Internet of Things – MQTT Publishing Format for Consumption by GENESIS64

V10.96.1

November 2020
An ICONICS Whitepaper
www.iconics.com

ICONICS
Make the Invisible Visible™

Contents

Document History

Version	Issued By	Issue Date	Comments
1	Bhargav Reddy, Jotham Kildea	11/17/20	Initial Draft
2			
3			

Document Overview

The document describes the message format to use for the JSON messages published via MQTT by any device to an MQTT broker that GENESIS64 subscribes to receive that data.

The document makes the following assumptions:

1. The End User has an MQTT publishing device with a known or configurable JSON message format.
2. The End User has an MQTT broker which can receive messages from their Publisher.

1. Preferred Features for a Publisher Device

There are a variety of sensors, gateways, and other devices and applications capable of publishing JSON-formatted messages to an MQTT broker. Some devices are rigidly defined in how they enqueue and publish messages, while others allow for some customization and control. The following capabilities are preferred in order to have a robust and reliable connection via MQTT publishing:

1. On device restart the device should publish all configured data points.
2. The device should support the configuration of a watchdog status message. It should be possible to configure this on a time or day or time increment update.
3. The device should publish new messages to the broker according to a Change of Value (COV) model, where updates to data points are only provided when there is some change in the underlying value. There may optionally be a configurable dead band for this threshold.
4. The device should have a configurable forced refresh rate, such that the present value of data points is published at some minimum interval, regardless of any change of value. This is done to ensure a maximum age of point values within the broker as well as to help detect stagnant values or communication errors.

2. MQTT Message Payload Format

The MQTT protocol defines the means of message transport but does not specify the format of the message that is sent. When used to publish data value updates from devices they are frequently encoded in JSON messages with a defined format. ICONICS can subscribe to messages in flexible formats, but to simplify configuration it is recommended to use one of the following. The following are 2 of the most used options of message formats

2.1. Topic for Publishing Data

When publishing messages to an MQTT broker, they are directed to “Topics” which will help to organize and identify the origin of the messages that the broker receives. Clients can subscribe to that topic to receive all updates related to a data source. It is important in situations where there may be multiple devices publishing messages with the same tag name to organize those messages in such a way so that they are maintained as unique data points. One approach can be to have each device publish messages under its own unique Topic, while another can be to include the Device Name as a keyword within the JSON message itself.

2.2. Simple Message Format (one value per message)

This is a simple format for the JSON message that can be published to an MQTT broker and accepted by an ICONICS subscriber. While it can result in a large total message count and is not very efficient, it is easy to set up and may be suitable for devices with very few datapoints or infrequently changing data.

Message Structure

```
{
  "id": "[tagname]",
  "v": "[value]",
  "q": "[quality]",
  "t": "[timestamp]"
}
```

Example Message

```
{
  "id": "CurrentOccupancy",
  "v": "7",
  "q": "true",
  "t": "1580413177050"
}
```

2.3. Standard Message Format (multiple values per message)

A more common format for the messages sent by the publisher is to include multiple tag updates in a single message. This format is similar to the simple format above, but groups multiple data updates into a single message. This will be more efficient in regard to the total message count and is well suited for devices which have an internally defined publish rate which would allow for publishing of multiple values at a standard time interval. The format can be extended to include other information about tags as needed.

Message Structure

```
[
  {
    "id" : "[tagname1]",
    "v" : "[value1]",
    "q" : "[quality1]",
    "t" : "[timestamp1]"
  },{
    "id" : "[tagname2]",
    "v" : "[value2]",
    "q" : "[quality2]",
    "t" : "[timestamp2]"
  },{…}
]
```

Example Message

```
[
  {
    "id" : "Temperature1",
    "v" : "-4.256",
    "q" : "true",
    "t" : "1605751173869"
  },{
    "id" : "Temperature2",
    "v" : "-6.159",
    "q" : "true",
    "t" : "1605751173870"
  }
]
```

2.4. Standard Message Format (multiple values with common attributes)

This format is similar to the standard format of multiple updates per message, but additionally allows for certain properties to be grouped globally for all datapoints in the message. By publishing messages in this format, it is assumed that all datapoints within the array share the attributes of the root object, unless explicitly overridden within that value. It can be an efficient format in terms of volume of data per message as it eliminates redundant data in the JSON message.

