TECHNICAL NOTE

Maple Model(s)	Title	TN5119
Advanced HMIs Smart HMIs	Using Ignition MQTT Engine with Maple Systems HMIs	

P/N: 0907-5119
Rev. 00 Date: 05/01/2017

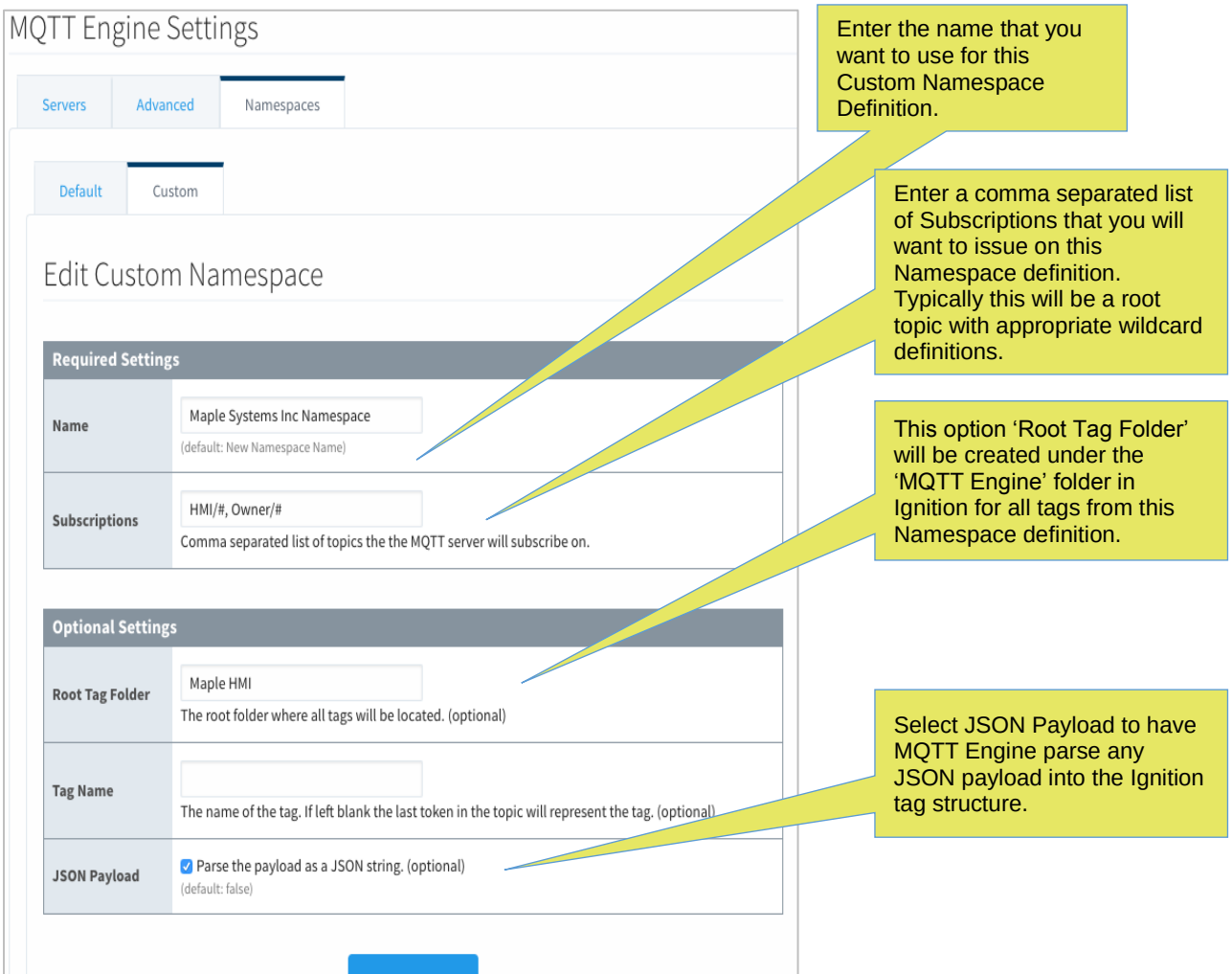
Summary

Ignition MQTT Engine now supports the native JSON format that is used by Maple System HMIs to publish HMI memory addresses using MQTT. This Tech Note assumes that the user already has the Ignition platform installed along with the MQTT Engine and MQTT Distributor (or other MQTT Server infrastructure). For instructions on how to setup MQTT publishing on a Maple Systems HMI, see the <MQTT Set-up> tech note.

