JSON-RBE MQTT Payload Format

Elecsys JSON-RBE Protocol Specification

Updated 7/6/2017

- MQTT Protocol
- Overview of JSON RBE
- Connection Sequence
- Topic Namespace
- Birth and Death Certificates
- Data Publish Payload Format
- Data Subscription Payload Format
    - Example JSON Data Subscription
- Gateway Birth Payload Format
- Gateway Subscription Payload Format
- Historical Data Payload

MQTT Protocol

MQTT has become an industry standard for IoT and Cloud publishing applications. It is a low bandwidth protocol for publish/subscribe over TCP/IP, with different levels of assurance for delivery. The protocol was originally designed for the specialized client devices and network types found in critical infrastructure SCADA and remote telemetry applications. Important features of the protocol are :

- **Minimizes Bandwidth** - The protocol minimizes network bandwidth by having a header overhead of only 2 bytes.
- **Assurance of Delivery** - MQTT offers three levels of delivery assurance, or Qualities of Service, referred to as QoS0, QoS1, and QoS2. Different qualities of service are appropriate for different types of data, and can be selected on a per-message basis by a publishing application.
  QoS 0 is a simple "fire and forget" model, which offers "at most once" delivery.
  QoS 1 uses a simple acknowledgment to provide "at least once delivery", which ensures that in the event of network failure, the message is eventually delivered. However, loss of the acknowledgment message can result in the client application (correctly) sending the message more than once.
  QoS 2 has more message flows in the conversation between the client and the server (sometimes referred to as a "broker"), but assures a once-and-once-only delivery of the message.

- **Last Will and Testament** - This mechanism is used to detect unexpected death, or failure, of the network or the client hardware. When an MQTT client connects to the broker, it can optionally specify a Will Message and Will Topic, and a keep-alive time, specified in seconds. If the client fails to publish anything to the broker during the keep-alive time, the broker assumes the untimely death of the client, and closes the client connection. The broker then publishes the Will Message using the Will Topic on behalf of the client to all other clients currently subscribed for the failed client.

The protocol has a very basic publish/subscribe verb set: CONNECT, DISCONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE, and an application-level keep-alive: PINGRequest and PINGResponse. There are also message acknowledgment verbs, which are used to manage the assured message delivery.

Overview of JSON RBE

The JSON RBE protocol is an application-level messaging protocol in the Elecsys Industrial Data Gateway (IDG) products, such as RediGate and Director. JSON RBE uses MQTT to publish **report by exception** (RBE) data and to indicate the health and communication status of the gateway and all attached field devices (RTUs, PLCs, flow meters, sensors, etc.). The gateway publishes RBE data using a Quality of Service of 0. This means that if an RBE publication is received when there is no active application subscribed, it will be discarded by the broker. This will prevent non-required messages building up at the broker.

The JSON RBE application protocol consists of several categories of activities:

1. Connect to the broker
2. Disconnect from the broker
3. Publish RBE device data to a host application
4. Accept device commands from a host application
5. Accept gateway system control commands

Connection Sequence

On connection of a gateway to the MQTT broker, the sequence of publications is as follows:

- Gateway BIRTH, including gateway system metrics.
- ‘RBE’ BIRTH publish for device #1 (initial dump of all data, could be one or more JSON packets of 4K bytes or less).
- ‘RBE’ publish for device #2, and so on for multiple devices.
- After that, data is sent on change of state (RBE).

The topic namespace and message structures for these messages are described in later sections.

Topic Namespace

All JSON RBE protocol messages originating from the gateway are published to the broker. The MQTT pub/sub mechanism requires that all messages contain a topic string, and an optional payload consisting of a stream of bytes. The Elecsys gateway JSON publisher allows for configurable topics to publish data and to subscribe for commands. The data topic should be configured according to the needs of the host application. Subscriptions ideally should allow for wildcards, so that a gateway can subscribe for only the data intended for itself or its attached devices.

Birth and Death Certificates

The JSON RBE protocol contains information about the communications status of the gateway and its field devices. A birth certificate will indicate that a device is now “ONLINE” using a JSON tag. The death certificate published on behalf of a device indicates that the device has gone “OFFLINE” using a JSON tag.

A gateway birth certificate will be issued each time the gateway makes an IP connection to a broker. A death certificate for the gateway (Last Will & Testament) will be lodged with the broker on initial connection. When the gateway MQTT connection terminates, the broker will publish this death certificate on behalf of the gateway. The payload of the gateway’s Last Will & Testament will be a string that has been configured for each gateway.

