MQTT Engine Custom Namespace

- Knowledge of Ignition and Module installation process: Cirrus Link Module Installation
- Have Ignition 8.0.16 or greater installed
- Have MQTT Engine 4.0.10 or later installed

Abstract

MQTT Engine Custom Namespaces are used to provide support for generic, non Sparkplug compliant MQTT messages with string based payloads. If a custom namespace is configured MQTT Engine will convert all messages received to tags where the topic of each message will translate directly to the tag's path and the payload will be the tag's value.

> ⚠ If MQTT Engine is receiving only custom namespace messages, we recommend disabling the Sparkplug B default namespace.
>
> If MQTT Engine is receiving messages from both the Sparkplug B default namespace and a custom namespace, you will need to ensure that your custom namespace subscription is refined so as not to pick up the messages from the Sparkplug B namespace (spBv1.0/.....)
>
> From release 4.0.14, MQTT Engine will match ALL namespaces for any incoming message. For example, two custom namespaces can be created where both subscribe on the same topic but one can be configured to parse the payload as JSON and one to parse the payload as string based.

MQTT Engine supports MQTT Engine String Replacement for both MQTT Topics and Payloads for character sequences that will become part of an Ignition tag path. While MQTT and Sparkplug both support characters such as . & % =, Ignition does not support these as valid characters in a tag path or tag name. As a result, sometimes it may be necessary to tell MQTT Engine to replace certain characters or strings of characters with something else so the tag path and tag names can be properly created in Ignition.

Examples

Subscriptions

Subscriptions support both the multi-level (#) and single level (+) wild card characters.

- (+) is a a single level wildcard that matches any name for a specific topic level. We can use this wildcard instead of specifying a name for any topic level in the topic filter.
- (#) is a multi level wildcard that we can use only at the end of the topic filter as the last level and matches any topic whose first levels are the same as the topic levels specified at the left-hand side of the # symbol.

Examples for wildcard usage are below:

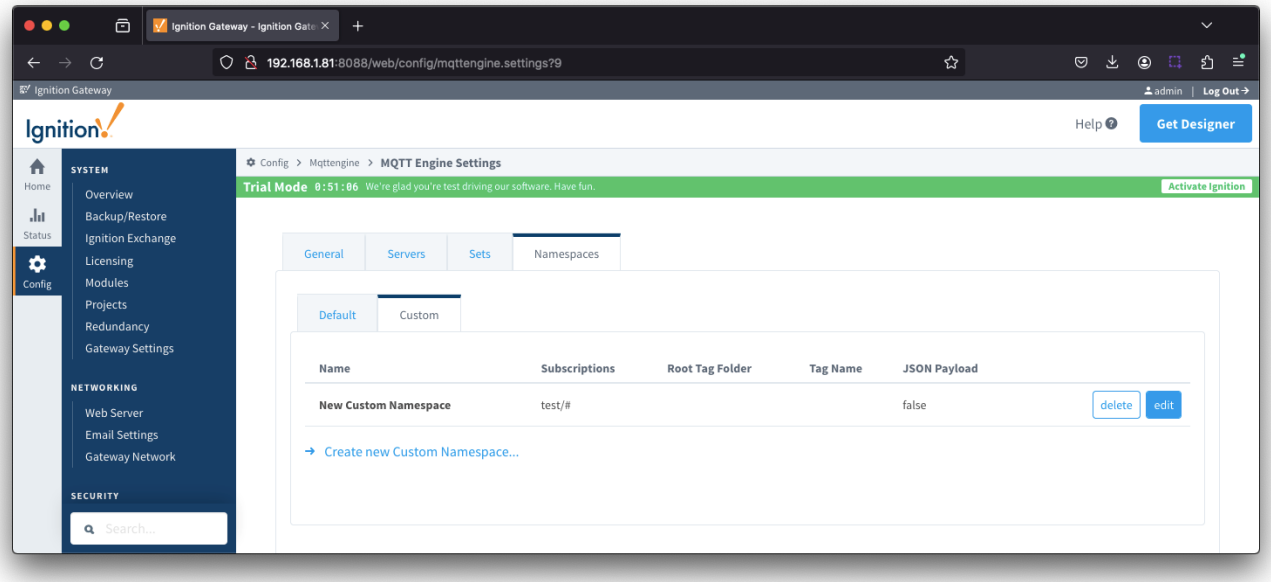
- subscribing on site/# covers
  - site/area1/pump
  - site/area1/flowmeter
  - site/area2/pump
  - site/area3
- subscribing on site/+/pump covers
  - site/area1/pump
  - site/area2/pump
- subscribing on site/+/+ covers
  - messages that are published with only two distinct topic levels below site such as site/area2/pump
  - messages that have less than, or more than, two levels will be filtered such as site/area3 or site/area2/pump/pressure