Setting up MQTT Engine

1. Configuring Ignition MQTT Engine for a custom Topic Namespace and JSON Payload

Open the Ignition Gateway "Configure" console and navigate to **MQTT Engine Setting/Namespaces/Custom**, and click on "**Create new Custom Namespace…"**



2. Testing JSON Payloads

Once the custom Topic Namespace has been configured and saved, we can proceed to publishing some actual messages using the MQTT sample project and see how the resulting JSON information is parsed into the Ignition tag structure.

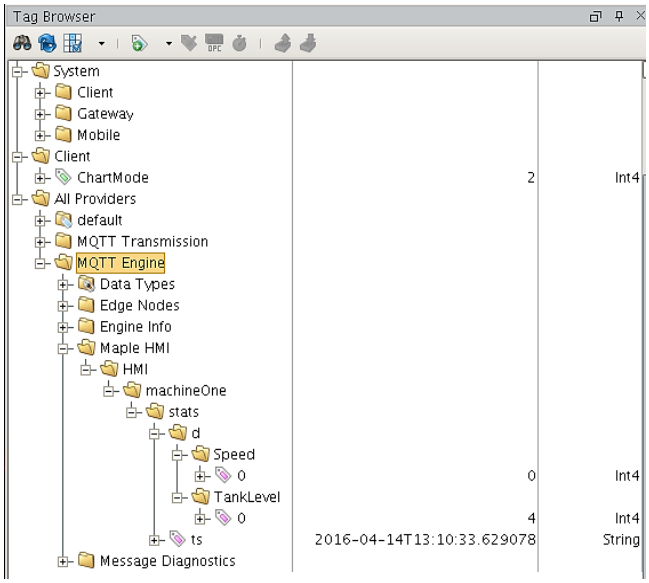
Open the Ignition Designer and then navigate to the All Providers/MQTT Engine folder in the tag browser. Note that prior to publishing any messages on the newly defined Topic Namespace, the defined “Maple HMI” folder does not exist.

Using the MQTT sample project we will publish an MQTT message. Then we will refresh the Ignition tag browser and see the resulting data structure.

Topic: **HMI/machineOne/stats**

JSON Payload:

```
{
        "d" : {
                "Speed" : [0],
                "TankLevel" : [4]
        },
        "ts" : "2016-04-14T13:10:33.629078"
}
```



3. Tag Creation Details

So what just happened when Ignition received the Maple Systems formatted MQTT message internal to the Ignition tag structure? First we can take a look at the JSON object that was published by the Maple Systems HMI device:

```
{
        "d" : {
                "Speed" : [0],
                "TankLevel" : [4]
        },
        "ts" : "2016-04-14T13:10:33.629078"
}
```

The JSON standard is well defined at http://json.org. The published JSON message contains 2 upper level elements, an object named "d" which we define as the object that contains the data points, and an object named "ts" which we define as the combined date and time timestamp based on when the HMI device published the message.

In this example, the "d" object contains 2 objects called "**Speed**", and "**TankLevel**". Note that because of the brackets ([ ]) in the JSON format, both Speed and TankLevel contain an array of values. In this case since there is only a single value in each array then these are arrays just containing a single element. Speed is an array of 1 with a value of 0, while TankLevel is an array of 1 with a value of 4.

