MQTT

Introduction

MAS 25.1 introduced MQTT connectivity from and to an MQTT broker. This document will explain how our platform communicates with other systems in an installation. We expect the reader to have basic knowledge of how this protocol works. If you do not know the basics, please read this page before reading further: MQTT Protocol Explained: Ultimate Guide for IoT Beginners .

Our platform supports three types of messages: data logs, alert lists and system logs. Each type can be enabled or disabled based on the customers wishes.

Data logging

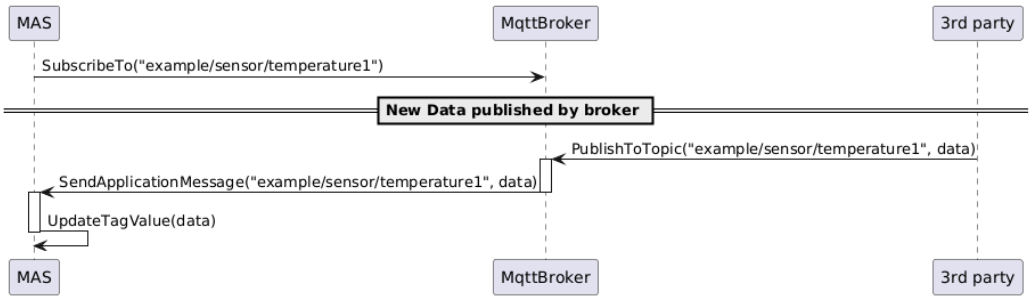
We support two different core concepts of data logging:

- Data **published** by 3rd parties: this group contains all the data that is pushed to the broker by 3rd parties and will eventually be sent to MAS.
- Data **subscribed** to by 3rd parties: this group contains all the data that is published from MAS to the broker. This can be data points, alerts and system logs.

Publish

The data flow for data published by 3rd parties is quite simple. Our engineers configure MAS to connect a tag to a topic and a JSON path. On system start-up, the MQTT service will run a client and subscribe to all configured topics. This way, each time some JSON data is published by a 3rd party, MAS will get a message from the broker and will proces that message. It will look for data in the configured JSON path and update the tag accordingly.

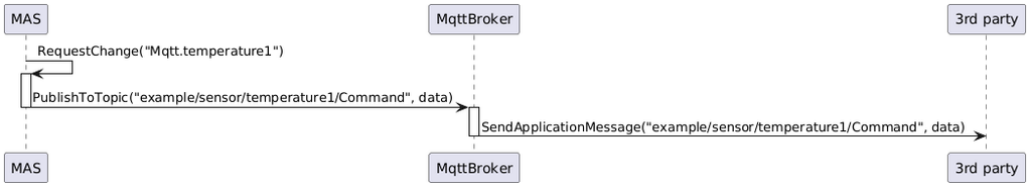
The following sequence diagram describes this proces:



Sequence diagram for data published by 3rd parties

Our platform can also make change requests to topics if a 3rd party listens to change requests. For example, if our platform calculates a suggested set-point for a sensor, it will publish that set-point on the so-called “Command topic” using the same JSON path that is configured in the example above. The default topic is “<topic name>/Command”, but is also custom configurable. It is the responsibility of the 3rd party to listen to the change request and to act on it.

The following sequence diagram describes this proces:

