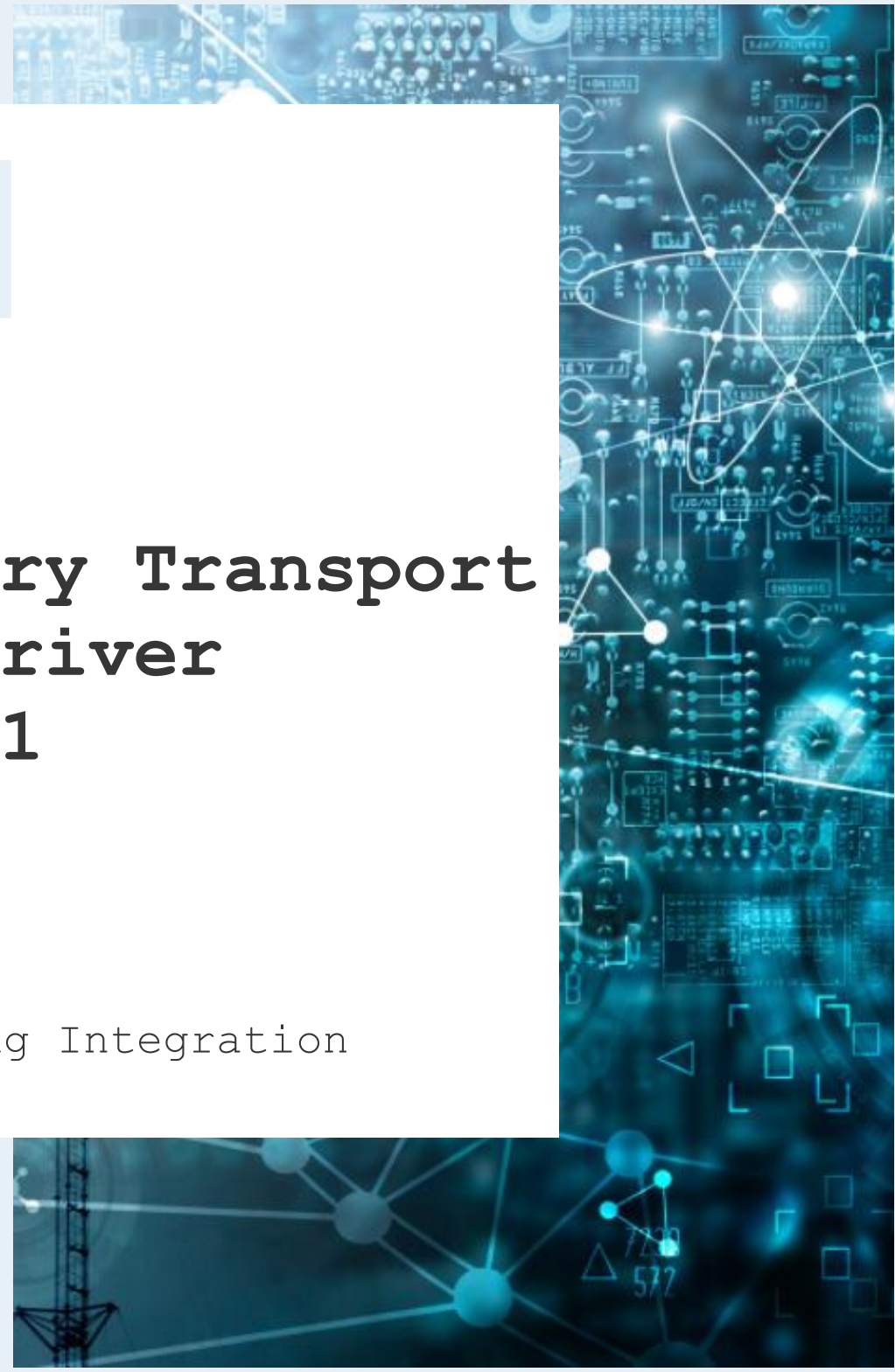
CHIPKIN



MQ Telemetry Transport Ethernet Driver FS-8705-101

Chipkin - Enabling Integration

salesgroup1@chipkin.com

Tel: +1 866 383 1657

Driver Version: 1.0.8
Document Revision: 8

TABLE OF CONTENTS

1 MQTT Description

The MQTT Driver allows the Chipkin QuickServer to transfer data from devices over Ethernet using the MQTT protocol. The MQTT Driver uses TCP. The default port is 1883 and is configurable.

The driver was developed for the streaming MQ Telemetry Transport (MQTT) protocol. MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT). It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth. See https://mqtt.org/ for more information.

The Chipkin QuickServer can emulate both a MQTT Subscriber (Client) and Publisher (Server). When configured as a Subscriber, the MQTT driver will connect to the configured MQTT brokers and subscribe to the configured topics for data. This data is stored on the Chipkin QuickServer to be mapped to other protocols or simply to be viewed. When configured as a Publisher, the MQTT driver connects to the configured MQTT brokers and publishes data received from other protocols to make them available to MQTT Subscribers.

The information that follows describes how to expand upon the factory defaults provided in the configuration files included with the Chipkin QuickServer.

1.1 MQTT Roles and Architecture

MQTT uses a publish/subscribe communication model. Unlike traditional client/server protocols where one device directly requests data from another, MQTT devices exchange information through a central Broker.

Understanding these roles is important when configuring the Chipkin QuickServer

1.1.1 MQTT Broker

The Broker is the central message server in an MQTT system. It accepts connections from clients, receives published messages, and forwards those messages to subscribed clients.

Examples of MQTT Brokers include:

- HiveMQ Public Broker
- Eclipse Mosquitto
- Amazon Web Services IoT Core
- Microsoft

The Broker is responsible for:

- Managing client connections
- Receiving published messages
- Delivering messages to subscribers
- Applying security rules and authentication
- Retaining messages (if enabled)
- Managing Quality of Service (QoS)

Important Note

The Chipkin QuickServer connects to an MQTT Broker as an MQTT client. It does not replace the Broker itself. A Broker must already exist on the network or in the cloud unless one is provided separately.

1.1.2 Publisher

A Publisher sends data to the Broker using a named Topic.

Examples:

- Temperature sensor publishes to: building1/ahu1/temperature
- Alarm system publishes to: site1/alarm/fire
- Chipkin QuickServer publishes values from BACnet or Modbus points

Publishers do not need to know which devices will receive the data.

1.1.3 Subscriber

A Subscriber listens for data by subscribing to one or more Topics.

Examples:

- Dashboard subscribes to building1/#
- Analytics software subscribes to site1/energy/+
- Chipkin QuickServer subscribes to commands or incoming values

Subscribers do not need to know which device published the data.

1.1.4 Topics

A Topic is the message path used to organize data. Topics are text strings separated by forward slashes.