Message Structure

```
{
  "dvcname" : "[dvcname]",
  "t" : "[timestamp]",
  "v" : "[quality]",
  "values" :
  [
    {
      "id" : "[tagname1]",
      "v" : "[value1]"
    },{
      "id" : "[tagname2]",
      "v" : "[value2]"
    },{…}
  ]
}
```

Example Message

```
{
  "dvcname" : "Controller6",
  "t" : "1605751173869",
  "v" : "true",
  "values" :
  [
    {
      "id" : "Temperature1",
      "v" : "-4.256"
    },{
      "id" : "Temperature2",
      "v" : "-6.159"
    }
  ]
}
```

2.5. Keywords to use in Messages

There are a variety of properties about data points which may be included in a message which are defined individually as “keywords”.

The naming of key in a message is not restricted, it can be any name provided that it does not include reserved characters. For example, the message formats provided above included keyword names such as “id”, “q”, “t”, and “v”. The key names could be other values like “name”, “quality”, “time”, “Value”, these were chosen simply because they make for short, compact messages.

Messages should include at least a minimum set of keywords so that the messages can be interpreted properly by the ICONICS Suite. They include:

PUBLISHNAME – The publish name can be considered as the tag name or point name for the data being published. This is the name of the tag as it will be referenced within an MQTT broker. It is important that each publishing device should have distinct Publish Names for its tags. Use of Duplicate tag names will invalidate the data coming from the device. Also, when there are multiple publishers being used it is important to differentiate them by device name or topic if they have common publish names for tags.

VALUE – This is the updated value which is being sent in the message. It can be in a variety of formats depending on the underlying data. In configurations where there are multiple data updates being published, there would be a Value for each updated datapoint.

TIMESTAMP – This is the date time associated with the value sample. It should be the time at which the sample was recorded, and not necessarily the time at which the message was published to or received by the broker. There are variety or local, Unix timestamp format options available in Workbench. Both UTC and device local time are supported, following formats such as:

- .TEXT – string, e.g. 2017-03-22T16:09:44.5348262Z
- .TICKS – Int64, e.g. 636257795760626391
- .UNIX – Int64, milliseconds since 1 January 1970

STATUS – This is the quality of the data tag. It can be encoded in a variety of formats such as plain text or OPC quality codes, however the easiest is a simple true/false state. Some available formats are:

- .CODE – number, e.g. 0x00000007
- .TEXT – string, e.g. Good – From Cache
- .GOOD – boolean
- .OPCQUALITY – number, status code converted to classic OPC, e.g. 192

A more complete list of supported status code inputs is as follows:

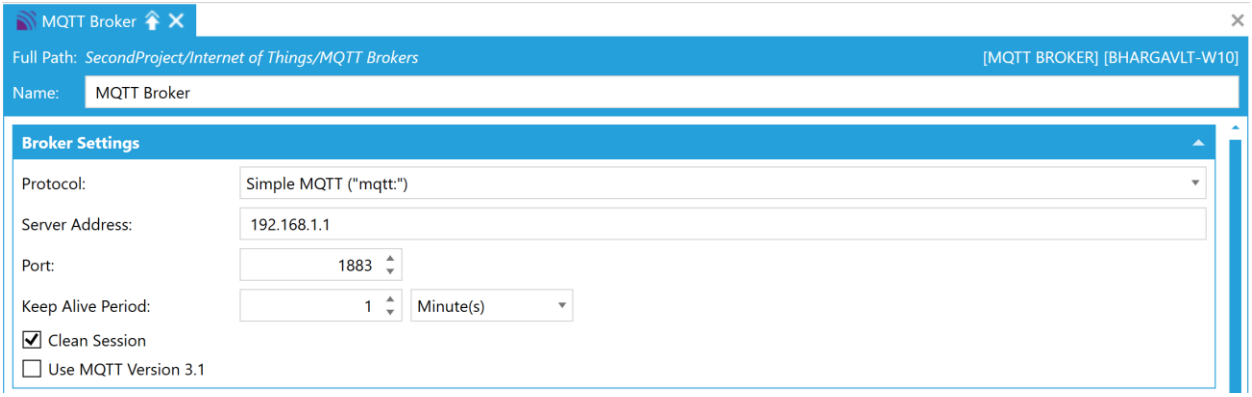
Quality State	Subquality	STATUS.CODE	STATUS.TEXT	STATUS.GOOD	STATUS.OPCQUALITY
Good	Good	0x00000000	Any string with "good" in it	TRUE	0xc0
GoodWriteDropped	Good	0x00000001			
GoodLicenseDemo	Good	0x00000002			
GoodMoreData	Good	0x00000003			
GoodNoData	Good	0x00000004			
GoodEntryInserted	Good	0x00000005			
GoodEntryReplaced	Good	0x00000006			
GoodFromCache	Good	0x00000007			
GoodLocalOverride	Good	0x00000009			0xd8
GoodClamped	Good	0x0000000b			
Bad	Bad	0x80000000	Any string without "good" in it	FALSE	0x00
BadInvalidRequestCategory	Bad	0x80000001			
BadInvalidRequestType	Bad	0x80000002			
BadInvalidPointHandle	Bad	0x80000003			
BadInvalidPointName, BadNodeIdInvalid	Bad	0x80000004			
BadNotImplemented	Bad	0x80000005			
BadInvalidInParams	Bad	0x80000006			
BadServerFailed	Bad	0x80000007			
BadUnexpectedError	Bad	0x80000008			
BadInvalidMethodName, BadMethodInvalid	Bad	0x80000009			
BadUserAccessDenied	Bad	0x8000000a			
BatTimeout	Bad	0x8000000b			
BadInternalError	Bad	0x8000000c			
BadNotWritable	Bad	0x8000000d			
BadNoDataAvailable	Bad	0x8000000e			
BadWaitingForInitialData	Bad	0x8000000f			0x20
BadTooManyOperations	Bad	0x80000010			
BadUnknownTypeName	Bad	0x80000011			
BadInvalidOutParams	Bad	0x80000012			
BadCommunicationError	Bad	0x80000013			0x18
BadInvalidProperty	Bad	0x80000019			
BadInvalidArgument	Bad	0x8000001a			
BadUnknownAggregateType	Bad	0x8000001b			
BadContinuationPointInvalid	Bad	0x8000001c			
BadTypeMismatch	Bad	0x8000001e			

BadConfigurationError	Bad	0x80000020			0x04
BadNotSupported	Bad	0x80000022			
BadInvalidRelativePath	Bad	0x80000023			
BadNoBound	Bad	0x80000024			
BadSecurityDisabled	Bad	0x80000025			
BadScalingFailed	Bad	0x80000026			
BadInvalidProcedureName	Bad	0x80000027			
BadProcedureStopped	Bad	0x80000028			
BadWouldBlock	Bad	0x80000029			
BadDataLost	Bad	0x8000002a			
BadServerClosedConnection	Bad	0x8000002b			
BadFilterNotAllowed	Bad	0x8000002c			
BadUnknownPointName	Bad	0x8000002d			
BadServerHalted	Bad	0x8000002e			
BadReadOnlySession	Bad	0x8000002f			
BadUncertain	Bad	0xc0000000			0x40
BadNotConnected	OPC Quality	0x80000600			0x08
BadDeviceFailure	OPC Quality	0x80000601			0x0c
BadSensorFailure	OPC Quality	0x80000602			0x10
BadOutOfService	OPC Quality	0x80000603			0x1c