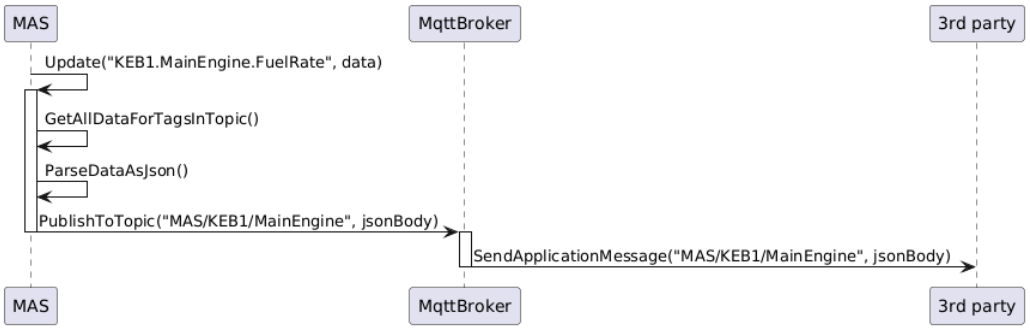


Sequence diagram of a change request initiated by MAS

Subscribe

3rd parties can subscribe to topics that are published by MAS. Not all tags are published to the broker, this is the responsibility of the project engineer to make a selection based on the customer's desires.
Every time a tag receives an update, it will automatically publish the changed value to the broker on the configured topic and JSON path.
If multiple tags are assigned to that topic, it will also send the latest values of those tags in the body. In the case when multiple tags are configured for a topic and the first publish happens when some tags still have no data, it will output the default value for that tag with the **HasValue** flag to **false** (see MQTT | JSON format for more info).
The messages that are published to each topic are retained.

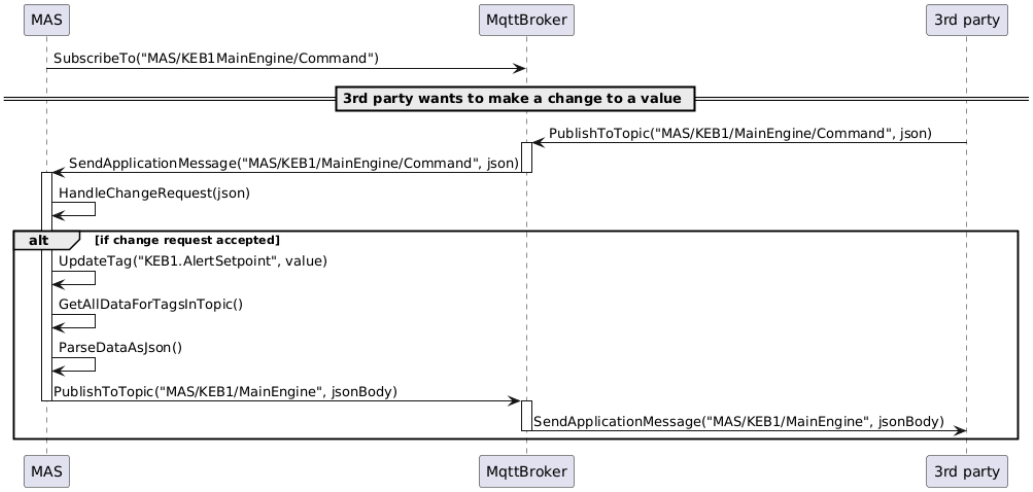
The following sequence diagram describes this proces:



Sequence diagram for data published by MAS

It is also possible for 3rd parties to send data to topics that will trigger a change request in MAS. For each topic that is published by MAS, it creates a subscription on the broker for “<topic>/Command”. 3rd parties can send the value they want to that topic in the same JSON format as it is produced on “<topic>”. MAS will either accept the change request or deny it.
If it is accepted, it will automatically publish the new data to the broker on “<topic>”.
If it is rejected, the request will be ignored and no new data will be published.

The following sequence diagram describes this proces:



Sequence diagram for a change request initiated by a 3rd party

Assume that the payload of the data looks like this for topic “MAS/KEB1/MainEngine“:

```
{
  "Cylinder1": {
    "Temperature": {
      "Value": 67.12,
      "HasValue": true,
      "IsValid": true,
      "TimeStamp": "2025-01-21T08:49:03.6735253Z"
    },
    "AlertSetpoint": {
      "Value": 80,
      "HasValue": true,
      "IsValid": true,
      "TimeStamp": "2025-01-21T08:49:03.6735253Z"
    }
  }
}
```

If a 3rd party wants to change the value of TestBool to false, it can send one of the following two JSONs as payload to topic “MAS/KEB1/MainEngine/Command“:

```
{
  "Cylinder1": {
    "AlertSetpoint": {
      "Value": 100
    }
  }
}
```

or

```
{
  "Cylinder1": {
    "AlertSetpoint": 100
  }
}
```

JSON format

The format that we provide the data in is JSON. Beside the updated value, we also send three other key-value pairs: **IsValid** indicates if the value is between the expected boundaries, **HasValue** indicates if the value had a producer and **TimeStamp** is the date and time when the tag received the updated value in ISO 8601 format.