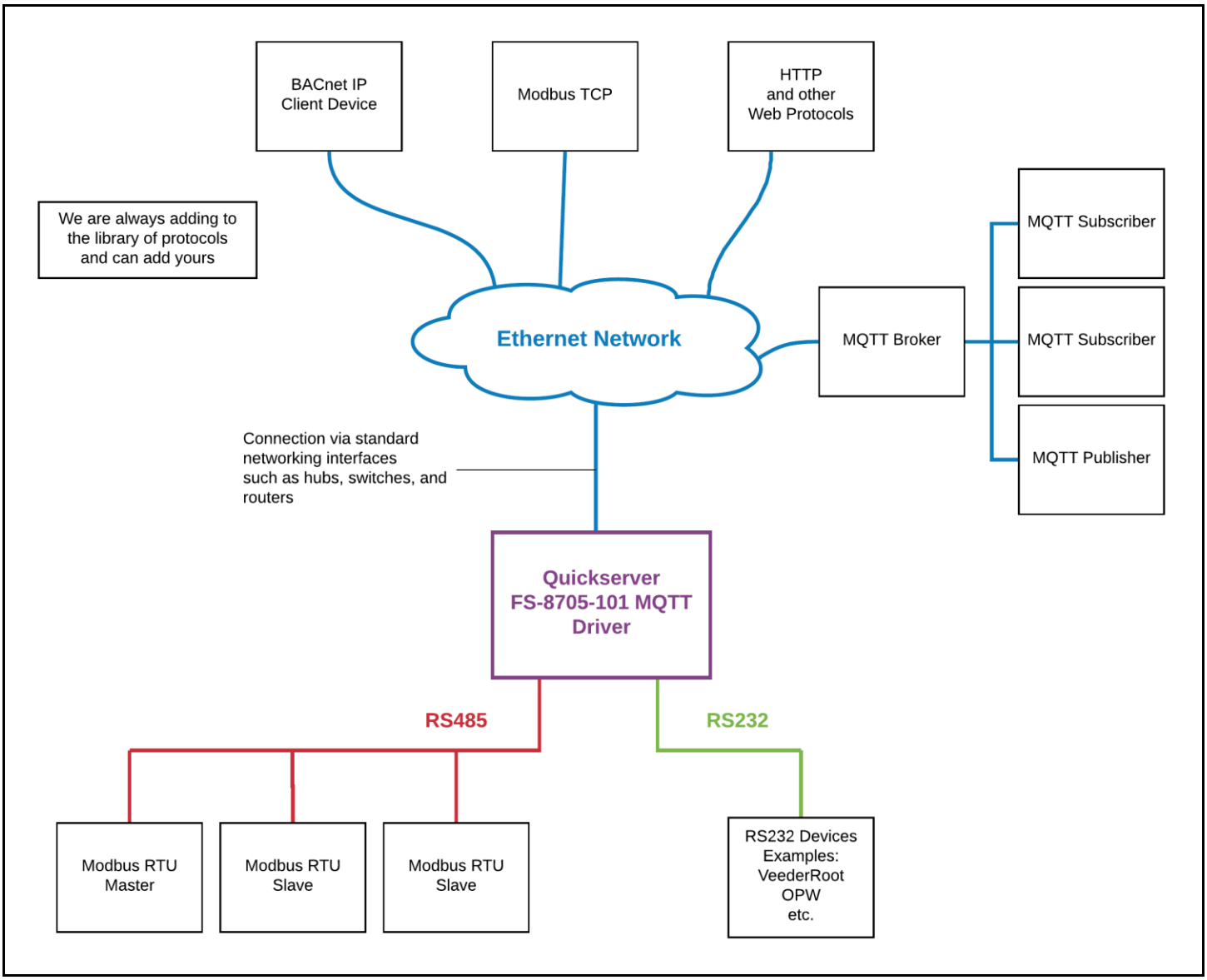
Examples:

- building1/ahu1/temperature
- building1/ahu1/setpoint
- site1/alarm/fire

Topics help organize systems by building, floor, device, or point name.

2 Connection Description

This block diagram lists common network connections that can monitor MQTT data using other protocols like Modbus® RTU/TCP, BACnet® and HTTP.



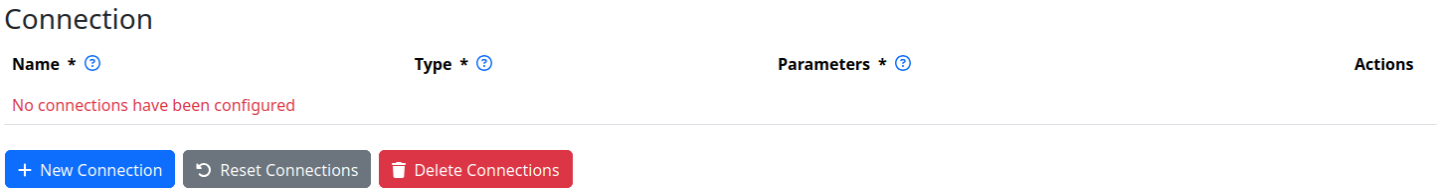
3 MQTT Configuration

To configure the MQTT driver, from the home page, visit the following link:
https://192.168.2.101/chipkinCore/ui/#/driver/mqtt

To configure the Chipkin QuickServer, follow the instructions below to add a Connection (information on how to setup the connection), Nodes (MQTT Broker information), and finally Tasks (subscriptions and publishes).

3.1 Create Connection

To set up the Chipkin QuickServer as a passive MQTT server, first create a connection. The connection contains information about the physical port.

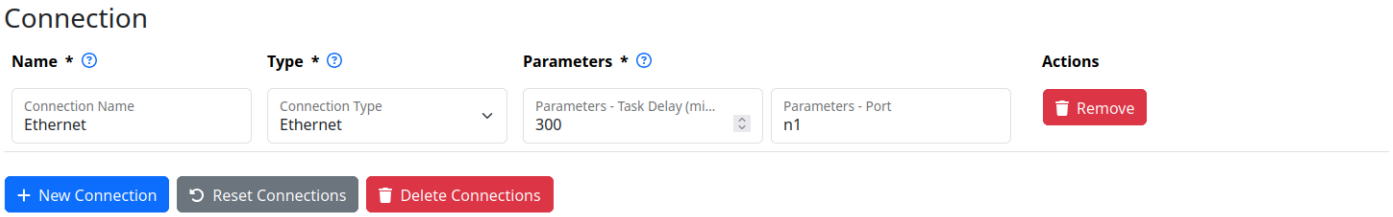


1. Click on the “New Connection” button to generate a new connection row.
2. Fill out the fields in the form. The fields are as follows:

Column Title	Function	Legal Values
Name	Name of the server, used internally as an identifier	Text, must be unique
Type	The type of connection this is	**Ethernet**
Parameters: Task Delay	The task delay in milliseconds used by the driver to wait between tasks. Defaults to 300 milliseconds.	Number - milliseconds
Parameters: Port	The physical port on the Chipkin QuickServer to use	**n1**

* Bolded values are defaults

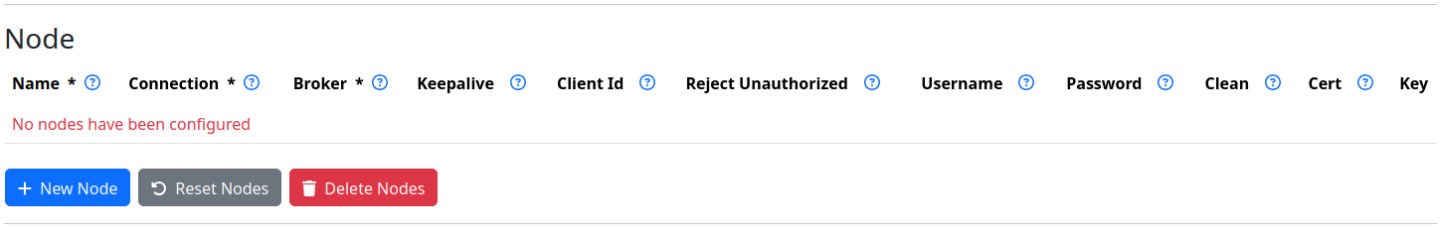
3. Click the “Save” button to add the connection.
   If successful, the new entry will be populated in the Connections table:



4. Reset Connections resets connections to the previously loaded configuration
5. Delete Connections clears the Connections table.

3.2 Create Node

Follow the instructions below to configure the information to connect to a MQTT Broker.



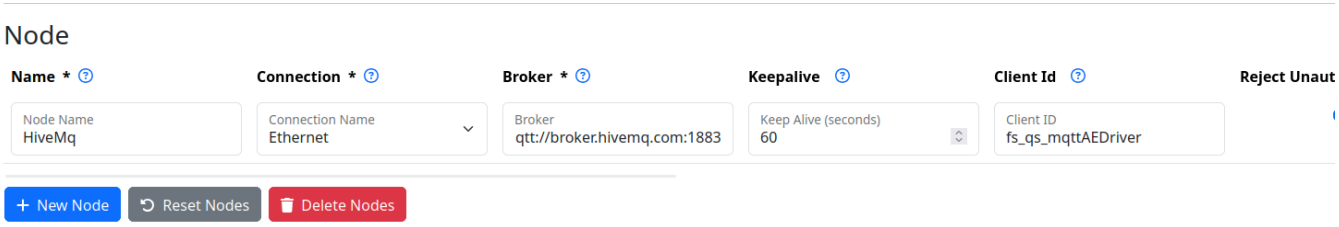
1. Click on the “New Node” button to generate an empty Node.
2. Fill out the fields in the form. The fields are as follows:

Column Title	Function	Legal Values
Name	The name of the broker	Text, must be unique
Connection	The name of the connection to use.	Text (Use the name of the Connection created in the previous section)
Broker	The URL of the MQTT broker in the format of {protocol}://{host}:{port}	Example: mqtt://broker.hivemq.com:1883
KeepAlive	The number of seconds to keep the connection alive, set to 0 to disable	0-3600, **60**
ClientId	The MQTT ClientId that represents this device connecting to the broker	Text, must be unique to the broker, Example: fs_qs_mqttAEDriver
Reject Unauthorized	Reject self-signed certificates. This is a security feature that should remain enabled unless you know what you are doing.	Boolean, **True**
Username	The username required by the broker, if any	Text
Password	The password required by the broker, if any	Text
Clean	Specifies where the connection starts as a new Session or is a continuation of an existing Session	Boolean, **True**
Cert	Paste the full certificate contents, including PEM header and footer lines. This value is stored as plaintext and is visible in diagnostics and exported configuration files. Use caution when sharing diagnostics or exports, and remove sensitive certificate material when appropriate.	Text
Key	Paste the full private key contents, including PEM header and footer lines. This value is stored as plaintext and is visible in diagnostics and exported configuration files. Use extreme caution when sharing	Text

	diagnostics or exports, and remove key material before distribution.	
CA	Paste the full CA certificate contents, including PEM header and footer lines. This value is stored as plaintext and is visible in diagnostics and exported configuration files. Use caution when sharing diagnostics or exports, and remove CA content when needed.	Text

***Note*:** Default values are bolded

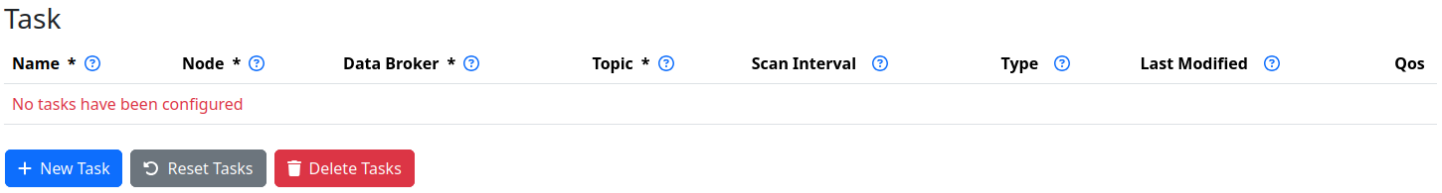
3. Click on the “Save” button to add the node.
4. If successful, the new entry will be populated in the Nodes table:
   Repeat the above steps to add additional nodes.



5. Reset Nodes resets connections to the previously loaded configuration
6. Delete Nodes clears the Nodes table.

3.3 Create Task

Create tasks to add subscriptions to topics or to add publishing tasks to publish data to the specified topic.



