Critical manufacturing
an ASM PT company

Connect IoT - MQTT Tutorial

12.0.1

November 2026

DOCUMENT ACCESS

Public

DISCLAIMER

The contents of this document are under copyright of Critical Manufacturing S.A. it is released on condition that it shall not be copied in whole, in part or otherwise reproduced (whether by photographic, or any other method) and the contents therefore shall not be divulged to any person other than that of the addressee (save to other authorized offices of his organization having need to know such contents, for the purpose for which disclosure is made) without prior written consent of submitting company.

12.0

Copy Page

Connect IoT - MQTT Tutorial

Overview

MQTT has no equipment server. Sensors and machines publish messages to a broker, each on a topic such as `hall2/machine-3/temperature`, and the broker delivers each message to every client subscribed to a matching topic. The MQTT driver is one such client: it does not host a broker, so one is required. The driver maps the whole protocol onto properties:

Definition entry	On the broker	Workflow task
Readable property	A subscription to its topic	Carried by an event
Writable property	A publish to its topic	`Set Equipment Properties`
Event	Fires when a message arrives on a trigger topic among its properties	`On Equipment Event`
Command	None: the protocol has no commands	-

Table 1: How the definition maps to the MQTT broker

This tutorial is for integrators who completed Connect IoT - Basic Configuration Tutorial and know the Control Flow designer. You build the integration of a hall for a **Resource**, such as `Hall-2`: the controller logs every temperature reading of the hall with the latest humidity, and publishes a setpoint of `180` to machine 1 whenever its temperature goes above `185`. At the end, the **Automation Manager** console shows each reading, and an MQTT client subscribed to the setpoint topic receives the setpoint.

> **Info**
>
> For every protocol parameter and extended data field of the driver, see Connect IoT - MQTT Driver.

Before You Start

- Complete Connect IoT - Basic Configuration Tutorial and have an **Automation Manager** that runs in console mode. For the Control Flow designer and its expressions, see Connect IoT - Control Flow Configuration Tutorial and Control Flow Expressions.
- Have an MQTT broker reachable from the machine where the **Automation Manager** runs, and an MQTT client that can publish and subscribe, to play the sensors and machine 1. The Test Orchestrator MQTT simulator can stand in for both: it starts an embedded broker, publishes with `Publish`, and waits for the setpoint with `WaitForMessage`. To run it against your MES, see Connecting to a Live MES ↗. The Test Orchestrator is a .NET test framework, currently in alpha.

Sequence of Steps

Step 1: Create the Automation Protocol and Set the Broker Parameters

Create an **Automation Protocol**, such as `MQTT`, for the package `@criticalmanufacturing/connect-iot-driver-mqtt`, as in How to: Create an Automation Protocol. The parameters that matter for this exercise are the following. You set the values for this broker in the controller, in Step 3.

Parameter	Default	Effect
`address`	Empty	The host of the broker. Empty means `localhost`.
`port`	`1883`	The port of the broker. `8883` is usual for MQTTS.
`protocol`	`MQTT`	`MQTTS` connects over TLS, with `certificate`, `privateKey`, and `certificateAuthority`.
`protocolVersion`	`3.1.1`	The MQTT version of the connection, `3.1.1` or `5.0`.

Parameter	Default	Effect
`user`, `password`	Empty	The broker credentials. Empty connects without authentication.
`byteOrder`	`BigEndian`	How the bytes of a binary numeric payload are read. Text payloads are not affected.

Table 2: Broker parameters of the MQTT protocol

`pollInterval` is not used by the driver: to report on a timer, set the `triggerMode` of an event to `OnInterval`. The QoS and the retain flag are not protocol parameters: each property carries its own, so two topics of one connection can use different levels.

The driver connects as a clean session, so the broker keeps nothing for it while it is away. It subscribes the topics of the readable properties only when the setup of the connection completes: a message published before then is lost, unless the broker retained it.

Step 2: Create the Driver Definition with the Topics

The MQTT package ships no driver definition. Declare each topic the controller reads or writes as a property, with the topic in **Topic Name** (the Property Device ID), and each reaction as an event.