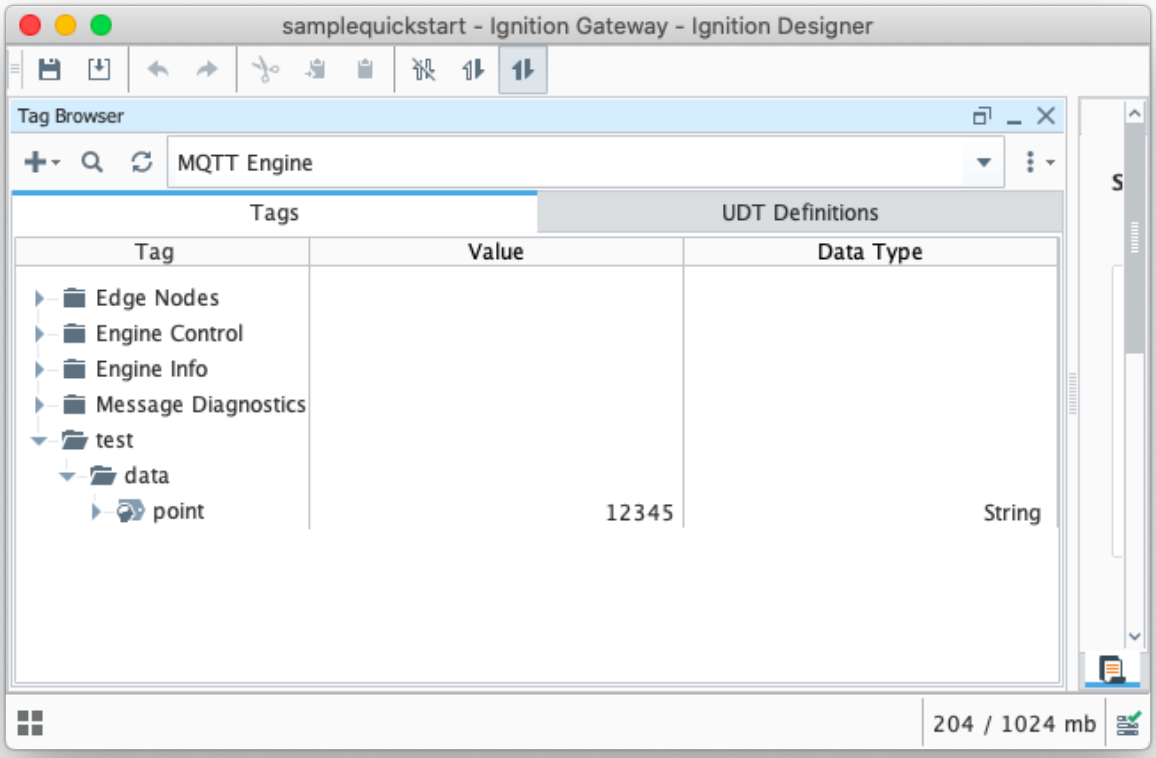
Note: Subscriptions such as site# or site+ are invalid as there is no topic level for the wildcard to denote.