3. Configuring the ICONICS Subscription

3.1. MQTT Broker configuration

1. Add an MQTT broker definition under the Internet of Things provider in Workbench.

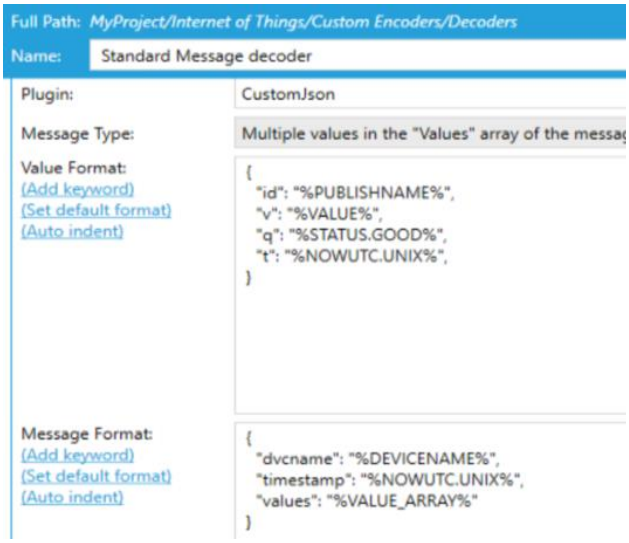


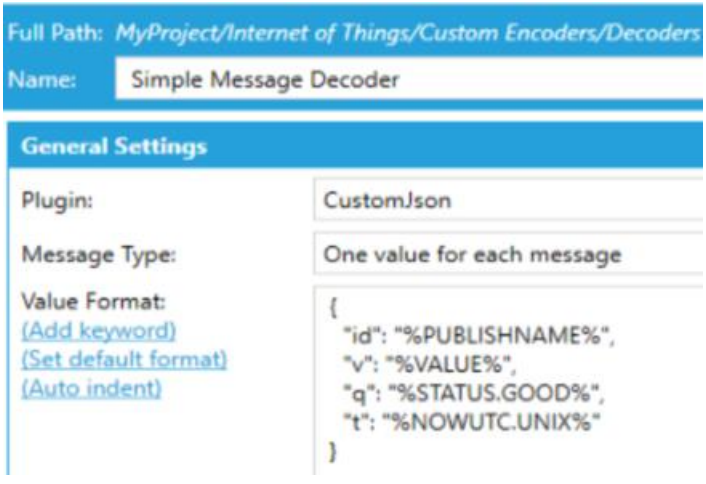
2. Make sure to define the IP address, port and security section of the form according to the Broker setup.

3.2. Configure the MQTT Message decoder

Next the JSON message format needs to be configured in Workbench. This is done by replacing the Key Values in the raw JSON with Keyword's available in ICONICS.

1. Create an encoder/decoder in the Internet of Things provide in Workbench.
2. In the Project Explorer, expand My project and then Internet of Things.
3. Right click on Custom Encoders/Decoders, click on Add Encoder/Decoder.
4. Change the Message Type to the appropriate selection from the drop-down menu. (previously described in section 2.2 and 2.3)





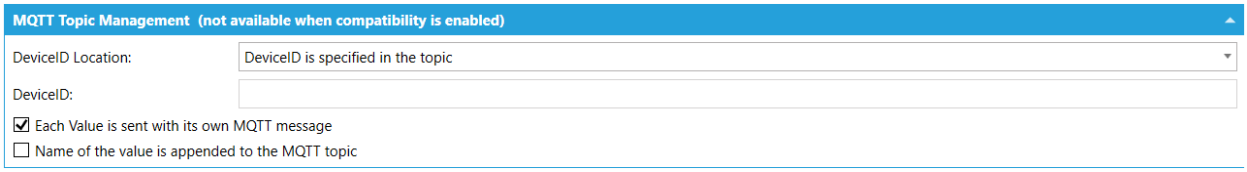
5. In the Value Format box add the decoder format by clicking on the blue edit defaults button.
6. Edit the default entries and add additional keywords. Remember to include the minimal keyword properties as described in section 2.4.

