name Lookup', 'Generate Unique Id', 'Export', 'Import', 'OK', 'Cancel', and 'Help'.

10. Now go to the MQTT Topics. Use the Add Topic to add a topic, for example:

EZLogixPLC/TestTopic

The screenshot shows the 'MQTT Setup' dialog box with the 'MQTT Topics' tab selected. A table is visible with columns 'Sl.No.', 'Topic', 'QoS', and 'Retain'. The 'Add Topic 1' dialog is open, showing the following configuration:

- Topic Name (Maximum 64 char topic name): EZLogixPLC/TestTopic
- QoS: At Most Once (0) At Least Once (1)
- Retain Message:

Buttons visible include 'Add Topic 1', 'Cancel', 'Export', 'Import', 'OK', 'Cancel', and 'Help'.

11. You can also select here the QoS (Quality of Service) and whether the message should be retained.

12. You have now configured your MQTT connection. Next you need to add the IIoT (MQTT) Publish instruction.



13. In the sidebar select the IIoT (MQTT) Publish instruction and add it to your logic. Double click on the instruction to bring up the configuration dialog.

IIoT (MQTT) Publish Instruction

Instruction Details

Broker and Topic

Publish to Broker: 52.3.184.147:16581 MQTT Setup

Topic

EZLogixPLC/TestTopic

Retain Message: No, QoS: At Most Once

Publish

Publish Type: On Rising Edge of Event Tag

Event/Enable Tag

Publish Time-interval: 0 Minute

Publish Status Tag

Status value definitions:

00: Normal operation (No Errors) 02: Connect failure
64: Done 04: Publish failure

Select Tags

For string tags > 40 char, only 40 char would be included in the value.

Decimal Places for Floating Point Tags: 5

Available Tags:

Name	Address	Type
PUBLISH TAG	R1	UNSIGNED_INT_16

>>

<<

Selected Tags: (0/10)

Name	Address	Type

Delete Tag(s)

Move Tag Up

Move Tag Down

OK Cancel Help

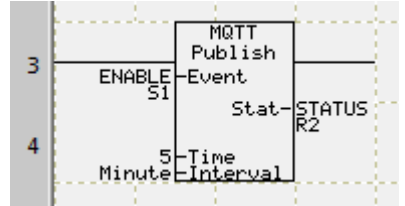
14. Under publish select the type of publishing you would like. For this example it will be at Regular Time Intervals (When Enable Tag is High).

15. Now add an Enable Tag, set the Publish Time-interval to 5 Minutes, and add an Status Tag.

16. Finally move the publish tag to the selected tag area. Final result will look like this:

Where this instruction will publish the Publish Tag to the broker every 5 minutes when the Enable (S1) tag is ON.

The published topic will be:
EZLogixPLC/TestTopic/PUBLISH TAG



Published value will include a timestamp and the current value of PUBLISH TAG (R1).

MQTT Publish Instruction

Instruction Details

Broker and Topic
Publish to Broker: 52.3.184.147:16581 MQTT Setup
Topic: EZLogixPLC/TestTopic
Retain Message: No, QoS: At Most Once

Publish
Publish Type: At Regular Time Intervals (When Enable Tag is High)
Event/Enable Tag: ENABLE
Publish Time-interval: 5 Minute
Publish Status Tag: STATUS
Status value definitions:
00: Normal operation (No Errors) 02: Connect failure
64: Done 04: Publish failure

Select Tags
For string tags > 40 char, only 40 char would be included in the value.

Available Tags:

Name	Address	Type

>> <<

Decimal Places for Floating Point Tags: 5

Selected Tags: (1/10)

Name	Address	Type
PUBLISH TAG	R 1	UNSIGNED_INT_16

Delete Tag(s) Move Tag Up Move Tag Down

OK Cancel Help

8.6 MQTT HIVEMQ Essentials

Most of this section has been taken from <http://www.hivemq.com/blog/mqtt-essentials/> and is their MQTT essentials blog posts. It has been condensed here for to describe the basics of MQTT and how it functions.

Pub/Sub Pattern

As already mentioned the main aspect in pub/sub is the decoupling of publisher and receiver, which can be differentiated in more dimensions:

- Space decoupling: Publisher and subscriber do not need to know each other (by ip address and port for example)
- Time decoupling: Publisher and subscriber do not need to run at the same time.
- Synchronization decoupling: Operations on both components are not halted during publish or receiving

In summary publish/subscribe decouples publisher and receiver of a message, through filtering of the messages it is possible that only certain clients receive certain messages. The decoupling has three dimensions: Space, Time, and Synchronization.