For the examples below we will subscribe on test/#

No Tag Name configured

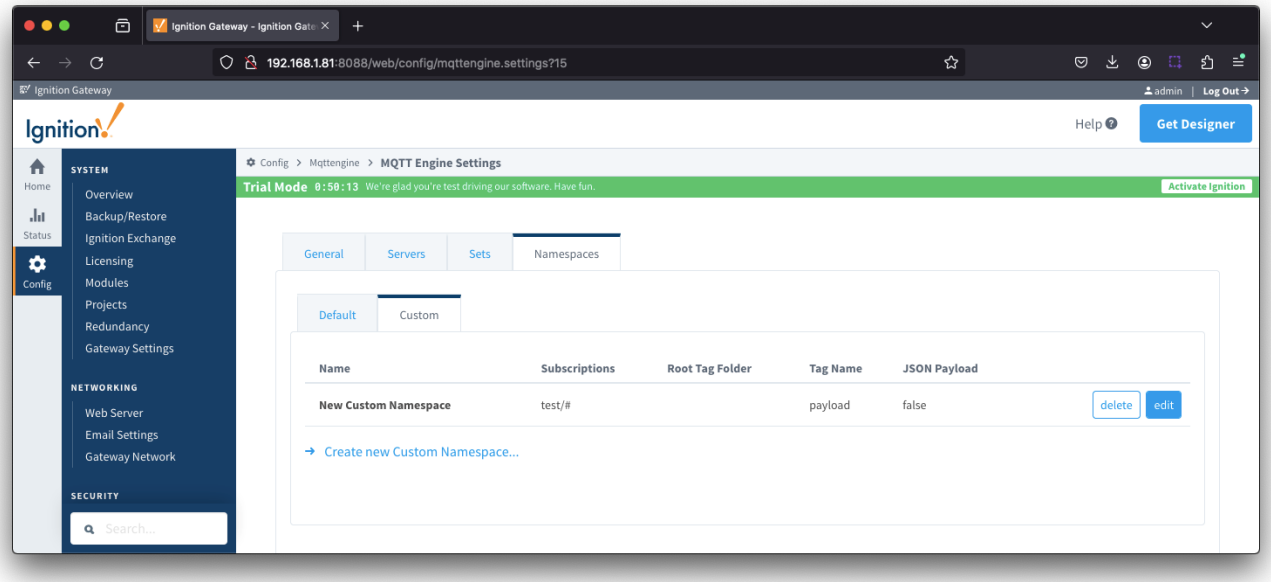
Let say we have a publish received on the topic "test/data/point" with value "12345". If no Tag Name is configured, and a message is received, MQTT Engine will create a two folders "test" and "data" and a tag "point" with the value of "12345".