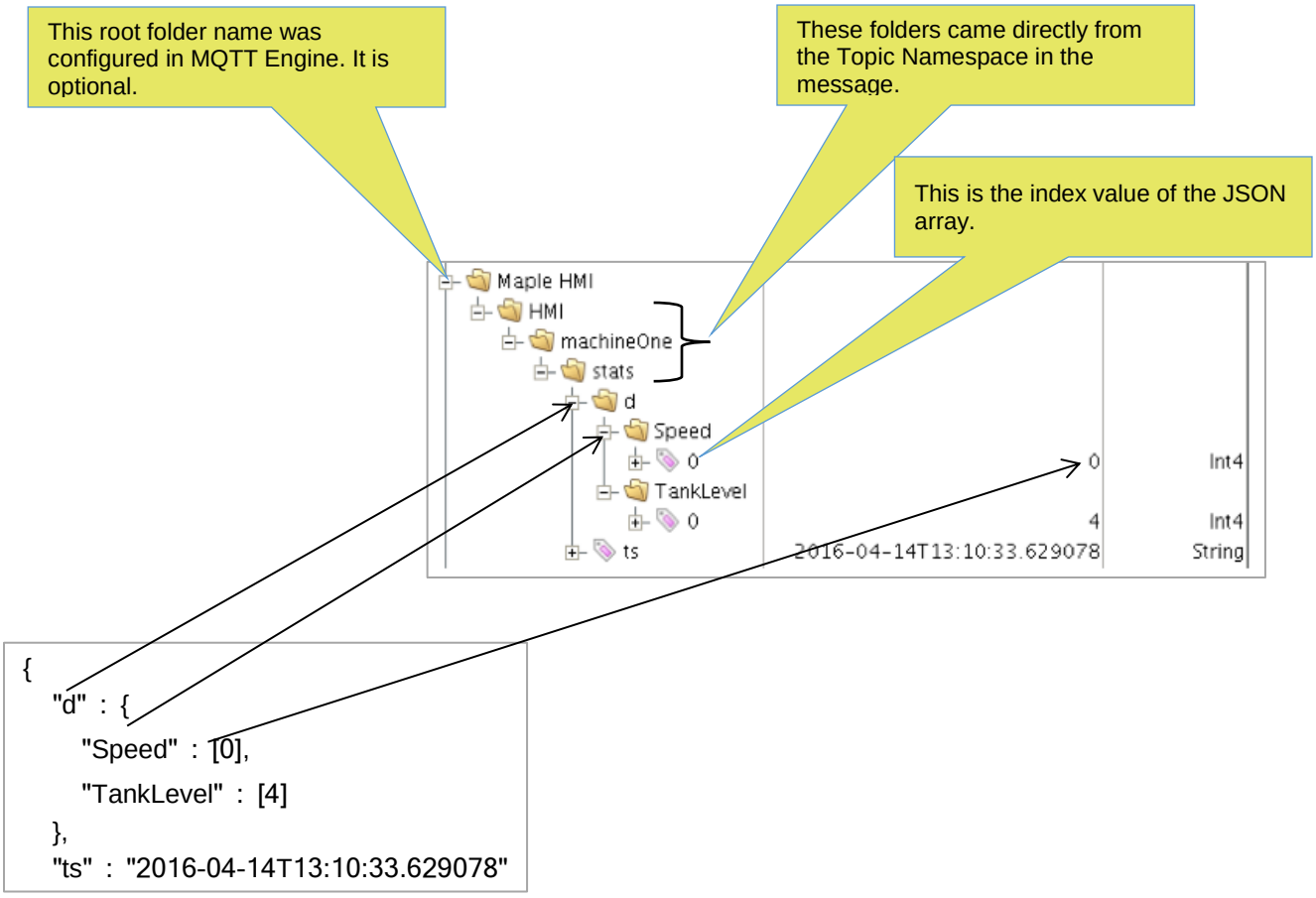
Now with the MQTT Engine Custom Namespace configuration, the Topic Namespace, and the JSON message, MQTT Engine can now create a tag hierarchy that represents the message it just received. In the Custom Namespace configuration we specified that all data received from the Maple Namespace be placed into a folder named "**Maple HMI**". So upon the reception of test message we just published from MQTTfx, a folder was created in MQTT Engine/Maple HMI. From there, MQTT Engine actual values in the received Topic Namespace to build out the rest of the folder structure. So in this example the complete folder path constructed was:

MQTT Engine/Maple HMI/machineOne/stats

With the Topic Namespace folder structure complete, now MQTT Engine will start to build out the tag structure that the JSON object represents. The first object, "d" is parsed into its representation which results in the "**d**" folder with the Speed array and TankLevel array as subfolders. Finally the actual data values of each of these arrays are populated into their respective array offsets. In this example this results in d/speed/0 = 0 (with an **Int4** data type), and d/TankLevel/0 = 4 (with an **Int4** data type).

The second object in the JSON message is the "ts" object which is not an array as the resulting value is a **String** data type.

Published JSON object shown in the Ignition tag browser.



Published JSON Object

4. Scaling Floating Point Values

MQTT Engine will create Ignition tags from JSON using appropriate type conversion:

"value" : 1234 ← Ignition Int4

"value" : 1234.0 ← Ignition Float8

"value" : "1234" ← Ignition String

"value" : true ← Ignition Boolean

In some applications the values being published from the Maple HMI may be integer values that actually need to be scaled before displaying in an Ignition dashboard. For example, a "d" value array might contain a value of 1234 that actually represents a floating point value of 12.34.

Observe the value of 1234 in the Ignition tag browser:

If you add the *Maple HMI/HMI/machineOne/stats/Speed/0* tag directly to the dashboard the resulting display value will be 1,234. But if you want to scale this value, highlight the “Property Editor” in Ignition Designer and select the “Value” property. By default the value being displayed will have a Binding Type of “Tag”.

To scale the value being displayed, select the “Expression” binding. By default this will show the tag path but now you can perform pretty much any mathematical operation you want to the value. In the screenshot below, the “Speed/0” tag is being divided by 100.0 resulting in the display of the scaled value.