1. Create an **Automation Driver Definition**, such as `Hall 2 MQTT`, with the **Automation Protocol** `MQTT` and the **Entity Type** `Resource`, as in Create Automation Driver Definition.
2. Add the properties. The **Protocol Data Type** `String` reads a text payload, and the **Type** is the value the workflow receives: a text such as `21.4` becomes a `Decimal`.

Name	Topic Name (Property Device ID)	Type	Protocol Data Type	Access	Extend data
`Temperature`	`hall2/+/temperatur`	`Decimal`	`String`	Readable	`qos 1`

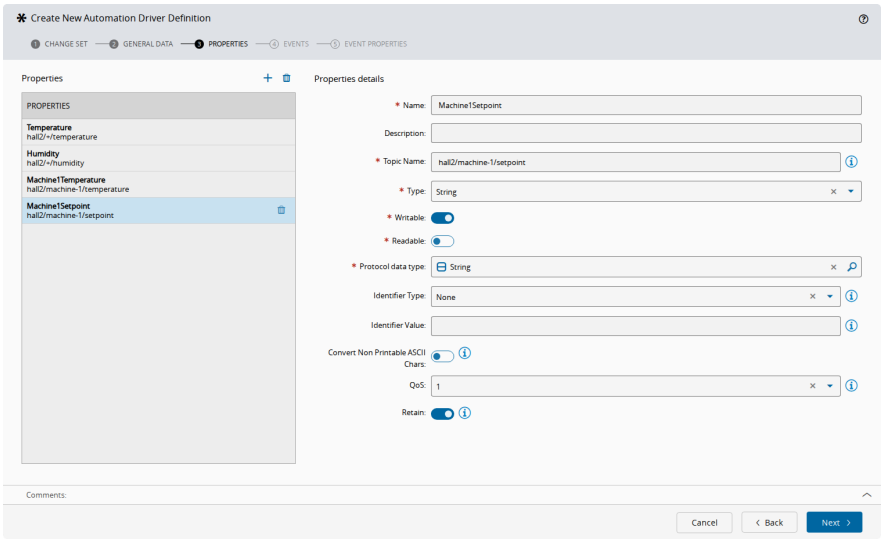
Name	Topic Name (Property Device ID) `e`	Type	Protocol Data Type	Access	Extend data
`Humidity`	`hall2/+/humidity`	`Decimal`	`String`	Readable	`qos` `1`
`Machine1Temperature`	`hall2/machine-1/temperature`	`Decimal`	`String`	Readable	`qos` `1`
`Machine1Setpoint`	`hall2/machine-1/setpoint`	`String`	`String`	Writable, not readable	`qos` `1`, `retain` `true`

Table 3: Properties of the MQTT driver definition

3. Add two events with the **Event Trigger Mode** (extended data `triggerMode`) `Always`, the default.

Event	Event properties
`SensorReading`	`Temperature` with `isTrigger` `true`, `Humidity` with `isTrigger` `false`
`Machine1Reading`	`Machine1Temperature` with `isTrigger` `true`

Table 4: Events of the MQTT driver definition



4. Select **Create**.

A message on `hall2/machine-3/temperature` matches `hall2/+/temperature` and fires `SensorReading`, because `Temperature` is a trigger and `triggerMode` `Always` fires on each qualifying message. A humidity message alone fires nothing, and `Humidity` carries the latest humidity of any sensor, not necessarily the one whose temperature fired the event. With `OnChange`, the event fires only when the value differs from the last one, and with `OnInterval`, every `interval` milliseconds with the latest values.

Write wildcards with `+` for one level and `/#` for everything below, such as `hall2/#`. A topic that is `#` alone, or that starts with `$`, never matches a received message. A wildcard property suits a state, where the latest value is what matters: when messages on several matching topics arrive within a few milliseconds, an occurrence can carry the value of a later message. When every message must be handled with its own value, declare one property and one event per topic, or have the publisher send one JSON payload per message and read it with the **Type** `Object`. The topic a wildcard message came on is in the original value of the property, `{ "topic": "<topic>", "value": <payload> }`: select the **Output Type** `RawValue` on the event to receive it.