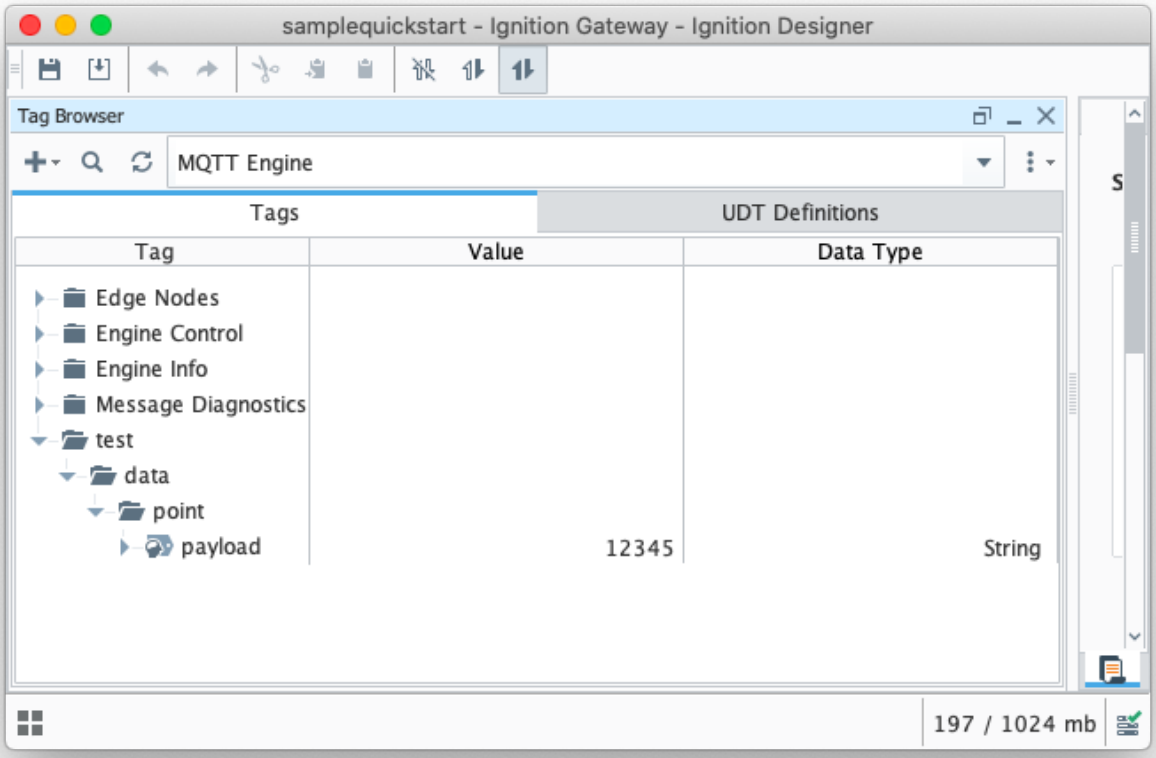




Tag Name Configured

If a Tag Name is configured, lets call it "payload", then MQTT Engine will convert each token in the topic to a folder and create a tag called "payload" with the value "12345"





In most cases it is useful to specify a Tag Name in order to prevent cases when a publish on a topic can overwrite a previously created tag, changing it into a folder. Consider the case where you have the following two publishes:

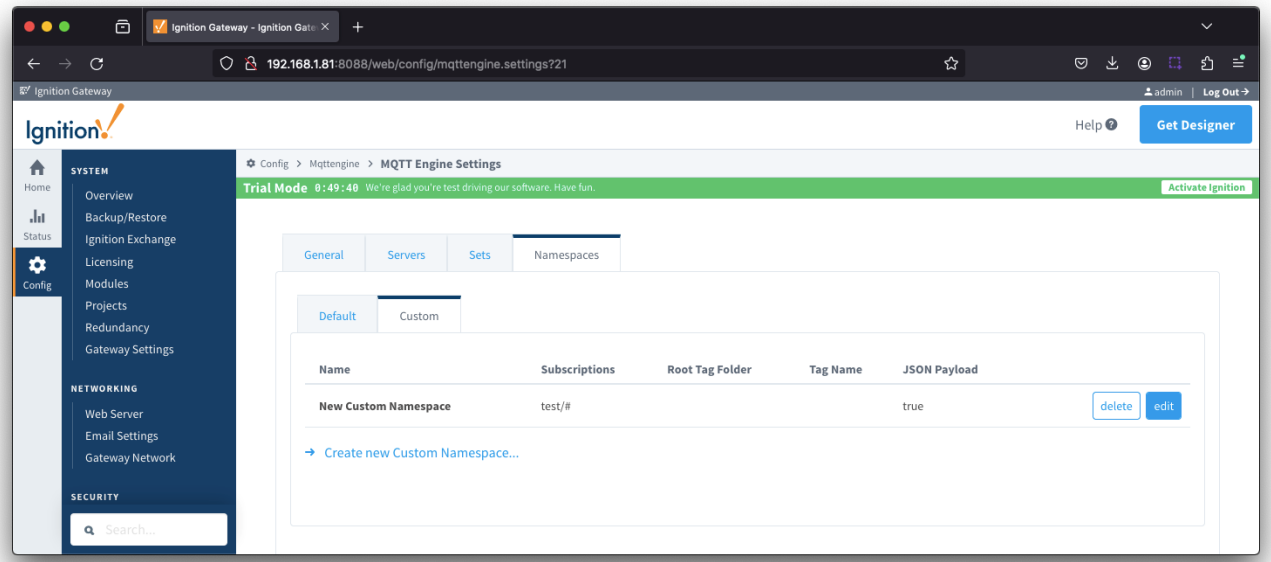
1st publish on topic "one/two" will create a tag named "two" in the folder "one"