Data Publish Payload Format

An example JSON data payload for publishing data from a field device is:

```
{
   "d": {
   "gwName": "GatewayName",
   "devName": "FieldUnit",
   "BoolTag1": false,
   "BoolTag2": false,
   "00003": false,
   "00004": false,
   "AnaTag1": 0,
   "40002": 0,
   "FloatTag1": 123.456,
   "47002": "",
   "StringTag1": "This is a string value",
   "49002": "This is another string",
   "rtuIsAlive": true,
   "SeqNumb": 0
   }
}
```

Explanation of tags:

- *gwName* = RediGate name (from System object)
- *devName* = device name (from Field Unit object)
- Data tags (*BoolTag1*, *StringTag1*, *40002*, etc.) are either the tag name (if configured) or the RTDB register number of the value. Values associated with each tag should follow JSON rules:
  - Boolean = true or false
  - Integer = number (no quotes)
  - Float = number that must include decimal point
  - String = text (in quotes)
- *rtuIsAlive* = true or false (included in last packet of data from a device on birth; after that, included in all packets to indicate communication status of field device)
- *SeqNumb* = starting from 0 on initial connection, increment to 65535 and roll over to 0. Host application can monitor for lost sequence numbers and re-request full device data if needed.

If a device changes from good to failed communication, a Device Publish message will be published with the next sequence number and *rtuIsAliv e* = false, such as:

```
{
       "d": {
       "gwName": "GatewayName",
       "devName": "FieldUnit",
       "rtuIsAlive": false,
       "SeqNumb": 10
       }
}
```

> *When publishing to MQTT, data values are published with corresponding tag name, if configured in ACE, with some exceptions (substitutions) noted below.*
>
> *With MQ-RBE, Sparkplug-B, or MQ-JSON:*
>
> - *If tag name includes a space, the space will be converted to an underscore _ character instead.*
> - *The space-to-underscore conversion also applies to the Ethernet/IP master protocol.*
>
> *With MQ-RBE or Sparkplug-B (not JSON):*
>
> - *If tag name includes a period . it's tag will be published with a forward slash / instead.*
> - *If tag name includes an integer between square brackets for an array (such as [23]), it's tag will be published with the integer surrounded by forward slash and underscore instead (such as /23_).*
> - *In Ignition, the forward slash in a published tag name creates a level in the collapsible tag hierarchy.*

Data Subscription Payload Format

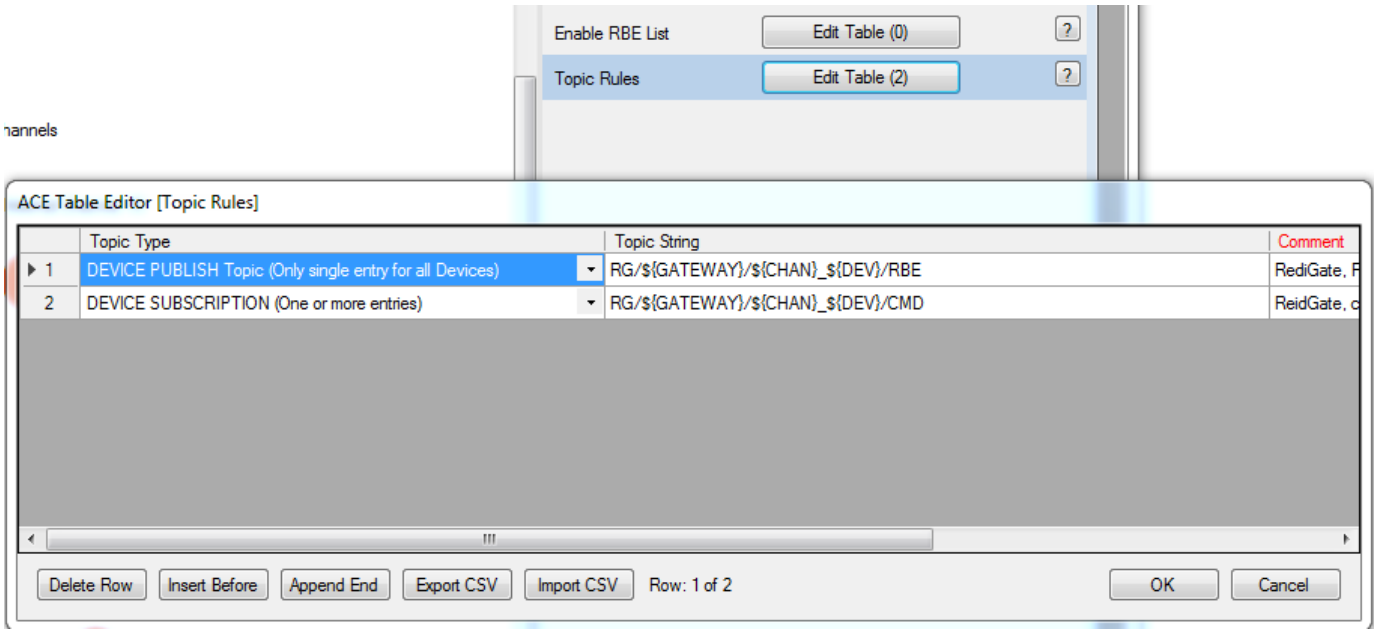
The JSON payload for Device Command messages published from a host to an end device is the same as the payload for the Device Data. Subscriptions for Device Command messages are optional. The Device Command values can either use the tag name (if configured) or register address of the registers to be written to. Only the tags written to need to be included in the Device Command message. The other tags in Device Publish messages (*gwName*, *devName*, *rtuIsAlive*, *SeqNumb*) can be omitted, or they will be ignored by the receiving device, assuming it has no registers configured with those tag names.