Note: The keywords indicate the data type of each key's value in the messages published via the MQTT protocol.

3.3. Subscriber connection

1. Add the IOT subscription to the broker.
2. Uncheck – enable compatibility with ICONICS clients.
3. Select the connection type to MQTT
4. Select the decoder added in the previous step.
5. Select the broker previously added.
6. Define the base topic that ICONICS needs to subscribe to.

Note: If you are using different topics for data publishing per device, it needs to be indicated in the topic management section.



a. DeviceID location option needs to be picked appropriately for the message format.
b. Each value is sent with its own MQTT message option should be disabled when using the standard message format that has multiple tag updates per MQTT message.

3.4. Heartbeat configuration

The Keep alive timeout on the subscription is set to 1 minute by default. This is how long the Point Manager waits before marking an MQTT message received from the broker as old/stale and ignores that update.

So, it is important for the Publishing device to send continuous updates within the Keep alive time settings in Workbench.

The heartbeat message could be a message that contains no tag updates or has a single tag's update that is used as a heartbeat reference tag.

The reference tag could be something as simple as the current time of the MQTT device for instance.

Examples:

Message Encoder Structure

```
{
  "id": "%PUBLISHNAME%",
```

Example Raw JSON Heartbeat Message

```
{
  "id": "Heartbeat",
```

```
  "v": "%VALUE%",
  "q": "%STATUS.GOOD%",
  "t": "%NOWUTC.UNIX%"
}
```

```
  "v": "0”,
  "q": true,
  "t": 1580413177050
}
```

Message Encoder Structure

```
{
  "dvcname" : "%DEVICENAME%",
  "timestamp" : "%NOWUTC.UNIX%",
  "values" :
  [
    {
      "id" : "%PUBLISHNAME%",
      "v" : "%VALUE%",
      "q" : "%STATUS.GOOD%",
      "t" : "%NOWUTC.UNIX%"
    },{
      "id" : "%PUBLISHNAME%",
      "v" : "%VALUE%",
      "q" : "%STATUS.GOOD%",
      "t" : "%NOWUTC.UNIX%"
 },{…}
  ]
}
```

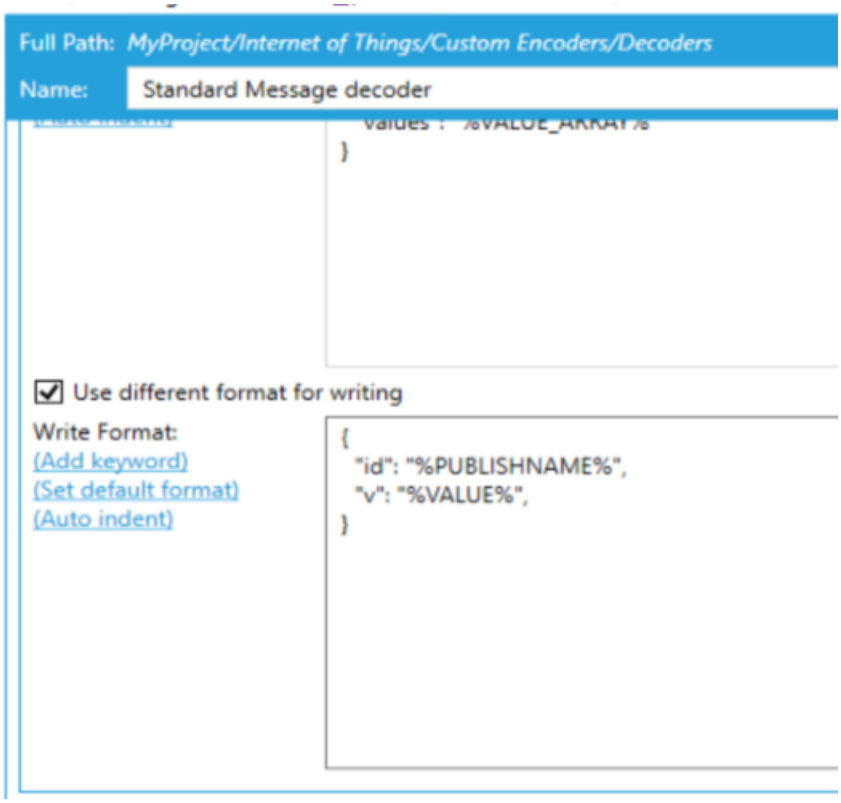
Example Message

```
{
  "dvcname" : "Controller6",
  "timestamp" : "1605751173869",
  "values" :
  [
    {
      "id" : "Temperature1",
      "v" : "-4.256",
      "q" : "true",
      "t" : "1605751173869"
    },{
      "id" : "Temperature2",
      "v" : "-6.159",
      "q" : "true",
      "t" : "1605751173870"
    }
  ]
}
```