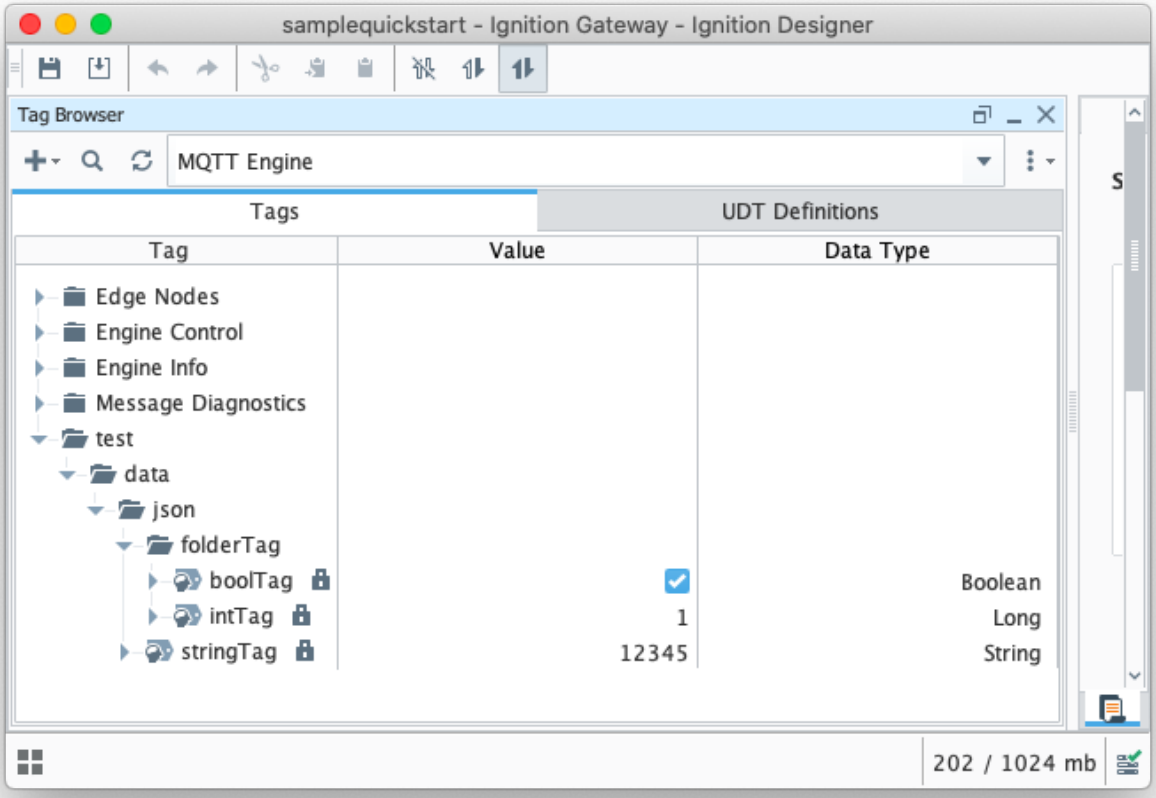
2nd publish on topic "one/two/three" will create a tag named "three" in the folder "one/two"

When the 2nd publish is received it will overwrite the first tag because "two" is now a folder instead of a tag. This folder/tag name collision can be avoided by specifying the Tag Name to always use for tags.

JSON Example

Let say we have a publish received on the topic "test/data/json" with value '{ "stringTag" : "12345", "folderTag" : { "intTag" : 1, "boolTag" : true } }'. MQTT Engine will create a three folders "test", "data" and "json" followed by a tag/folder structure representing the JSON value of the payload.





Common Use Cases

Managing Ignition timestamps for MQTT data when using custom namespaces

Reading bytes from an incoming binary message

How do I managing MQTT messages with changing payload JSON structure at MQTT Engine

Additional Resources

- Inductive Automation's Ignition download with free trial
  - Current Ignition Release
- Cirrus Link Solutions Modules for Ignition
  - Ignition Strategic Partner Modules
- Questions about this tutorial?
  - 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/