An example of the payload of a publish by our platform is:

```
{
  "Value": 3,
  "HasValue": true,
  "IsValid": true,
  "TimeStamp": "2025-01-21T08:49:03.6736026Z"
}
```

Combining tags to a single topic

It is also possible to send multiple tags to a single topic. The project engineer will configure the same topic for multiple tags but each with a different JSON path resulting in a bigger JSON. If one tag receives an update, it will also send the latest values of all other tags that belong to the updated topic.

An example of the payload of a topic by our platform with multiple tags is:

```
{
  "TestBool": {
    "Value": true,
    "HasValue": true,
    "IsValid": true,
    "TimeStamp": "2025-01-21T08:49:03.6735253Z"
  },
  "TestInt32": {
    "Value": 3,
    "HasValue": true,
    "IsValid": true,
    "TimeStamp": "2025-01-21T08:49:03.6736026Z"
  },
  "TestDouble": {
    "Value": 1.5,
    "HasValue": true,
    "IsValid": true,
    "TimeStamp": "2025-01-21T08:49:03.6736345Z"
  },
  "TestDateTime": {
    "Value": "1970-01-01T00:00:03Z",
    "HasValue": true,
    "IsValid": true,
    "TimeStamp": "2025-01-21T08:49:03.6736831Z"
  },
  "TestTimeSpan": {
    "Value": "00:00:03",
    "HasValue": true,
    "IsValid": true,
    "TimeStamp": "2025-01-21T08:49:03.6737229Z"
  },
  "TestIpAddress": {
    "Value": "0.0.0.3",
    "HasValue": true,
    "IsValid": true,
    "TimeStamp": "2025-01-21T08:49:03.6737572Z"
```

```
  }
}
```

Alert Lists

In MAS, each location has its own alert list. This list can contain the same alerts as other locations but in a different order. Alerts related to the location appear with a higher priority than alerts from other locations.
Each location will publish its own active alert list to its own topic. This topic has the following format “MAS/<location name>/AlertList“ where <location name> is one of the locations on the ship, such as “Bridge“ or “Engine Room“.

Each time the alert list updates, it will wait a configurable amount of time for more updates and publishes the latest state of the alert list to the broker. This prevents our platform from spamming the broker with alert list updates.

JSON format

The alert list will be published as a JSON array where each element has the following format:

```
{
    "FailureType": "H",
    "OnDelay": "00:00:05",
    "WatchLevel": "90",
    "Severity": "Alarm",
    "Code": "T002-H",
    "VdrId": 0,
    "Description": "Tank TANK2 High Level",
    "CallGeaWhenActivated": false,
    "InvalidAlertPrefix": "",
    "InvalidAlertPrefixForSmallScreenInterface": "",
    "IntendedOperatorResponse": "Please check the operation manual",
    "DisallowInhibit": false,
    "CheckedDuringCommissioning": false,
    "ApprovedByCustomer": false,
    "ApprovedByClass": false,
    "LockTag": "",
    "LockOperator": "None",
    "LockLevel": "",
    "AlertState": {
      "Watch": false,
      "Lock": false,
      "Bound": true,
      "Valid": true,
      "State": false,
      "Active": true,
      "Acknowledged": false,
      "Inhibited": false,
      "Escalated": false,
      "LastStateChange": "2025-01-27T15:31:44.9117862Z",
      "FirstOccurrence": "2025-01-27T15:31:44.2698311Z",
      "OnDelaySecondsRemaining": 65535,
      "OnEscalationSecondsRemaining": 65535,
      "InvokeAfterActivation": false,
      "Name": "AlertState"
    },
    "Name": "H"
}
```

System logging

Each MAS pc has it system log. These are messages that contain information about internal processes that happen on our platform. Each system log is published to the “MAS/SystemLog” topic, which means that it is the responsibility of the subscriber to keep track of the total system log history.

