# Alarm Event Propagation

## MQTT Modules v4.0.23 and above

In MQTT Transmission and MQTT Engine support was added to transmit alarm events from the Edge (MQTT Transmission) to the Central Gateway (MQTT Engine) . They also now support acknowledgements from the Central Gateway (MQTT Engine) back to the Edge (MQTT Transmission).



The Alarm Event Propagation implementation is limited as detailed below:

- Alarms over the GAN from an Engine gateway where alarms were forwarded over MQTT from Transmission is supported from v4.0.31 and on as long as the Remote Tag Providers 'Alarm Mode' is set to 'Queried'. This feature requires Ignition 8.1.49+
- Alarm pipelines and notifications exposed across MQTT along with alarm data is currently not supported
- Alarm Status Table filtering to a Sparkplug Group/Edge and/or Device ID is not fully supported
    - In release 4.0.28 Sparkplug IDs were added to the source and display path of alarms at MQTT Engine to allow for Alarm Status Table filtering to Sparkplug IDs
    - From release 4.0.28 onward, MQTT Engine supports explicitly inserting alarm events into the Alarm Journal on arrival

To configure alarm event propagation from Transmission to Engine, follow these steps:

### Ignition

1. Stop the Ignition instances for both MQTT Transmission and MQTT Engine
2. Add a line similar to the following to the data/ignition.conf file on each Ignition instance replacing the 'XX' with the proper index per the wrapper.java.additional statements.

```
wrapper.java.additional.XX=--add-opens=java.base/java.util.concurrent.atomic=ALL-UNNAMED
```

Restart the Ignition instances

### MQTT Transmission

1. Ensure that Alarm Event Enable has been set under the Transmitter Alarm Settings configuration
2. Ensure that a Disk Backed History Store has been configured and that the Transmitter has been configured under History Settings to use this history store. We recommend configuring a rolling buffer to manage a MQTT Transmission keep alive timeout scenario.

### MQTT Engine

1. Ensure that the Alarm Event Enable has been set under the General Alarm Settings configuration

### Testing

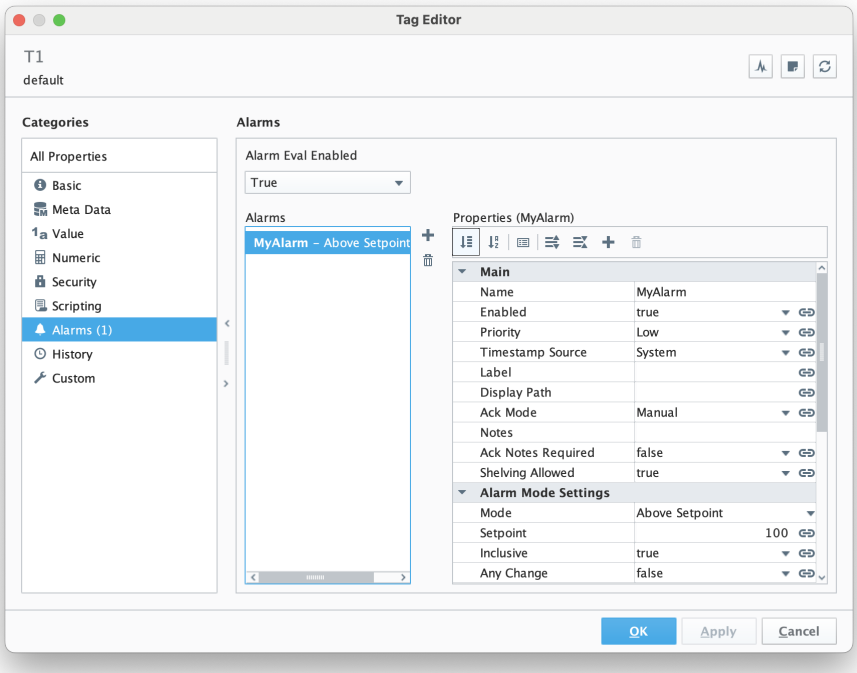
1. Set up a tag on the Transmission side with an alarm.