Scalability

Pub/Sub also provides a greater scalability than the traditional client-server approach. This is because operations on the broker can be highly parallelized and processed event-driven. Also often message caching and intelligent routing of messages is decisive for improving the scalability.

Message Filtering

So what's interesting is, how does the broker filter all messages, so each subscriber only gets the messages it is interested in?

Option 1: Subject-based filtering

The filtering is based on a subject or topic, which is part of each message. The receiving client subscribes on the topics it is interested in with the broker and from there on it gets all message based on the subscribed topics. Topics are in general strings with an hierarchical structure, that allow filtering based on a limited number of expression.

Option 2: Content-based filtering

Content-based filtering is as the name already implies, when the broker filters the message based on a specific content filter-language. Therefore clients subscribe to filter queries of

messages they are interested in. A big downside to this is, that the content of the message must be known beforehand and cannot be encrypted or changed easily.

Option 3: Type-based filtering

When using object-oriented languages it is a common practice to filter based on the type/class of the message (event). In this case a subscriber could listen to all messages, which are from type Exception or any subtype of it.

There are some drawbacks to consider. The decoupling of publisher and subscriber, which is the key in pub/sub, brings a few challenges with it. You have to be aware of the structuring of the published data beforehand. In case of subject-based filtering, both publisher and subscriber need to know about the right topics to use. Another aspect is the delivery of message and that a publisher can't assume that somebody is listening to the messages he sends. Therefore it could be the case that a message is not read by any subscriber.

Distinction from Message Queues

So there are many confusions about MQTT, its name and if it is implemented as a message queue or not. We will try to bring light into the dark and explain the differences. In our last post we already pointed out that the name MQTT comes from an IBM product called MQseries and has nothing to do with "message queue". But regardless of the name, what are the differences between MQTT and a traditional message queue?

A message queue stores message until they are consumed

When using message queues, each incoming message will be stored on that queue until it is picked up by any client (often called consumer). Otherwise the message will just be stuck in the queue and waits for getting consumed. It is not possible that message are not processed by any client, like it is in MQTT if nobody subscribes to a topic.

A message will only be consumed by one client

Another big difference is the fact that in a traditional queue a message is processed by only one consumer. So that the load can be distributed between all consumers for a particular queue. In MQTT it is quite the opposite, every subscriber gets the message, if they subscribed to the topic.

Queues are named and must be created explicitly

A queue is far more inflexible than a topic. Before using a queue it has to be created explicitly with a separate command. Only after that it is possible to publish or consume messages. In MQTT topics are extremely flexible and can be created on the fly.

MQTT Connection Information

Below is all basic information that is necessary to connect to a MQTT broker from a MQTT client.

ClientId

The client identifier (short ClientId) is an identifier of each MQTT client connecting to a MQTT broker. As the word identifier already suggests, it should be unique per broker. The broker uses it for identifying the client and the current state of the client. If you don't need a state to be hold by the broker, in MQTT 3.1.1 (current standard) it is also possible to send an empty ClientId, which results in a connection without any state. A condition is that clean session is true, otherwise the connection will be rejected.

Clean Session

The clean session flag indicates the broker, whether the client wants to establish a persistent session or not. A persistent session (Clean Session is false) means, that the broker will store all subscriptions for the client and also all missed messages, when subscribing with Quality of Service (QoS) 1 or 2. If clean session is set to true, the broker won't store anything for the client and will also purge all information from a previous persistent session.

Username/Password

MQTT allows to send a username and password for authenticating the client and also authorization. However, the password is sent in plaintext, if it isn't encrypted or hashed by implementation or TLS is used underneath. We highly recommend to use username and password together with a secure transport of it. In brokers like HiveMQ it is also possible to authenticate clients with an SSL certificate, so no username and password is needed.

Will Message

The will message is part of the last will and testament feature of MQTT. It allows to notify other clients, when a client disconnects ungracefully. A connecting client will provide his will in form of an MQTT message and topic in the CONNECT message. If this clients gets disconnected ungracefully, the broker sends this message on behalf of the client. We will talk about this in detail in an individual post.

Keep Alive

The keep alive is a time interval, the clients commits to by sending regular PING Request messages to the broker. The broker response with PING Response and this mechanism will allow both sides to determine if the other one is still alive and reachable. We'll talk about this in detail in a future post.