System logs are processed and batched per host. This means that we cannot guarantee that the order of the messages is chronological.

JSON format

The system log message contains a single JSON object that will have the following format:


```
{
  "Id": 0,
  "LoggedToCdp": false,
  "EventTime": "2025-01-28T08:25:46.1665856Z",
  "Host": {
    "Descriptor": "ServiceHost",
    "Instance": "PHW2",
    "ServiceType": "AA",
    "Priority": 0
  },
  "Service": {
    "Descriptor": "WagoCodesys35Adapter",
    "Instance": "KEB2",
    "ServiceType": "AP",
    "Priority": 1
  },
  "Severity": "Warning",
  "Message": "WagoCodesys35FieldbusMaster.Work<KEB2>: Exception caught during execution of state machine: BadConnectionClosed",
  "Details": "Opc.Ua.ServiceResultException: BadConnectionClosed\n   at Opc.Ua.Bindings.UaSCUaBinaryClientChannel.BeginSendRequest(IServiceRequest request, Int32 timeout, AsyncCallback callback, Object state)\n   at Opc.Ua.SessionClient.CreateSession(RequestHeader requestHeader, ApplicationDescription clientDescription, String serverUri, String endpointUrl, String sessionName, Byte[] clientNonce, Byte[] clientCertificate, Double requestedSessionTimeout, UInt32 maxResponseMessageSize, NodeId& sessionId, NodeId& authenticationToken, Double& revisedSessionTimeout, Byte[]& serverNonce, Byte[]& serverCertificate, EndpointDescriptionCollection& serverEndpoints, SignedSoftwareCertificateCollection& serverSoftwareCertificates, SignatureData& serverSignature, UInt32& maxRequestMessageSize)\n   at Opc.Ua.Client.Session.Open(String sessionName, UInt32 sessionTimeout, IUserIdentity identity, IList`1 preferredLocales, Boolean checkDomain)\n   at Chameleon.Automation.Core.Services.OpcUaClient.OpcUaClientFieldbusController.CreateTargetModel(OpcUaClientConfigurationView configurationView, IPingTargetParameters targetParameters) in C:\\data\\Chameleon.Core\\Chameleon.Automation.Core\\Services\\OpcUaClient\\OpcUaClientFieldbusController.cs:line 118\n   at Chameleon.Automation.Core.Utility.RedundantFieldbus.StateMachine.FieldbusStateMachine`3.SafeInvokeCreateTargetModel(Transition transition) in C:\\data\\Chameleon.Core\\Chameleon.Automation.Core\\Utility\\RedundantFieldbus\\StateMachine\\FieldbusStateMachine.cs:line 276\n--- End of stack trace from previous location ---\n   at Chameleon.Automation.Core.Utility.RedundantFieldbus.StateMachine.FieldbusStateMachine`3.Context.ThrowExceptionIfNeeded() in C:\\data\\Chameleon.Core\\Chameleon.Automation.Core\\Utility\\RedundantFieldbus\\StateMachine\\FieldbusStateMachine.cs:line 64\n   at Chameleon.Automation.Core.Utility.RedundantFieldbus.StateMachine.FieldbusStateMachine`3.Kick(IConfigurationProvider`1 configurationProvider, ITargetDetectorStateMachineInterface`1 targetDetector, IStateProvider stateProvider, JoinableTaskFactory joinableTaskFactory, IFieldbusControllerImplementation`3 controller, CancellationToken cancellationToken) in
```

```
C:\\data\\Chameleon.Core\\Chameleon.Automation.Core\\Utility\\RedundantFieldbus\\StateMachine\\FieldbusStateMachine.cs:line 254\n   at Chameleon.Automation.Core.Utility.RedundantFieldbus.FieldbusMaster`4.Work(CancellationToken cancellationToken) in C:\\data\\Chameleon.Core\\Chameleon.Automation.Core\\Utility\\RedundantFieldbus\\FieldbusMaster.cs:line 154"
}
```