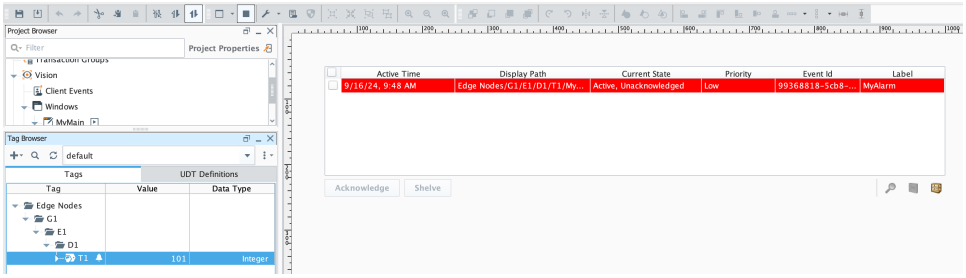
The Ack Mode must be set to Manual or Auto

Prior to 4.0.30 the only alarm properties that are propagated are: Name, Priority and Notes

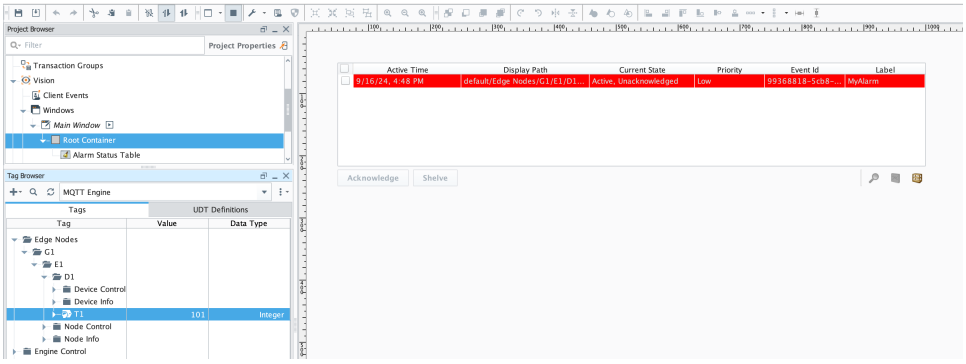
Below is an example that will generate an event if the value of the tag goes above 100:



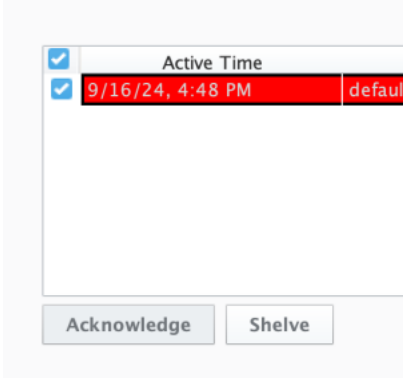

2. Trigger the alarm.
   In this example, we set the value of the memory tag to 101. This resulted in an alarm event at the Edge as shown below in the Alarm Status Table:



3. Because the alarm event was published via MQTT, we can also see it on the Central Gateway running MQTT Engine:



4. At this point, we can acknowledge the alarm at either the Edge or the MQTT Engine side via the Alarm Status Table by clicking the 'Acknowledge' button as shown below. Once acknowledged, it will be reflected as so on both the Edge and the MQTT Engine sides.




In order for the alarm acknowledgement or clearing at the Edge to be published to MQTT Engine, the Edge must be in an active Sparkplug session with MQTT Engine or have determined that the connection is unavailable such that the event changes are stored to the Disk Backed History Store for publication on reconnect.

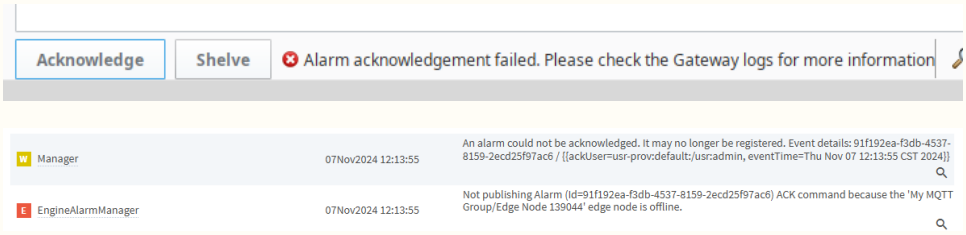
MQTT Transmission can take upto 1.5 times the configured keep alive time to determine that there is a connection failure and any alarm acknowledgements or clearings can be lost during this time leaving alarms 'stranded' at MQTT Engine. How to remove stranded alarms from the MQTT Engine Alarm Status Table.