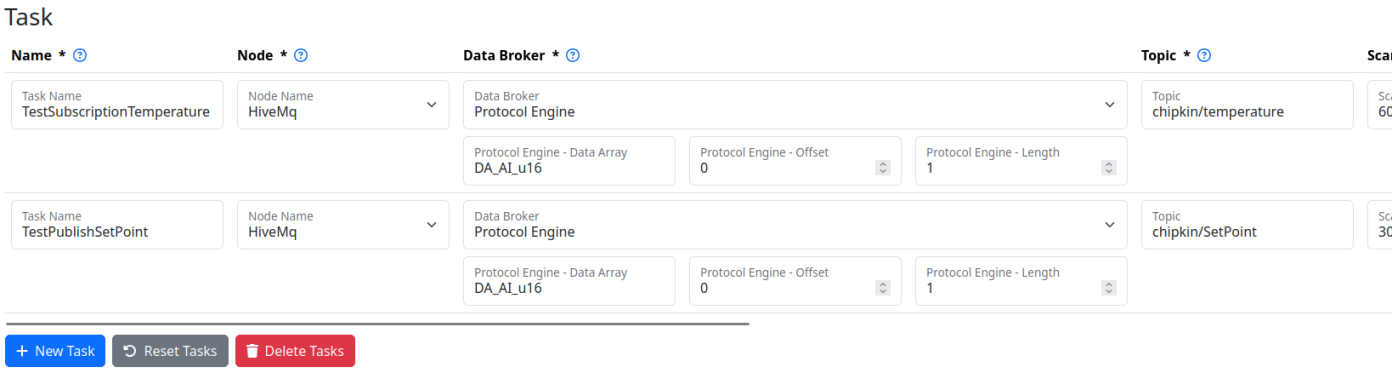
1. Click on the “Create Task” button to open the Create Task form.
2. Fill out the fields in the form. The fields are as follows:

Column Title	Function	Legal Values
Name	The name of the variable to add.	Text, must be unique
Node	The broker to use.	Text (Use the name of a node created in the previous section)
DataBroker	The engine to register values with. Protocol Engine - Data Arrays, used by BACnet and Modbus, and other building automation protocols. Application Engine - A simple Key/Value store, used to store strings and unformatted data.	
DataBroker Protocol Engine: Name	The data array in the protocol engine to retrieve the value.	One of the Data Array names, Example: DA_AI_u16
DataBroker Protocol Engine: Start	The starting offset in the array to retrieve the value	0 to (“Data_Array_length” - 1)
DataBroker Protocol Engine: Length	The length of values to read from the data array (optional)	1 to (“Data_Array_length” - 1)
DataBroker Application Engine: Path	A string provided as a key to a key/value data store.	Text
Topic	The MQTT topic to subscribe to or publish to	Text, must not contain MQTT Topic wildcards
ScanInterval	How often to subscribe or publish data	0-3600, **60**
Type	The type of task this is. Valid values: subscribe, publish	Subscribe, Publish
Last Modified Display Format	Optional - Controls how a task's last modified timestamp is formatted in a **Publish** payload. Defaults to **Unix Epoch** timestamps. Valid Values: **Unix Epoch**, **UTC**, **Custom**	**Unix Epoch**, UTC, Custom Unix Epoch - The Last Modified timestamp in seconds since January 1, 1970 (Unix epoch). Example: **1713877200**

	When **Custom** is selected, the format string is passed directly to Luxons DateTime.toFormat() function, so all Luxon format tokens are supported. Full Luxon formatting documentation: Luxon Formatting Guide Direct reference for supported tokens: Luxon Table of Tokens	UTC - Outputs the Last Modified timestamp as a UTC date/time string. Example: **2026-04-23T19:57:51.394Z** Custom - **Custom**: Outputs the Last Modified timestamp using a custom Luxon DateTime.toFormat() format string. Example: **yyyy-MM-dd HH:mm:ss** becomes **2024-04-23 15:00:00**
QOS	The quality of server	**0**, 1, 2

***Note*:** Default values are bolded

3. Click the “Save” button to add the task.
   If successful, the new entry will be populated in the Tasks table:

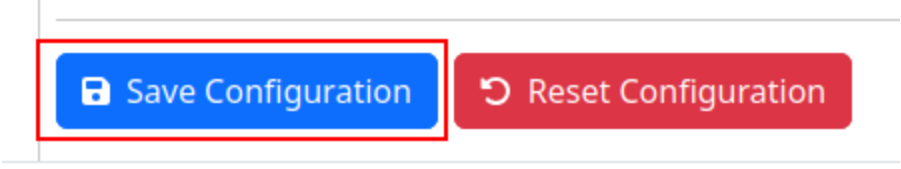


   Repeat the above steps to add additional variables.

4. Reset Tasks resets connections to the previously loaded configuration
5. Delete Tasks clears the Tasks table.

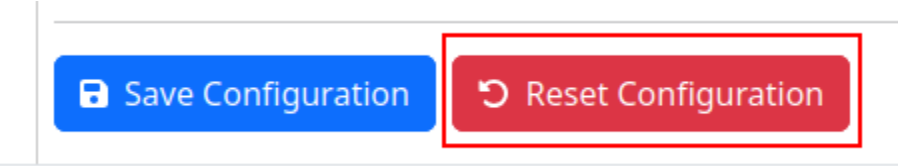
3.4 Saving the Server Configuration

When the configuration is complete, click on the “Save Configuration” button to save all of the updates and changes. For the configuration to take effect, reboot the system.



3.5 Resetting the Server Configuration

To reset the configuration from when the webpage was loaded, click the “Reset Configuration” button. Then follow the instructions in the sections above to create new connections, nodes, and tasks.



4 MQTT Test Tools

A list of MQTT testing tools that you can use to test the functionality of your system.

4.1 HiveMQ MQTT Browser Client

The MQTT is a MQTT client that runs in the browser. It can be used to test subscriptions by publishing topics that the Chipkin QuickServer is subscribe to and can also subscribe to topics that the Chipkin QuickServer is publishing to. For more information, see [The Public MQTT Broker by HiveMQ - Check out our MQTT Demo]

5 License

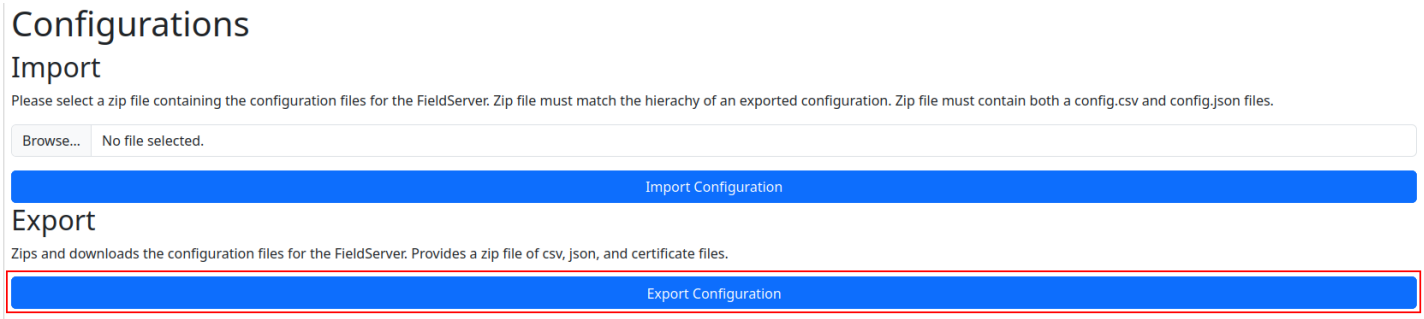
The license system is no longer used for MQTT drivers.