Publish Functionality

After a MQTT client is connected to a broker, it can publish messages. MQTT has a topic-based filtering of the messages on the broker, so each message must contain a topic, which will be used by the broker to forward the message to interested clients. Each message typically has a payload which contains the actual data to transmit in byte format. EZLogix PLC MQTT Publish sends the data in basic text with time stamp included. Below is some more information on the message attributes:

Topic Name

A simple string, which is hierarchically structured with forward slashes as delimiters. An example would be “myhome/livingroom/temperature” or “Germany/Munich/Octoberfest/people”.

QoS

A Quality of Service Level (QoS) for this message. The level (0, 1 or 2) determines the guarantee of a message reaching the other end (client or broker).

Retain-Flag

This flag determines if the message will be saved by the broker for the specified topic as last known good value. New clients that subscribe to that topic will receive the last retained message on that topic instantly after subscribing.

Payload

This is the actual content of the message. EZLogix PLC MQTT Publish sends the data in basic text with time stamp included.

Packet Identifier

The packet identifier is a unique identifier between client and broker to identify a message in a message flow. This is only relevant for QoS greater than zero. Setting this MQTT internal identifier is the responsibility of the client library and/or the broker.

DUP flag

The duplicate flag indicates, that this message is a duplicate and is resent because the other end didn't acknowledge the original message. This is only relevant for QoS greater than 0. This resend/duplicate mechanism is typically handled by the MQTT client library or the broker as an implementation detail.

Subscribe Functionality

Publishing messages doesn't make sense if no one ever receives the message, or, in other words, if there are no clients subscribing to any topic. A client needs to send a SUBSCRIBE message to the MQTT broker in order to receive relevant messages. A subscribe message is pretty simple, it just contains a unique packet identifier and a list of subscriptions.

Packet Identifier

The packet identifier is a unique identifier between client and broker to identify a message in a message flow. This is only relevant for QoS greater than zero. Setting this MQTT internal identifier is the responsibility of the client library and/or the broker.

List of Subscriptions

A SUBSCRIBE message can contain an arbitrary number of subscriptions for a client. Each subscription is a pair of a topic and QoS level. The topic in the subscribe message can also contain wildcards, which makes it possible to subscribe to certain topic patterns. If there are overlapping subscriptions for one client, the highest QoS level for that topic wins and will be used by the broker for delivering the message.

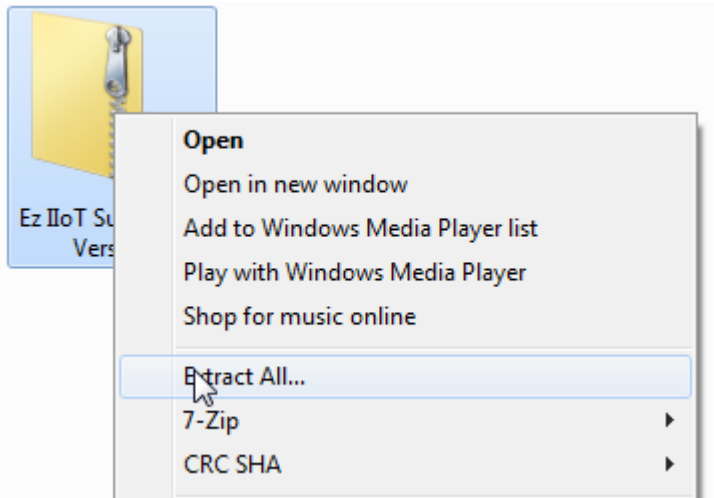
8.7 EZ-IIoT Subscriber Utility

The data EZLogix PLC publishes to the broker is accessible through any third party subscriber utility but EZ Automation has created its own take on this utility. The EZAutomation subscriber utility is developed to make it very easy to see current updated information as well as store any previously published information. This utility will data log any MQTT messages that it sees when subscribed to the broker.

8.7.1 Install EZ-IIoT Subscriber Utility

The EZ-IIoT Subscriber Utility is a separate setup which can be downloaded from www.EZAutomation.com. The EZ-IIoT Subscriber Utility can be installed on any computer that the EZLogix Designer Pro can and at least 2 MB of free space on hard drive for installation. Follow directions below to setup the utility.

1. Download the EZ-IIoT Subscriber Utility ZIP file from the website.



2. Extract the zip folder to the location where you want to place the utility.
3. The utility will now run. Please follow directions below to setup your broker connection.