`Machine1Setpoint` is not readable, so the driver does not subscribe to it. A readable topic that the controller also publishes to comes back as a received message and fires the events it belongs to. The setpoint is a state that a client connecting later needs, so it is published with `retain`: the broker keeps the last value of the topic and hands it to each new subscriber. Leave `retain` off for events, such as a count or an alarm occurrence, because a retained event is replayed to each new subscriber as if it had just happened.

Step 3: Create the Controller and Connect It to the Broker

1. Create an **Automation Controller**, such as `Hall 2 Controller`, with the **Scope** `Connect IoT`, the **Workflow Type** `Control Flow`, the **Entity Type** `Resource`, and one driver definition, `Broker`, with `Hall 2 MQTT`. See Create Automation Controller.
2. On the **Setup** page, open the settings of the `Equipment Configuration` task and, on the **Communication Parameters** tab, set `address` to the host of your broker, such as `localhost`, and `port`, `user`, and `password` as the broker requires. Keep **Automatic Setup** enabled on the `Driver Event` task.

The driver receives the communication parameters when it connects. After you change a parameter of a connection that is already `Communicating`, restart the controller instance.

Step 4: Log Every Reading of the Hall

Create a Control Flow page, such as `Hall Readings`, and add the following blocks for the driver `Broker`:

Block	Settings to set	Placement
`On Equipment Event`	**On Equipment Event** `SensorReading`	Trigger
`Log Message`	**Mode** `RawText`; **Default Message** `Temperature {{ $<event block>.Temperature }}, humidity {{ $<event block>.Humidity }}`	Inside the handler of the trigger

Table 5: Blocks that log the readings

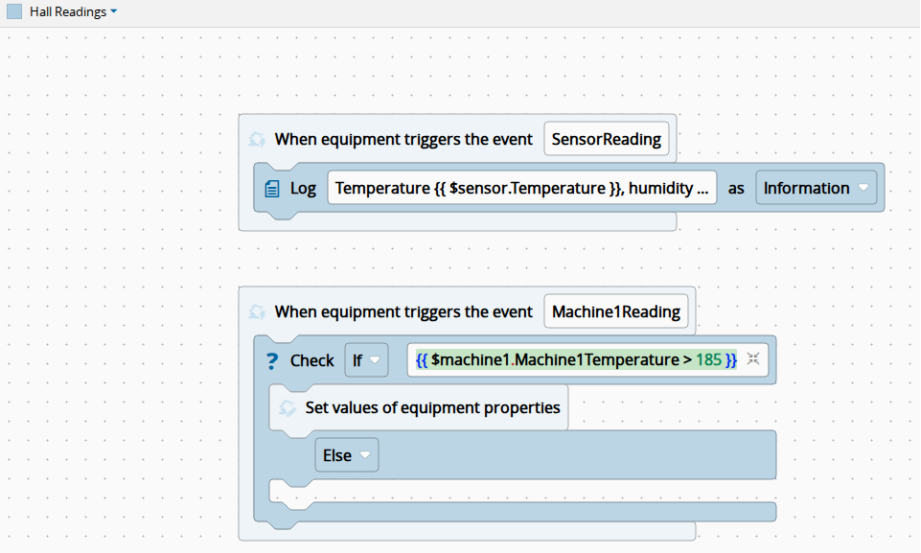
Replace `<event block>` with the block name of the `On Equipment Event` task. Inside `{{ }}`, typing `$` suggests the names of the preceding blocks. In Control Flow, `Log Message` evaluates the expressions of its **Default Message** when it runs, so one block logs both values.

Step 5: Publish a Setpoint When Machine 1 Runs Too Hot

Add the following blocks to the `Hall Readings` page:

Block	Settings to set	Placement
`On Equipment Event`	**On Equipment Event** `Machine1Reading`	Trigger
`Condition`	`If` condition `{{ $<event block>.Machine1Temperature > 185 }}`, where `<event block>` is the name of the `Machine1Reading` block	Inside the handler of the trigger
`Set Equipment Properties`	Input `Machine1Setpoint` with the value `180`	Inside the `If` branch

Table 6: Blocks that publish the setpoint



Publishing is writing a property: `Set Equipment Properties` publishes `180` to the topic of `Machine1Setpoint`, converted to its protocol data type, with its `qos` and `retain`. Keep `identifierType` `None` on a property that is only published. With `Token` or `RegularExpression`, a value that does not match `identifierValue` is not published, and the write still reports success.