An example subscription JSON data payload is:

```
{
   "d": {
   "BoolTag1": false,
   "40002": 123
   }
}
```

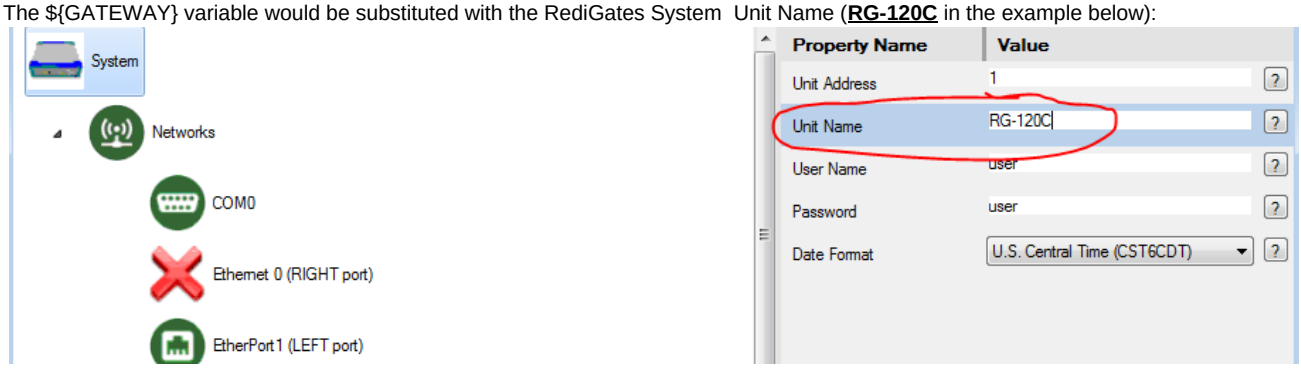
Example JSON Data Subscription

Consider if your RediGate had the following topics configured in a JSON RBE Topic Rules table:

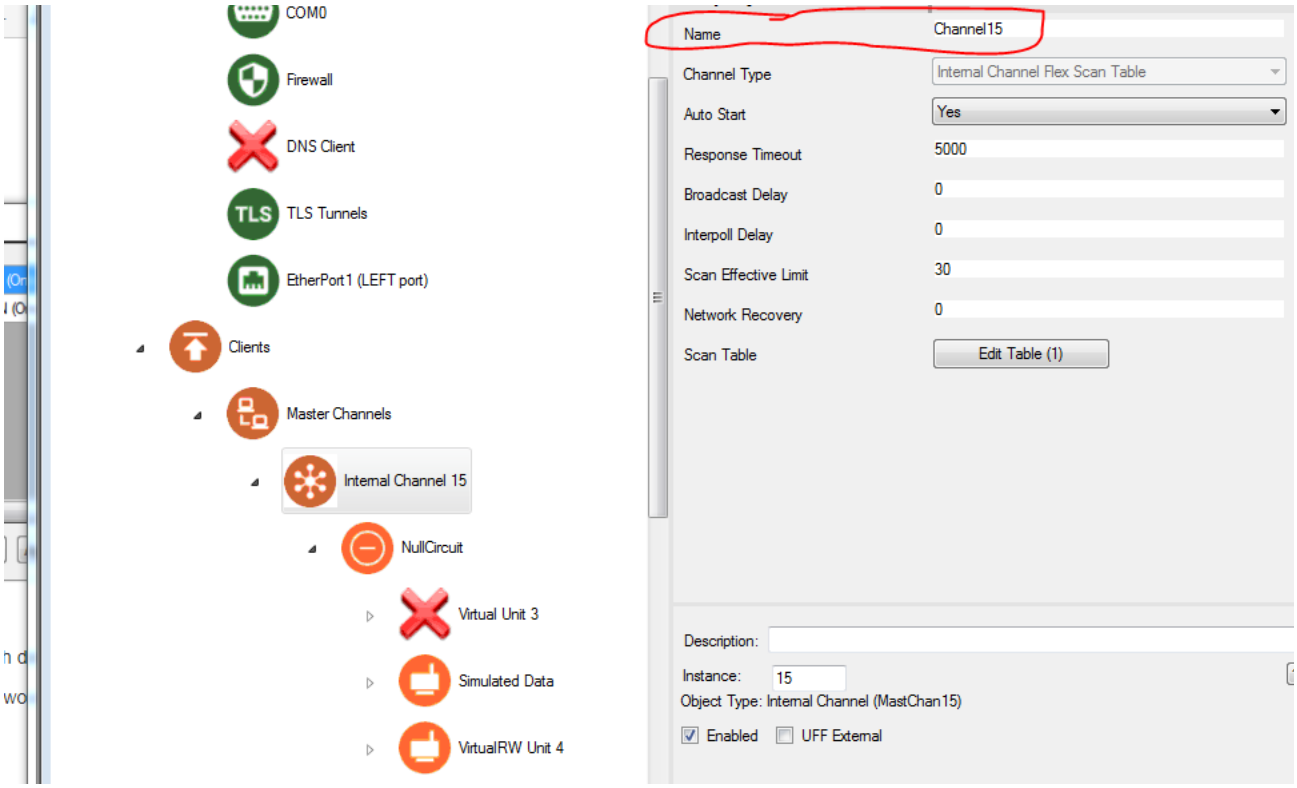


The RediGate would publish data on the RG/${GATEWAY}/${CHAN}_${DEV}/RBE topic with the payload described above in the Data Publish Payload Format section. The topic would be built using the values corresponding to the ${variable}'s, described below:

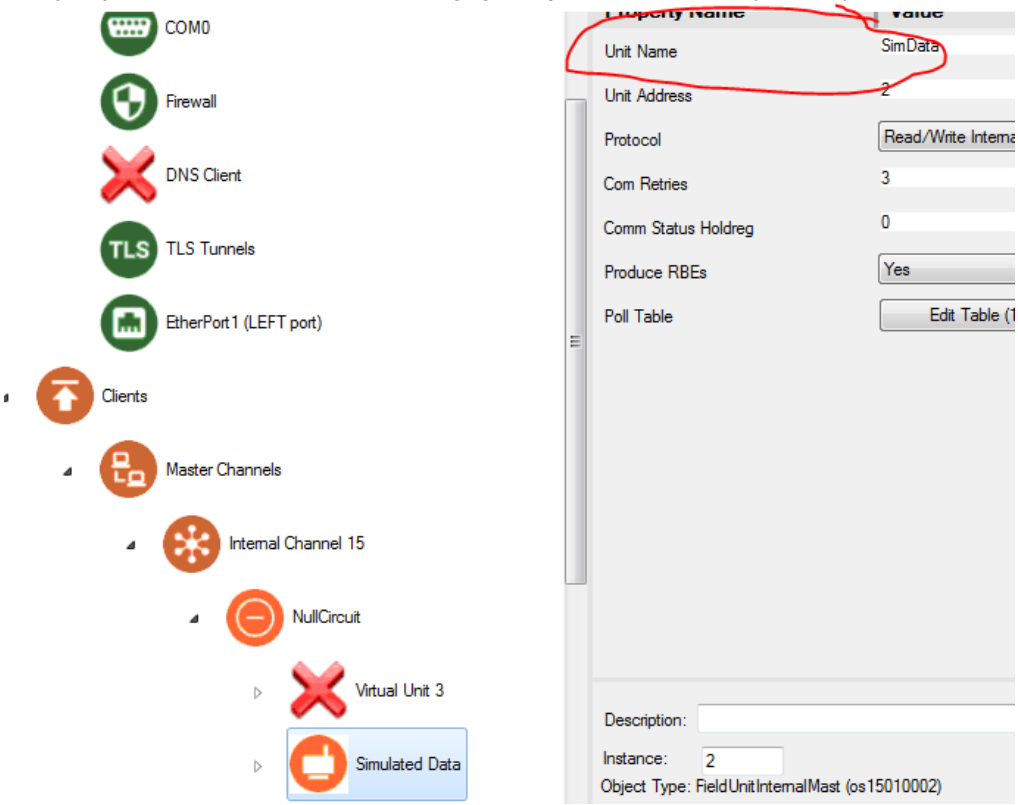
- The ${GATEWAY} variable would be substituted with the RediGates System Unit Name (**RG-120C** in the example below):