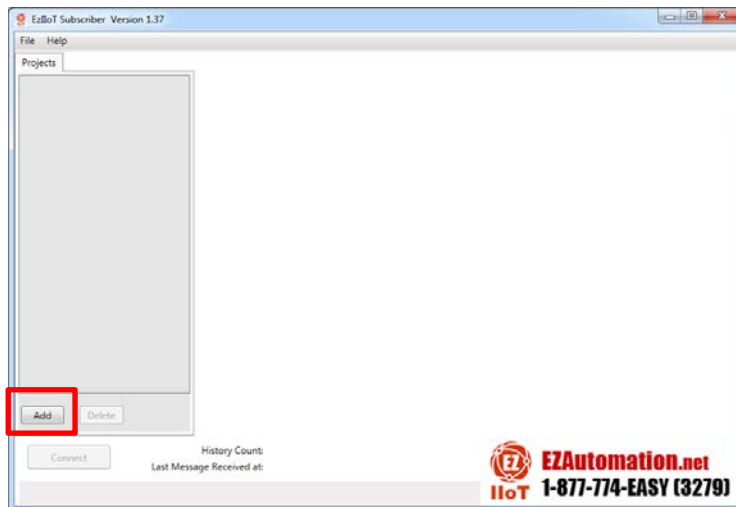
Note: The EZ-IIoT Subscriber Utility requires .NET Framework 4.5 which you might need to install from the Microsoft Windows Website.

8.7.2 EZ-IIoT Subscriber Utility Setup

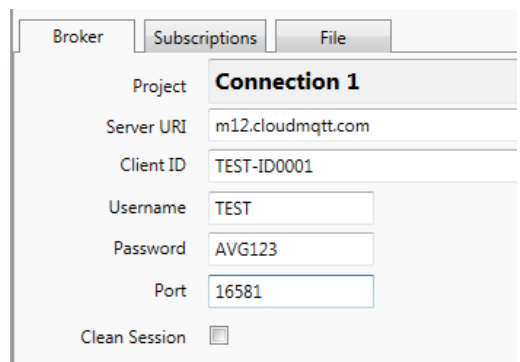
The EZ-IIoT Subscriber Utility is very easy to setup. The only information needed is listed in the table below. To setup the utility please follow the instructions below.

Information Type	Example Information
Domain Name (Server URI)	m12.cloudmqtt.com
Client ID	Test-ID0001
User Name	TEST
Password	AVG123
Port Number	16581

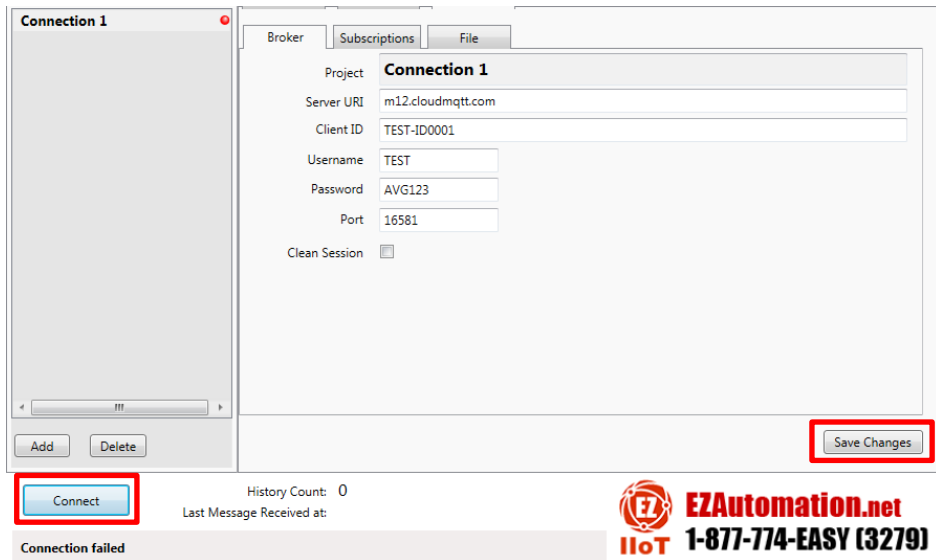
1. Open the EZ-IIoT Subscriber Utility. In the projects area click the “Add” button.