6 Importing and Exporting Configurations

It is possible to export the current configuration to back it up or simply to make some edits. Users can also import either the entire configuration via a zip file or a PE (Protocol Engine) configuration.

6.1 How to Export the Configuration

1. Go to the system configuration page http://{IP_ADDRESS}/chipkinCore/ui/#/system/import-export
2. Click the Export Configuration button.



6.2 How to Import the Configuration

The file to import the configuration must be a zip file. The zip file should contain the following files and folders:

- config.json - this file contains all configurations for the Application Engine
- documents - this folder contains any driver specific documents. For example, license product keys, certificates, etc.
- config.csv - this file contains the configurations for the Protocol Engine.

To make sure the folder directory is correct, do an Export first, then extract the files, edit them, then zip them up again.

To import the configuration:

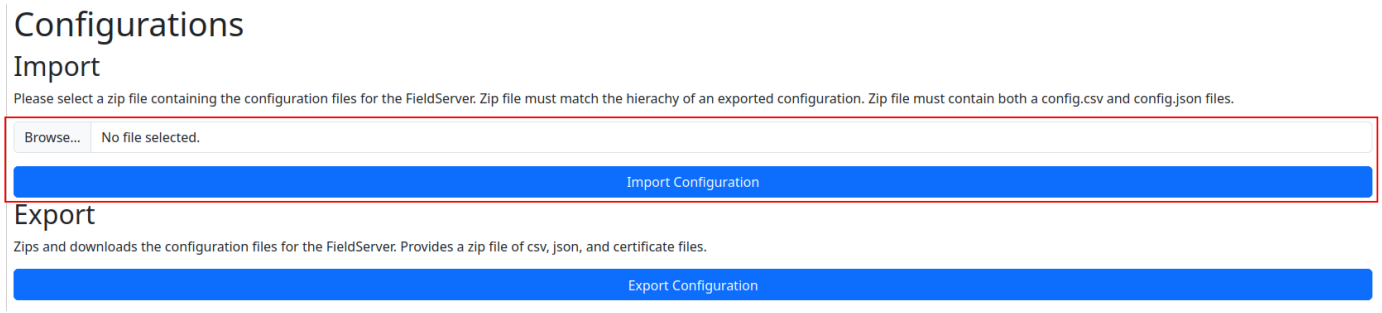
1. Go to the system configuration page http://{IP_ADDRESS}/chipkinCore/ui/#/system/import-export
2. Click the “Browse” button in the “Import/Export System Configuration” section and select the zip file containing the configuration to import.
3. Click the “Import Configuration” button and wait for the configuration to finish importing.
4. If successful, a success message will appear prompting a restart of the Chipkin QuickServer for the changes to take effect.

6.3 How to Import a PE Configuration

It is possible to import a PE (Protocol Engine) configuration separately. To import a PE configuration:

1. Go to the system configuration page http://{IP_ADDRESS}/chipkinConfiguration/ui/

2. Click the “Browse” button in the “Import Specific Configuration” section and select the csv file containing the pe configuration to import.



3. Click the “Import PE Configuration” button and wait for the configuration to finish importing.
4. If successful, a success message will appear prompting a restart of the Chipkin QuickServer for the changes to take effect.
5. Click the Restart Chipkin QuickServer button in the sidebar to restart the Chipkin QuickServer. It will take approximately 2-5 seconds to restart.

7 Revision History

DATE	RESP	DOC. REV.	COMMENT
08 Dec 2020	ACF	1	Created initial document
09 Dec 2020	ACF	2	Removed Firmware Section
02 June 2021	YC	3	Updated document format
27 June 2022	SWS	4	Updated tasks to support length. Removed license section Added payload examples
25 Apr 2026	JJK	5	Updated Tasks section to support differing LastModified times within Publish payloads. Updated pictures Added SSL instructions within Tasks Added NTP section Added Sample Configurations
27 Apr 2026	SWS	6	Added `1.1 MQTT Roles and Architecture` Updated font of code sections Move NTP into its own appendix section General updates to terms, grammar, and spelling
27 Apr 2026	JJK	7	Update Cert/CA/Key node sections to be more descriptive
6 May 2026	JJK	8	Added resyncIntervalHours within ntpSettings section.

Appendix A. Troubleshooting

See this updated troubleshooting guide

https://docs.chipkin.com/articles/mqtt-troubleshooting-guide/

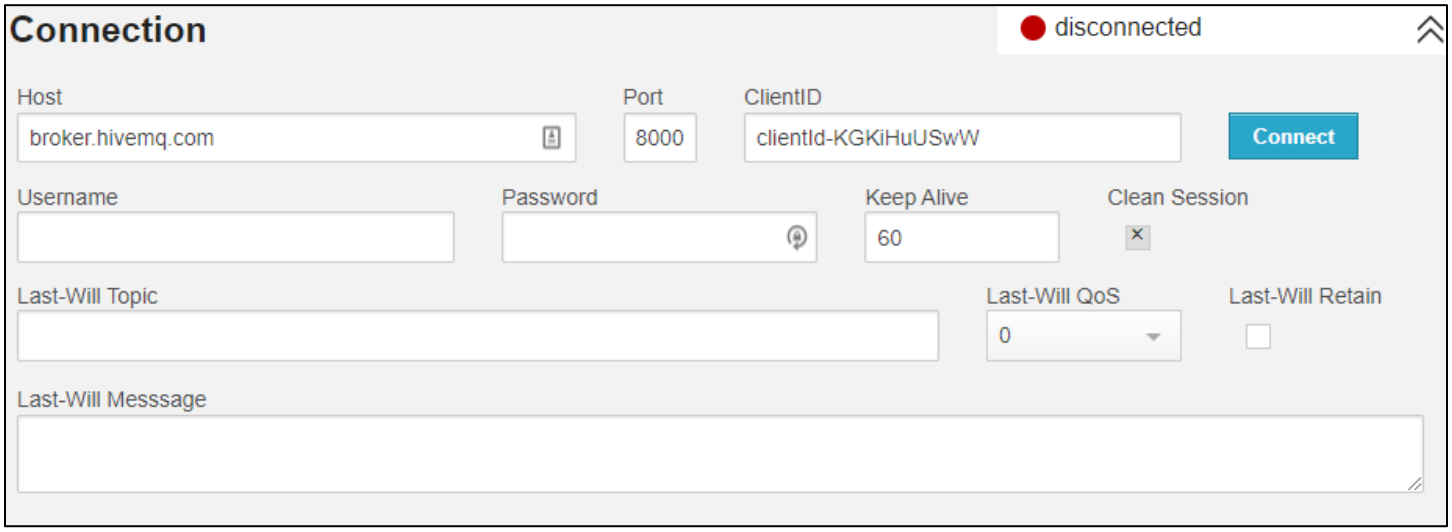
Appendix A.1 Debugging a MQTT connection

- If the Chipkin QuickServer is not receiving any data from subscriptions, verify the MQTT Broker connection is correct and that the topic exists on the Broker.
- If the Chipkin QuickServer is not publishing any data, verify the MQTT Broker connection is correct and check that the data point being published exists.
- Check if the MQTT Broker requires a username and password.
- Verify comms by taking a wireshark log or a Chipkin QuickServer diagnostics log. https://docs.chipkin.com/articles/support-reporting/

Appendix A.2 Using HiveMQ MQTT Browser Client for Testing

Follow the steps in this section to setup the HiveMQ MQTT Browser Client tool to test MQTT devices.