The recommended approach to mitigate this issue is to configure a Rolling History Buffer for the Disk Backed History Store which will cover data loss during a keep alive timeout scenario

While testing the alarm acknowledgements whilst the Edge is disconnected, you can confirm that the connection status via the MQTT Transmission Servers view in the UI or through the MQTT Client Online tag

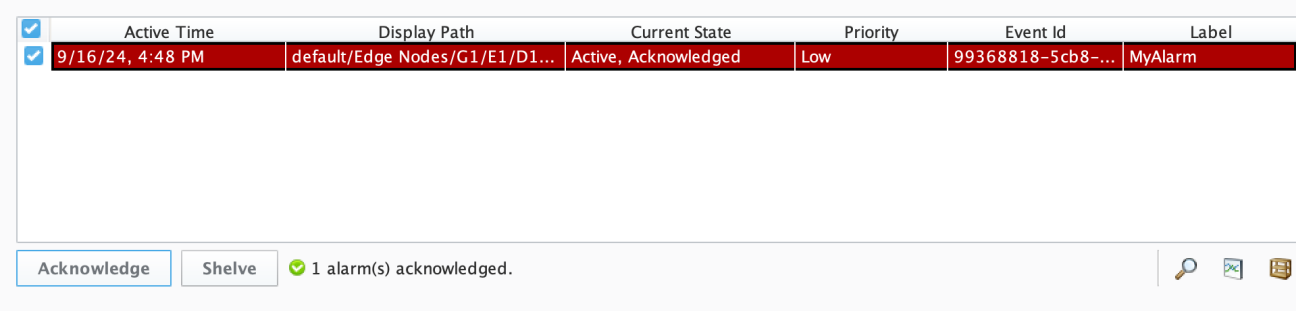
In order for the alarm to be acknowledged from MQTT Engine, the Edge must be in an active Sparkplug session with MQTT Engine

If the Edge node is offline, the following will be displayed in the Alarm Status Table and Ignition logs at MQTT Engine:

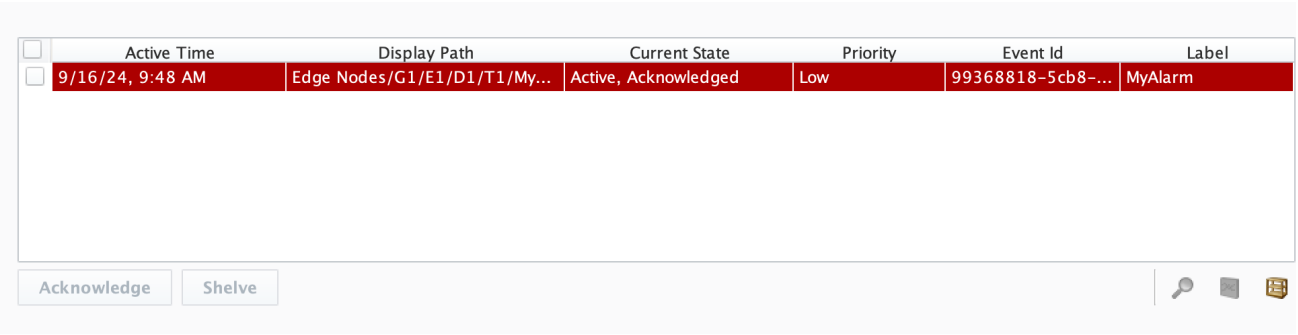



Once the Edge node comes online, any new alarm acknowledgement will be successful from MQTT Engine.

After doing so, you can see the acknowledged alarm on the Central Gateway (MQTT Engine side) :




and the Edge (MQTT Transmission side) :




### Removing stranded alarms from the MQTT Engine Status Table

If alarms have been 'stranded' in the MQTT Engine Alarm Status Table, they can be cleared using the MQTT Engine Python Scripting API with the readAlarms and clearAlarms calls.

## MQTT Modules v4.0.16 to v4.0.22

In MQTT Transmission and MQTT Engine v4.0.16 support was added to propagate triggered alarm events from the Edge to a Central Ignition Gateway. This can be used to insert alarm events into Ignition's alarm journal on the MQTT Engine side. Currently, these alarm events on the MQTT Engine side can not be currently be used in alarm pipelines, and they can not be remotely acknowledged from MQTT Engine back to MQTT Transmission.

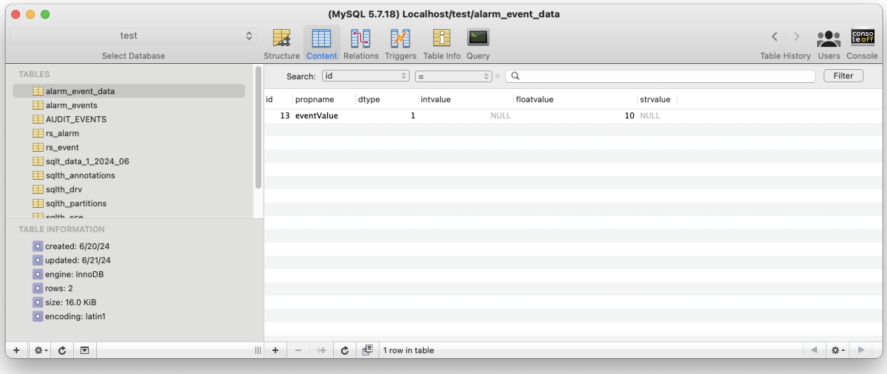
To configure alarm event propagation from Transmission to Engine to insert them into the Engine side alarm journal, follow these steps.