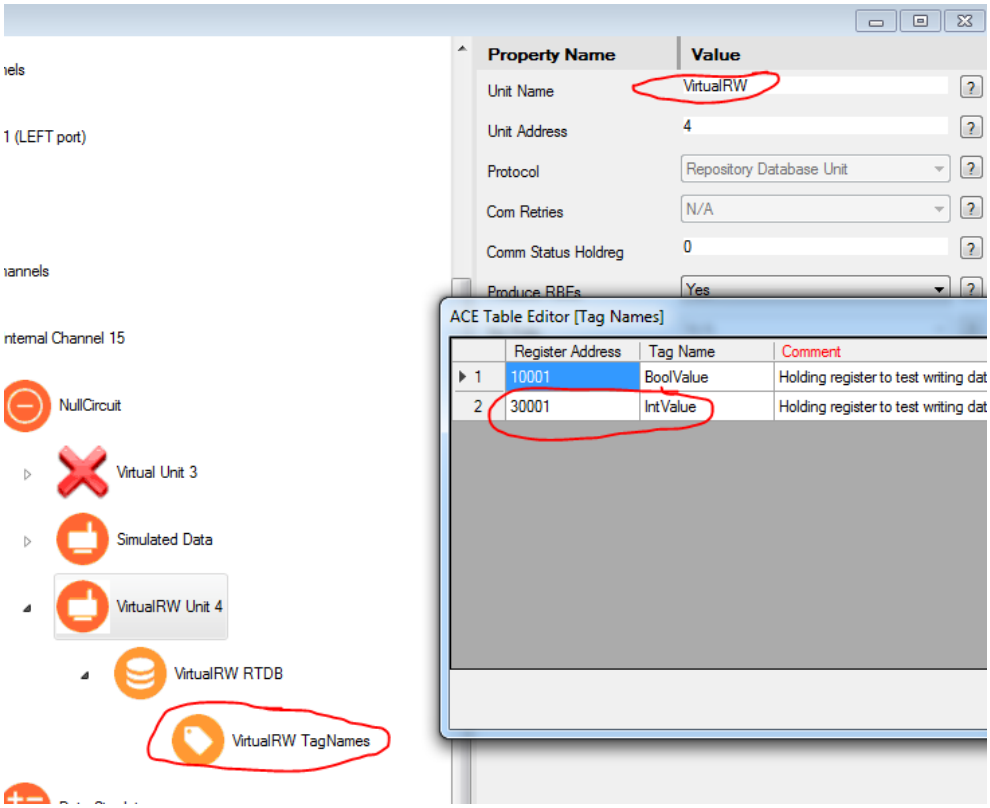
- The ${CHAN} with the channel name of the device with the changing data (since data is only reported on exception) (**Channel15** in the example below):



- The ${DEV} with the device name that had changing data **(SimData** in the example below):