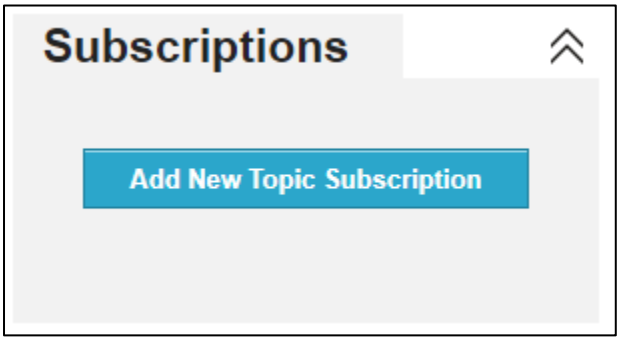
1. Access the Browser Client: MQTT Websocket Client (hivemq.com)
2. Fill out the Broker Connection information:



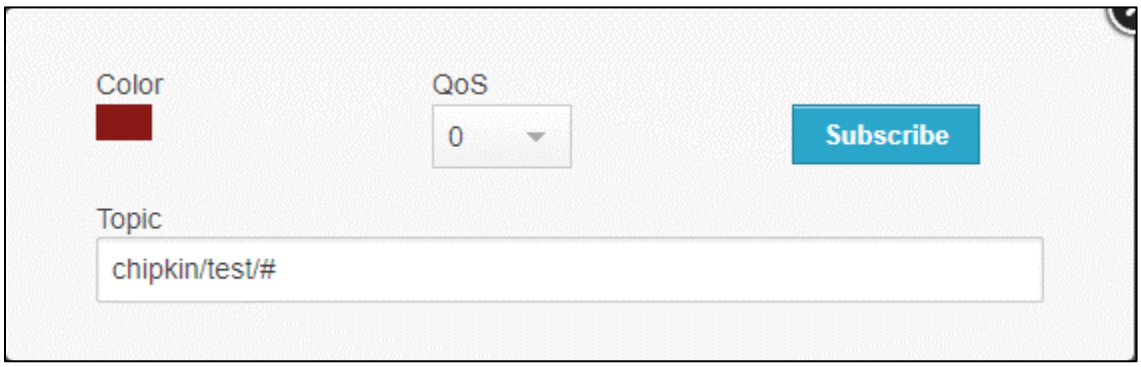
- **Host:** Fill out the host name of the MQTT Broker. If using the example broker, use broker.hivemq.com
- **Port:** This tool uses websockets, so connect using port 8000.
- Keep all the other options with their default values.

Finally click the “Connect” button to connect to the Broker. If successful, you will see the following:

3. Setup a Subscription or Publish:
    a. To setup a subscription:
        i. Click the “Add New Topic Subscription” button



        ii. Fill out the form to subscribe to a topic:

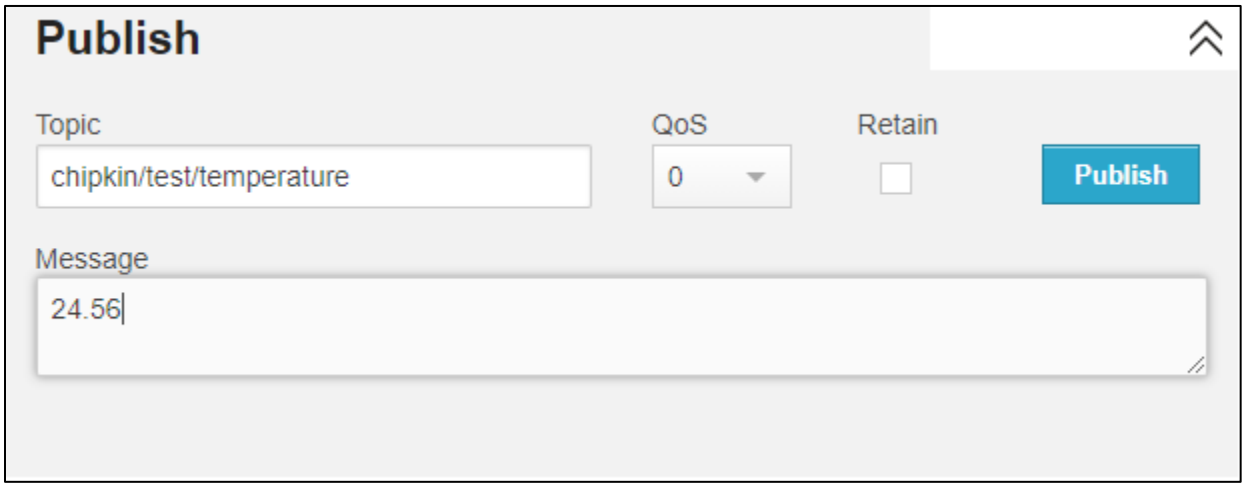


- **Color:** Choose a color to represent the subscription.
- **QoS:** Select the Quality of Service. If unsure, select 0.
- **Topic:** Add the topic to subscribe to. If testing all points, you can use wildcards. In the image above, this will record all data points published to chipkin/test/{any topic}. Otherwise, specify the specific topic, for example: chipkin/test/SetPoint.