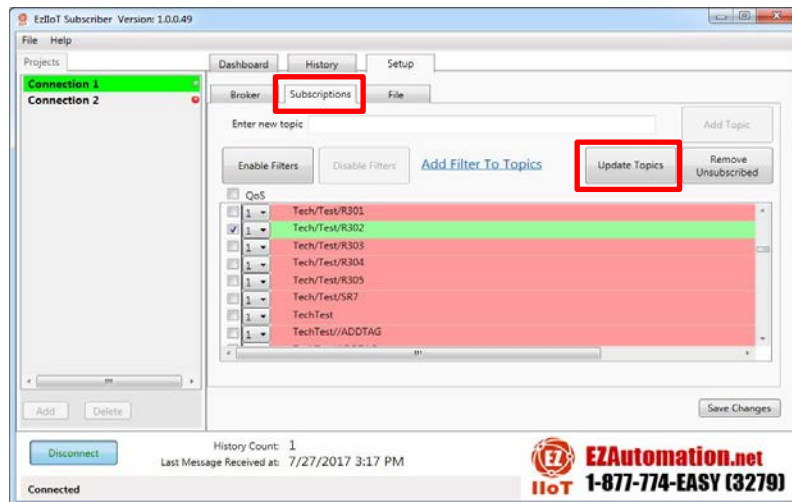
2. In the new connection enter the information from the broker. The example shown uses the example information in the table above. You can also rename the project in the Broker Setup window.



3. Click the “Save Changes”. You will now have the Connect option in the information below. Use the “Connect” button to connect to your broker.



4. As soon as you are connected the Project will turn green. Now in the Setup tab go to the Subscriptions tab. Click the Update Topics to get the topics you have access to. This will only retrieve topics that have been published with the 'Retain Flag' set to true AND have been published at least once. If this is not true you can add any topic you would like. Then select Topics you would like to subscribe to. Once select the topic is subscribed and you will now be updated in the History tab about its value. Please see the next section for the full functionality of the Utility.

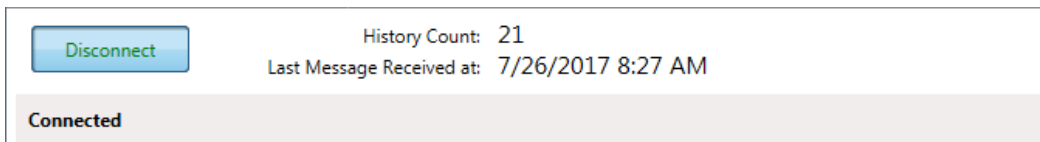


8.7.3 EZ-IIoT Subscriber Utility Functions

The EZ-IIoT Subscriber Utility has 5 tabs total for its full functionality. This section will go through the 5 tabs and list its functionality. There are 2 main tabs (Dashboard, History) and 3 setup tabs (Broker, Subscriptions, and File).

Connection Status

The connection status is visible in all tabs and allows the user to connect and disconnect from the broker. It also lists the current history count and when the last message was received. If there are any errors they will also be listed here.



Projects List

The project list allows switching between all the different connection setups. Only one connection can be connected at a time. New connection can only be added when you are disconnected from the broker. The green light indicates which project/connection is actively connected to a broker.

Use the **Add** and **Delete** buttons to add and delete connections when not connected to broker.

Dashboard Tab

The dashboard is the main view screen for any Project / Connection. It allows the user to have an overview of this broker connection and monitor any important topics.

The screenshot shows the Dashboard Tab interface with the following elements and callouts:

- Tab Navigation:** A box containing 'Dashboard', 'History', and 'Setup' tabs.
- Project/Connections Name:** A box containing 'Project: Connection 1'.
- Topic Information (See Subscription Tab):** A box containing 'Topics Subscribed / Available: 9 / 149'.
- Add Topics to Dashboard:** A button labeled 'Add Topics to Dashboard'.
- Remove All:** A button labeled 'Remove All'.
- Topic Cards:** Two topic cards are shown. The first is titled 'Topic' and the second 'Topic 2'. Each card has a close button (X) in the top right corner.
- Callout for Add Topics:** 'Use this to add important topics to the dashboard to monitor its value and status.'
- Callout for Remove All:** 'Removes all topics from dashboard. Does not unsubscribe.'
- Individual Topic Card Callout:** 'Each individual topic added to dashboard will have its information box. See below for more information.'

The individual topic card shown in the callout has the following details:

- Topic Name: Tech/Test/R300
- Value: 47
- Published: 7/26/2017 8:27 AM
- Received: 7/26/2017 8:27 AM

Dashboard Highlighted Topics

Any topic added to the dashboard will have a box appear where the current status / value can be monitored. This box will list the topic name at the top. The last received value is the value in the middle. Finally it will list the publish time and Utility receive time at the bottom. To eliminate this topic from the dashboard use the X or the Remove All option.