1. Add a line similar to the following to the data/ignition.conf file but replace the 'XX' with the proper index per the wrapper.java.additional statements. This must be done on both the MQTT Transmission side as well as the MQTT Engine side.

```
wrapper.java.additional.XX=--add-opens=java.base/java.util.concurrent.atomic=ALL-UNNAMED
```

2. Restart Ignition.
3. Set 'Alarm Event Enable' in the MQTT Transmission Transmitter configuration to true.
4. Set up an Alarm Journal in Ignition on the MQTT Engine side.
5. Set up a tag on the Transmission side with an alarm.
6. Trigger the alarm

7. After doing so, you should see something similar to the following in the alarm journal on the MQTT Engine side:




## Troubleshooting

If you have not updated the data/ignition.conf file on the Ignition instance(s) running MQTT Transmission and MQTT Engine correctly you will see this error in the Ignition logs from the com.cirruslink.mqtt.common.gateway.agent.alarm.AgentAlarmListener

```
Failed to handle the event
com.inductiveautomation.ignition.common.gson.JsonIOException: Failed making field 'java.util.concurrent.atomic.
AtomicReference#value' accessible; either increase its visibility or write a custom TypeAdapter for its
declaring type.
```

Double check that the *wrapper.java.additional.XX=--add-opens=java.base/java.util.concurrent.atomic=ALL-UNNAMED* syntax has been added to the ignition.conf file wrapper.java.additional statements and indexed correctly on both the MQTT Transmission and MQTT Engine Ignition instances.

```
Example from Ignition version 8.1.44
#********************************************************************
# Wrapper Java Properties
#********************************************************************
# Java Application
#  Locate the java binary on the system PATH:
#  Specify a specific java binary:
set.JAVA_HOME=lib/runtime/jre-win
wrapper.java.command=%JAVA_HOME%/bin/java

# Tell the Wrapper to log the full generated Java command line.
#wrapper.java.command.loglevel=INFO

# Java Main class.  This class must implement the WrapperListener interface
#  or guarantee that the WrapperManager class is initialized.  Helper
#  classes are provided to do this for you.  See the Integration section
#  of the documentation for details.
wrapper.java.mainclass=org.tanukisoftware.wrapper.WrapperSimpleApp
...
# Java Classpath (include wrapper.jar)  Add class path elements as
#  needed starting from 1
wrapper.java.classpath.1=lib/wrapper.jar
wrapper.java.classpath.2=lib/core/common/*
wrapper.java.classpath.3=lib/core/gateway/*

# Java Library Path (location of Wrapper.DLL or libwrapper.so)
wrapper.java.library.path.1=lib
wrapper.java.library.path.2=lib/core/gateway

# Java Bits.  On applicable platforms, tells the JVM to run in 32 or 64-bit mode.
#wrapper.java.additional.auto_bits=TRUE
wrapper.java.additional.auto_bits.macosx=FALSE

# Java Additional Parameters
wrapper.java.additional.1=-Ddata.dir=data
```

```
wrapper.java.additional.2=--add-opens=java.base/java.lang=ALL-UNNAMED
wrapper.java.additional.3=--add-opens=java.base/java.io=ALL-UNNAMED
wrapper.java.additional.4=--add-opens=java.base/java.security.cert=ALL-UNNAMED
wrapper.java.additional.5=--add-opens=java.base/java.util=ALL-UNNAMED
wrapper.java.additional.6=--add-opens=java.base/jdk.internal.misc=ALL-UNNAMED
#wrapper.java.additional.6=-Xdebug
#wrapper.java.additional.7=-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=*:8000
wrapper.java.additional.7=--add-opens=java.base/java.util.concurrent.atomic=ALL-UNNAMED

# Initial Java Heap Size (in MB)
wrapper.java.initmemory=1024
```

## Additional Resources

- Inductive Automation's Ignition download with free trial
  - Current Ignition Release
- Cirrus Link Solutions Modules for Ignition
  - Ignition Strategic Partner Modules
- Support questions
  - Check out the Cirrus Link Forum: https://forum.cirrus-link.com/
  - Contact support: support@cirrus-link.com
- Sales questions
  - Email: sales@cirrus-link.com
  - Phone: +1 (844) 924-7787
- About Cirrus Link
  - https://www.cirrus-link.com/about-us/