Do not publish or subscribe with `Send Notification to Driver` or `Subscribe in Driver`. Their subjects are the channel between the controller and the driver process, not MQTT topics: with no readable property, the connection reaches

`Communicating` while the broker sees no subscription. The subjects that the driver defines for itself, `connect.iot.driver.template.*`, are deprecated.

Step 6: Connect the Controller and Start It

Connect the controller to the **Resource** `Hall-2` and the **Automation Manager**, with the same **Resource** for the driver `Broker`, as in How to Create a Controller Instance. Start the **Automation Manager** in console mode.

On the Connect IoT page, the driver instance reaches `Communicating`. A broker that lists its subscriptions shows the three readable topics and not the setpoint topic.

Validate the Result

1. With the MQTT client, subscribe to `hall2/machine-1/setpoint`.
2. Publish `48` on `hall2/machine-3/humidity`. Nothing is logged: `Humidity` is not a trigger.
3. Publish `21.4` on `hall2/machine-3/temperature`. The console shows `Temperature 21.4, humidity 48`.
4. Publish `182.4` on `hall2/machine-1/temperature`. The console logs the reading, because the topic also matches `hall2/+/temperature`, and nothing is published on the setpoint topic.
5. Publish `186.1` on `hall2/machine-1/temperature`. The client receives `180` on `hall2/machine-1/setpoint`.
6. Disconnect the client and subscribe again to `hall2/machine-1/setpoint`. The broker hands it the retained `180` at once.
7. Stop the broker for a few seconds and start it again. The driver instance moves to `Disconnected`, retries after each **Reconnect Interval** of the `Equipment Configuration` task, and passes through `Disconnected` on every failed attempt. When the broker is back, the instance returns to `Communicating` and subscribes again. Publish a temperature to confirm that readings are logged again.

Next Steps

To report each reading to MES, continue with Connect IoT - Talking to MES Tutorial. A handler that reacts to `Disconnected` on the `OnCommunicationChange` event runs on every retry of an outage: keep a flag that is set when the loss is reported and cleared on `Communicating`, and report the loss only when the flag is clear. To keep such a flag, see Connect IoT - Timing and State Tutorial. For other protocols, see Connect IoT - OPC UA Tutorial and Connect IoT - Raw File and CSV Tutorial, and for symptoms such

as a connection that is `Communicating` with nothing subscribed, see Troubleshooting by Symptom.

Legal Information

Disclaimer

The information contained in this document represents the current view of Critical Manufacturing on the issues discussed as of the date of publication. Because Critical Manufacturing must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Critical Manufacturing, and Critical Manufacturing cannot guarantee the accuracy of any information presented after the date of publication. This document is for informational purposes only.

Critical Manufacturing makes no warranties, express, implied or statutory, as to the information herein contained.

Confidentiality Notice

All materials and information included herein are being provided by Critical Manufacturing to its Customer solely for Customer internal use for its business purposes. Critical Manufacturing retains all rights, titles, interests in and copyrights to the materials and information herein. The materials and information contained herein constitute confidential information of Critical Manufacturing and the Customer must not disclose or transfer by any means any of these materials or information, whether total or partial, to any third party without the prior explicit consent by Critical Manufacturing.

Copyright Information

All title and copyrights in and to the Software (including but not limited to any source code, binaries, designs, specifications, models, documents, layouts, images, photographs, animations, video, audio, music, text incorporated into the Software), the accompanying printed materials, and any copies of the Software, and any trademarks or service marks of Critical Manufacturing are owned by Critical Manufacturing unless explicitly stated otherwise. All title and intellectual property rights in and to the content that may be accessed through use of the Software is the property of the respective content owner and is protected by applicable copyright or other intellectual property laws and treaties.

Trademark Information

Critical Manufacturing is a registered trademark of Critical Manufacturing.

All other trademarks are property of their respective owners.

Copyright © 2023 Critical Manufacturing. All rights reserved.