The highlighted topic card shows the following structure:

- Topic Name: Topic
- Value: Value
- Published: Date and Time published
- Received: Date and Time received

Eliminating the topic from the dashboard does not unsubscribe. *Note: Each time a new message is received for this topic it will flash to indicate status change.*

History Tab

The history tab lists all the received values from all subscribed topics. Filters exist to navigate and narrow down information. Also the history can be cleared. The connection status area will list the total count of received values from all topics listed in the history tab. The history can be saved manually but it is also saved automatically (please see **Setup > File** Tab for more information).

The screenshot shows the 'History' tab interface. At the top, there are three tabs: 'Dashboard', 'History', and 'Setup'. A callout box labeled 'Tab Navigation' points to these tabs. Below the tabs, the connection name 'Connection 1' is displayed. On the right side, there are two buttons: 'Clear History' and 'Save History'. Callout boxes explain these buttons: 'Clear History (does not clear saved .csv file)' and 'Manually Save Current History in new .csv file'. On the left side, there are two buttons: 'Enable Filters' and 'Disable Filter'. A callout box explains: 'Enable/Disable Filter Use **Add Filter to Topics** to create Filter (Please see next page for more information)'. In the center, there is a link 'Add Filter To Topics'. Below these elements is a table with the following data:

Unique Id	Received At	Topic	Broker Sent At	Message	QoS	Retained Flag	Dup. Flag	
1246	7/26/2017 9:51:49 AM	Tech/Test/R300	7/26/2017 9:51:46 AM	124	0	NO	NO	
1247	7/26/2017 9:51:49 AM	Tech/Test/R301	7/26/2017 9:51:46 AM	124	0	NO	NO	
1248	7/26/2017 9:51:49 AM	Tech/Test/R302	7/26/2017 9:51:46 AM	124	0	NO	NO	
1249	7/26/2017 9:51:49 AM	Tech/Test/R303	7/26/2017 9:51:46 AM	124	0	NO	NO	
1250	7/26/2017 9:51:49 AM	Tech/Test/R304	7/26/2017 9:51:46 AM	124	0	NO	NO	
1251	7/26/2017 9:51:49 AM	Tech/Test/R305	7/26/2017 9:51:46 AM	124	0	NO	NO	
1252	7/26/2017 9:51:49 AM	Tech/Test/SR7	7/26/2017 9:51:46 AM	1	0	NO	NO	

History Information

Unique ID – Each received message will have a unique ID number per connection which can be used to reference the received message. It can be used to search in the .csv file as well.

Received At – This is the time and date that the message was received by the Utility.

Topic – The subscribed topic name.

Broker Sent At – When the publisher sent the message to the broker. Can be incorrect if publisher (EZLogix) has wrong date and time.

Message – The actual message. The utility is formatted to expect EZLogix format of messages. The EZLogix messages are formatted to include the Time Stamp of when the message was sent and then the message value. The EZLogix publish format is “TimeStamp, Value”. Example below:

Received message: 1501073628, 291

The corresponding history result is:

Broker Sent At: 7/26/2017 12:53:48

Message: 291

QoS – Quality of Service from the publisher. Set on the publisher (EZLogix) side.

Retained Flag – This will tell you if it is a currently published message or if it is a retained message. The message will say “NO” for retained flag if you are subscribed while it is published. Otherwise if the message is set on the publisher side as retained you will receive the message as soon as you subscribe. Please see example below:

Event 1:

Utility: Subscribes

Publisher: Publish Message 1 with Retain Message set

Utility: Message 1 received and has “NO” for retained flag

Event 2:

Publisher: Publish Message 2 with Retain Message set

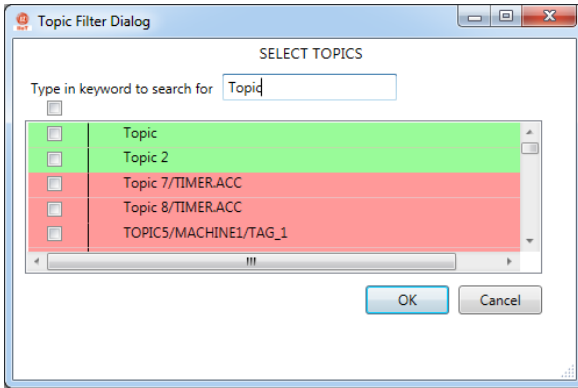
Utility: Subscribes

Utility: Message 2 received and has “YES” for retained flag

Note: Retain flag will not be “YES” unless the message was published before the user subscribed

Dup. Flag – The duplicate flag will be set to “YES” if the message has been received more than once by the Broker.