Click the “Subscribe” button to add the subscription. New data points will appear in the Messages section:

b. To publish to a topic:

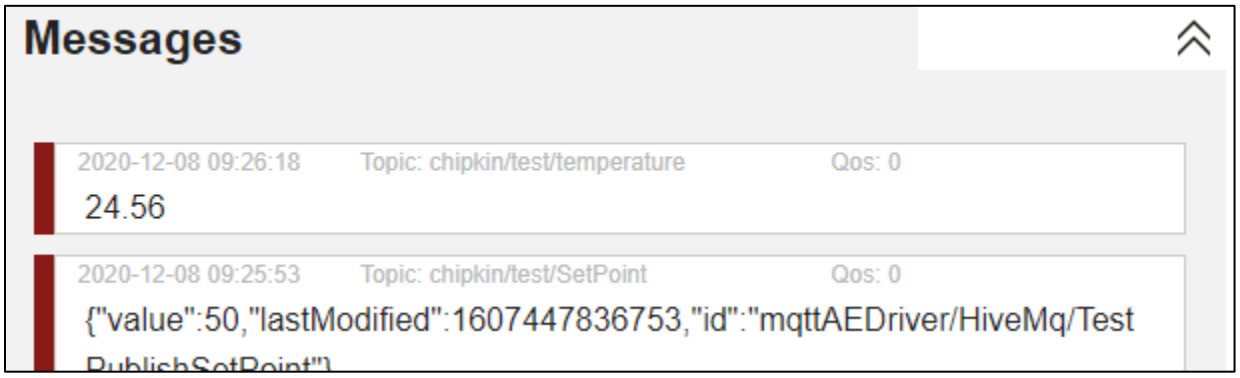
i. Fill out the Publish form:



- **Topic:** The topic to publish to (cannot contain wildcards)
- **QoS:** Select the Quality of Service, set to 0 if unsure.
- **Message:** The data value to publish.

When done, click the “Publish” button.

If the client has been setup with a subscriptions for the same topic that you are publishing to, the published message will show up in the Messages section.



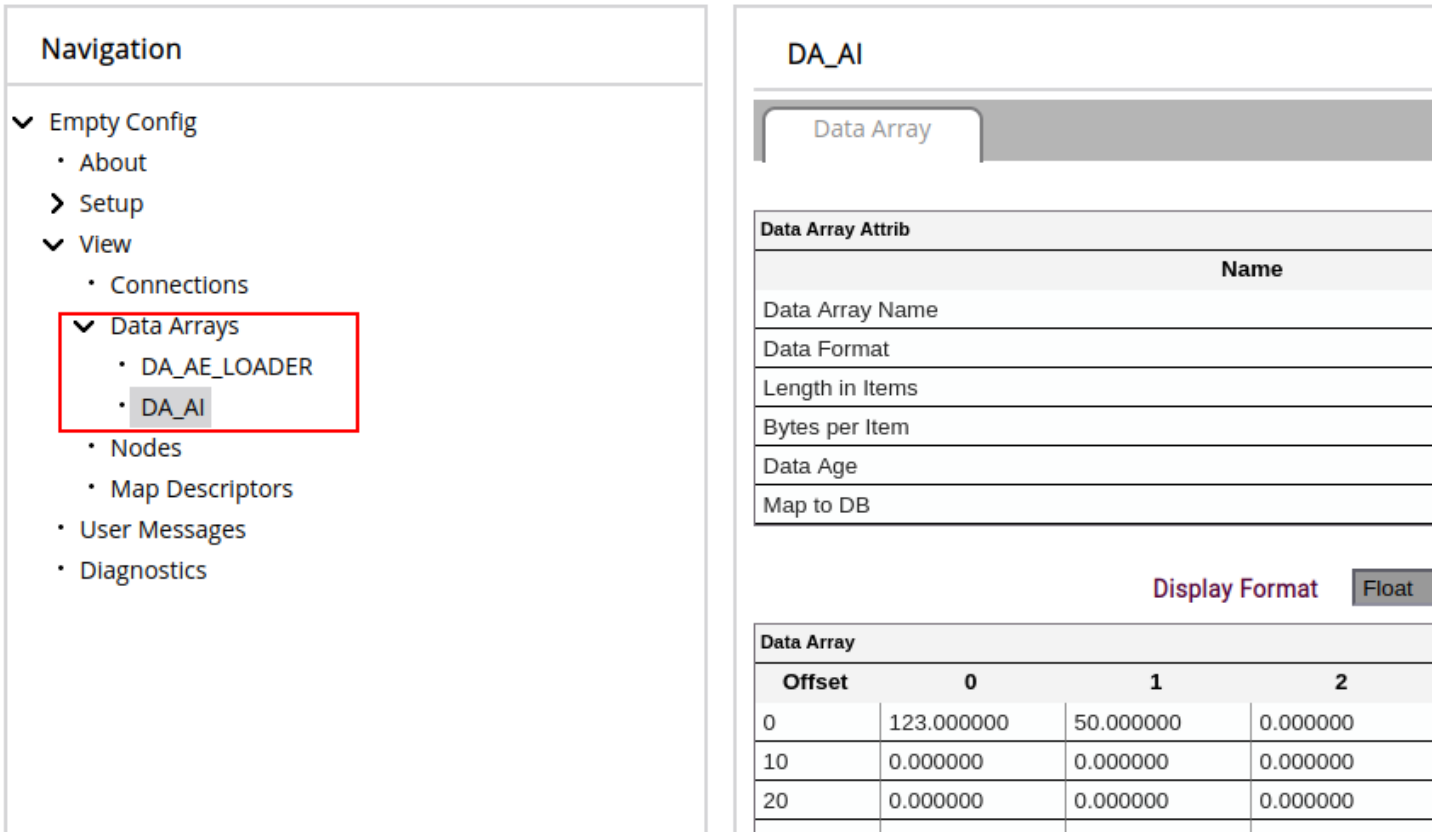
Appendix A.3 Testing Chipkin QuickServer as a MQTT Subscriber

The following instructions are how to confirm that a Chipkin QuickServer that has been configured as a MQTT Subscriber is working correctly.

1. Follow the instructions in Appendix A.2 to use HiveMQ MQTT Browser Client to publish data to the Chipkin QuickServer configured as a MQTT Subscriber.

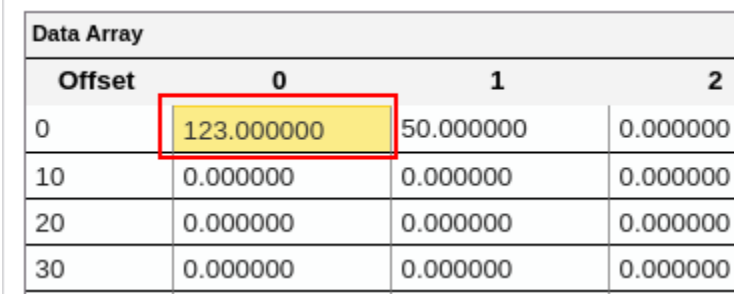
2. Access the Data Array page on the Chipkin QuickServer interface