In the example above, if you wanted to publish the integer value "12345" data back to a data register named "IntValue" in VirtualRW Unit 4 (name = VirtualRW):



you would send this payload:

```
{"d": {"IntValue": 12345}}
```

to this topic:

```
RG/RG-120C/Channel15_VirtualRW/CMD
```

Gateway Birth Payload Format

When the gateway first connects to the MQTT broker, it will publish its birth message with sequence number of zero. This includes a number of “metrics” with gateway system information, such as those listed below:

```
{
       "d": {
       "gwName": "GatewayName",
       "Tarball_Date": "2017-06-16-0900",
       "MQttBroker_IP": "127.0.0.1",
       "MQtt_NumbConnects": 2,
       "Numb_Devices": 3,
       "Device[0]_Name": "Modbus-1",
       "Device[1]_Name": "Virtual",
       "Device[2]_Name": "VirtualRTU7",
       "ACE_Config_XML_Info": "469d14e029131bf930b0147175206344
/usr/director/config/BlueMix-JSON_Demo1.xml.gz",
       "ACE_Config_UFF_Info": "8ce078a1a3360e9ea5d2cd7768193c11 /usr/director/config/RG-110E.uff",
       "Gateway_Uptime": " 00:00:21 up 38 min, load average: 1.33, 1.00, 0.77",
       "Free_Memory": "Mem: 253988 25296 228692 0 0",
       "Disk_Usage": "ubi0:rootfs 57476 21716 35760 38% /",
       "Gateway_Time": "1999-11-30-00:00:21.896",
       "MQtt_Msgs_Recv": 3,
       "MQtt_Msgs_Sent": 3,
       "Connection": "ONLINE",
       "SeqNumb": 0
       }
}
```


Gateway Subscription Payload Format

The gateway may subscribe to system service requests using a Gateway Subscription topic. These subscriptions are optional, but are needed in order to receive gateway-level system commands or device-level commands. A host application may send a command to the RediGate using a JSON packet with integer value for SystemCommand. For example:

General System Command


```
{
       "d:" {
       "SystemCommand": 2
       }
}
```


(or as a single line, with white space removed)


```
{"d:"{"SystemCommand":2}}
```


The value of the *SystemCommand* tag should be:

1 = Gateway reboot.

2 = Resend all data from all devices – issue full gateway birth and device data update, but do not reset *SeqNumb* to zero.

6 = Set system time (include *SystemTime* variable; see below).

7 = Next MQTT server address – disconnect from current IP address and use the next address in the list, or reconnect to the same server if only one is configured.

9 = Force gateway reconfigure – stop gateway software, check for software or configuration update, and restart (may result in a full gateway reboot).

Payload for System Command 6 (set system time)


```
{
       "d:" {
       "SystemCommand": 6,
       "SystemTime": "2017-06-26T17:24:36.162Z"
       }
}
```


The *SystemTime* variable should be set to UTC time in ISO-8601 format, as shown. In NodeRED, the methods .toJSON() or .toISOString() will convert Date() to ISO format.

Historical Data Payload

When using the store and forward feature on the RediGate, the JSON-RBE object will publish the historical data in a different format to distinguish the historical data from the normal "report-by-exception" data. Data is published based on the RTU name (or devName), and is organized underneath timestamps corresponding to when the data was stored. The payload below shows a historical payload that was published for a gateway named "RediGate120E" that contained an RTU/Device named "RemoteDevice1":

JSON-RBE Historical Payload Example


```
{
"gwName":"RediGate120E",
"devName":"RemoteDevice1",
"SeqNumb":0,
"h": {

"2017-09-06T14:17:47.000Z": {
"Second": 46,
"SinFunc": -0.4440860
},
"2017-09-06T16:22:05.000Z": {
"Hour": 11,
"Minute": 22,
"Second": 1,
"SinFunc": -0.2794150
},
"2017-09-06T16:22:18.000Z": {
"Second": 16,
"SinFunc": 0.9835880
},
"2017-09-06T16:22:33.000Z": {
"Second": 31,
"SinFunc": -0.6020240
},
"2017-09-06T16:22:48.000Z": {
"Second": 46,
"SinFunc": -0.4440860
},
"2017-09-06T16:23:03.000Z": {
"Minute": 23,
"Second": 1,
"SinFunc": -0.2794150
},
"2017-09-06T16:23:18.000Z": {
"Second": 16,
"SinFunc": 0.9835880
},
"2017-09-06T16:23:33.000Z": {
"Second": 31,
"SinFunc": -0.6020240
},
"2017-09-06T16:23:48.000Z": {
"Second": 46,
"SinFunc": -0.4440860
}
}
}
```