Topic Filter



The Topic Filter can be used to filter by different topics. Use the keyword selector to search for needed topics. Then select the topics you would like to see when filter is enabled. Click “OK” to finish setting up filter.

On the main screen use enable filter to see only previously selected topics.

Setup Broker

This tab is used to configure the broker information before connecting to the broker. Please see the setup instructions in the previous section for more information.

The screenshot shows a web interface for configuring a broker connection. At the top, there are three tabs: 'Dashboard', 'History', and 'Setup'. The 'Setup' tab is selected. Below the tabs, there are two sub-tabs: 'Broker' and 'File'. The 'Broker' sub-tab is selected. The main content area is titled 'Connection 1' and contains the following fields:

- Project: Connection 1
- Server URI: m12.cloudmqtt.com
- Client ID: TEST-ID0001
- Username: TEST
- Password: AVG123
- Port: 16581
- Clean Session:

At the bottom right, there is a 'Save Changes' button. Red boxes and arrows highlight specific elements:

- 'Tab Navigation' points to the 'Setup' tab.
- 'Setup Navigation' points to the 'Broker' sub-tab.
- 'Needed Broker Information (Please see setup section)' points to the Username and Password fields.
- 'Select this if you would like to **Unsubscribe** from all topics when you **Disconnect** from the Broker' points to the 'Clean Session' checkbox.
- 'Make sure to save changes before connecting' points to the 'Save Changes' button.

Note: You cannot connect with the new settings until you save changes.

Setup Subscriptions

This tab is used to subscribe to different topics. This tab is only available when connected to the broker. You can either add a topic or subscribe to topics that already exist on the broker. Use the filter to narrow down the topics you would like to work with.

The screenshot shows the 'Setup Subscriptions' interface. At the top, there are three tabs: 'Dashboard', 'History', and 'Setup'. Below these are three sub-tabs: 'Broker', 'Subscriptions', and 'File'. A red box labeled 'Tab Navigation' points to the top tabs, and another red box labeled 'Setup Navigation' points to the sub-tabs. Below the sub-tabs is an input field for 'Enter new topic' with an 'Add Topic' button. There are also 'Update Topics' and 'Remove Unsubscribed' buttons. A section contains 'Enable Filters', 'Disable Filters', and a link 'Add Filter To Topics'. Below this is a list of topics. The first topic is 'QoS' with a checked checkbox and a dropdown menu showing '1'. The second topic is 'Tech/Test/R303' with a dropdown menu showing '1' and a 'Delete' button. A red box labeled 'Enable/Disable Filter' points to the 'Enable Filters' and 'Disable Filters' buttons. Another red box labeled 'Use Add Filter to Topics to create Filter (Please see next page for more information)' points to the 'Add Filter To Topics' link. A third red box labeled 'Subscribe all visible topics' points to the checked checkbox next to 'QoS'. A 'Save Changes' button is at the bottom right.

How to Subscribe to a Topic

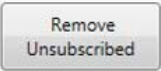
This close-up shows the topic list. The first row is 'QoS' with a checked checkbox. The second row is 'Tech/Test/R300' with a checked checkbox and a dropdown menu showing '1'.

To subscribe just check the box next to the Topic you would like to subscribe to. You can change the Quality of Service (QoS) for communication between utility and Broker for that topic at any time by using the dropdown (QoS of 1 or 0 allowed). Also you can subscribe to all visible topics by using the check box next to QoS.



Update Topics

The update topics will download all topics that exist as retained messages on the broker. Only the topics that you have permission to see will be downloaded. You can also add any topics you would like at any time. This will only retrieve topics that have been published with the 'Retain Flag' set to true AND have been published at least once.



Remove Unsubscribed

The remove unsubscribed option will delete all unsubscribed topics currently visible in the Subscription window.



Delete

You can delete individual topics by right clicking on topic and selecting the delete option.