4. Writing values to the MQTT device using ICONICS

4.1. Decoder modification for Writing back to the Gateway

By default, there is no write capabilities enabled on the subscriber settings. To allow write back to the device, enable the 'Use different format for writing' on the Custom decoder used in Workbench.



The default write format looks like:

Message Encoder Structure

```
{
  "id": "%PUBLISHNAME%",
  "v": "%VALUE%",
}
```

Example Raw JSON Message

```
{
  "id": "TemperatureSetpoint",
  "v": "21.2",
}
```

This will need to be handled on the Gateway's MQTT subscription to parse these write requests back to the device data points.

Founded in 1986, ICONICS is an award-winning independent software provider offering real-time visualization, HMI/SCADA, energy management, fault detection, manufacturing intelligence, MES, and a suite of analytics solutions for operational excellence. ICONICS solutions are installed in 70 percent of the Global 500 companies around the world, helping customers to be more profitable, agile and efficient, to improve quality, and to be more sustainable.

ICONICS is leading the way in cloud-based solutions with its HMI/SCADA, analytics, mobile and data historian to help its customers embrace the Internet of Things (IoT). ICONICS products are used in manufacturing, building automation, oil and gas, renewable energy, utilities, water and wastewater, pharmaceuticals, automotive, and many other industries. ICONICS' advanced visualization, productivity, and sustainability solutions are built on its flagship products: GENESIS64™ HMI/SCADA, Hyper Historian™ plant historian, AnalytiX® solution suite, and MobileHMI™ mobile apps. Delivering information anytime, anywhere, ICONICS' solutions scale from the smallest standalone embedded projects to the largest enterprise applications.

ICONICS promotes an international culture of innovation, creativity, and excellence in product design, development, technical support, training, sales, and consulting services for end users, systems integrators, OEMs, and channel partners. ICONICS has over 350,000 applications installed in multiple industries worldwide.

World Headquarters
100 Foxborough Blvd.
Foxborough, MA, USA, 02035
+1 508 543 8600
us@iconics.com

European Headquarters
Netherlands
+31 252 228 588
holland@iconics.com

Australia
+61 2 9605 1333
australia@iconics.com

Canada
+1 647 544 1150
canada@iconics.com

China
+86 10 8494 2570
china@iconics.com

Czech Republic
+420 377 183 420
czech@iconics.com

France
+33 4 50 19 11 80
france@iconics.com

Germany
+49 2241 16 508 0
germany@iconics.com

India
+91 265 6700821
india@iconics.com

Italy
+39 010 46 0626
italy@iconics.com

Middle East
+966 540 881 264
middleeast@iconics.com

Singapore
+65 6667 8295
singapore@iconics.com

UK
+44 1384 246 700
uk@iconics.com

Microsoft Partner | 2017 Partner of the Year Winner Application Development Award

Gold Microsoft Partner

VISUALIZE HISTORIZE MOBILIZE ANALYZE IoT CLOUD