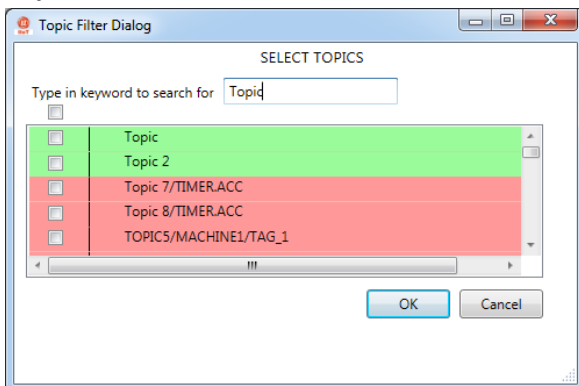
Add Topic



You can at any point add a topic to subscribe to by typing in the topic and pressing Add Topic.

Note: you will need to do this for any topic which does not have a retained flag since the update topics will not populate the list with these.

Topic Filter



The Topic Filter can be used to filter by different topics. Use the keyword selector to search for needed topics. Then select the topics you would like to see when filter is enabled. Click "OK" to finish setting up filter.

On the main screen use enable filter to see only previously selected topics.

Setup File

This tab is used to configure how the Utility will save the messages it has received. Here you can name the save file and change the save folder. You can also configure conditions of saving and when a new file is created.

The screenshot shows the 'Setup File' configuration window for 'Connection 1'. The window has a top navigation bar with 'Dashboard', 'History', and 'Setup' tabs. Below this is a sub-navigation bar with 'Broker', 'Subscriptions', and 'File' tabs. The main configuration area is divided into several sections:

- File:** Contains 'Base Name' (set to 'TopicData') and 'Folder Name' (set to 'C:\'). A 'Browse' button is next to the folder name field.
- Saving Action:** Has two radio buttons: 'Automatically' (selected) and 'Manually'.
- Save To File:** Contains two checked checkboxes:
 - 'Append -YYMMDD-HHMM to Base Name when a new file is created.'
 - 'Create a new file after filesize exceeds' followed by a text box containing '1000' and the unit 'Kbytes'.

Red callout boxes provide additional information:

- 'Tab Navigation' points to the top navigation bar.
- 'Setup Navigation' points to the sub-navigation bar.
- 'Use this to set the name of the .csv and where it will be saved' points to the 'File' section.
- 'Select whether history is saved automatically or you need to save manually' points to the 'Saving Action' section.
- 'These settings are used when Saving Action is set to Automatic. Use these settings to set when a new file is created and how it will be named.' points to the 'Save To File' section.
- 'Make sure to save changes since changes are not implemented till they are saved.' points to the 'Save Changes' button at the bottom right.

Note: The newest data will always be saved in the Base Name .csv file. If new files are created then data is either saved in files with the date and time appended. Or if that format is not used the oldest files will be in "Base Name1.csv", second oldest in "Base Name2.csv". Also if the Base Name is open in excel, write is not possible so a new file with name "Base Name_.csv will be created.

8.7.4 EZ-IIoT Subscriber Utility Best Practices

This section will mention some common best practices when using the EZ-IIoT Subscriber Utility.

Utility Use

Recommended uses of this utility (can be used for multiple purposes at same time):

- Monitor tags – This utility can be used to monitor about 4-10 tags from the dashboard.
- Data Log – When this utility is subscribed it can be used to data log tag values for later analysis. *Note: it is stored as a .csv file.*
- Check Status – This utility can also be used to just check status of machine periodically by subscribing to see current status.
- Troubleshoot – This utility can also be used to see tag values for off-site troubleshooting capability.

CSV Files

When looking at saved history (data logging) in the CSV files the best way to view is to create a copy and then view in excel. If the CSV is open in excel the utility can write to it and will create a new file. Also note the oldest data will have a unique ID of 0 and the newest will have the highest value unique ID.

Client ID

Please make sure to use different Client IDs for each subscriber. If the same client ID is used for multiple subscribers only 1 will ever be able to connect to the Broker at a time. If the client IDs are different all subscribers up to your broker limit can connect to the Broker at the same time.

Username and Password

Each username and password can be limited to only certain topics thereby allowing users specific access to needed information. Therefore it is best to create a different username and password for each user. A username and password can be used in multiple locations to connect at the same time but it is not recommended.

EZ-IIoT Subscriber Utility Acceptable Data Format

The EZLogix publish format is “TimeStamp, Value”. The EZ-IIoT Subscriber Utility expects data in this format. Example below:

Received message: 1501073628, 291

The corresponding history result is:

Broker Sent At: 7/26/2017 12:53:48

Message: 291