+-



3. Follow the instructions in Appendix A.2 to publish a value using the HiveMQ MQTT Browser Client.

4. Verify that the values update in the correct Data Array offsets.

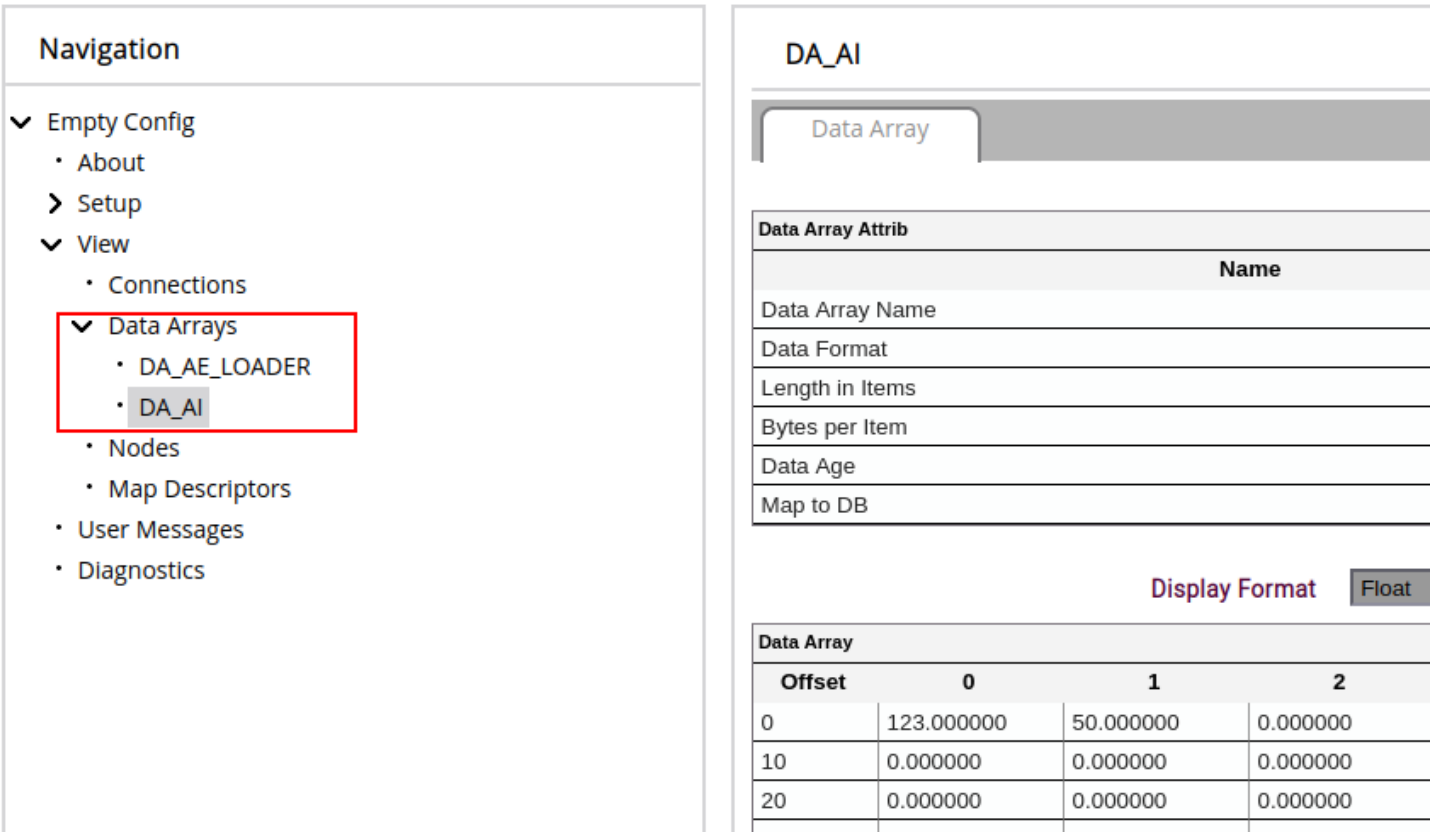


Appendix A.4 Testing Chipkin QuickServer as a MQTT Publisher

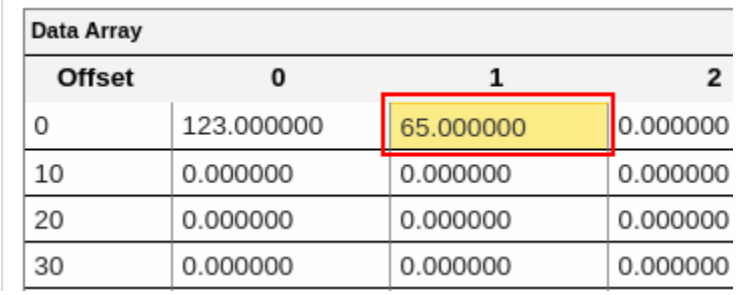
The following instructions are how to confirm that a Chipkin QuickServer that has been configured as a MQTT Publisher is working correctly.

1. Follow the instructions in Appendix A.2 to use HiveMQ MQTT Browser Client to subscribe to a topic that the Chipkin QuickServer is publishing to.

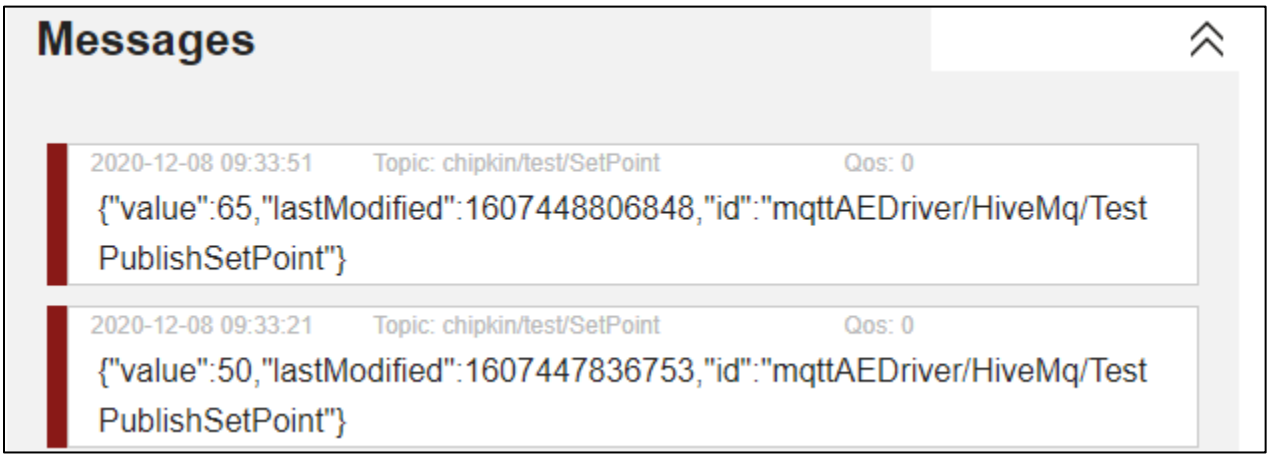
2. Access the Data Array page on the Chipkin QuickServer interface



3. Edit the value in the Data Arrays



4. Verify in the Message section of the HiveMQ MQTT Browser client that the value was published:



Appendix B. Example Configuration

```
{
  "mqttAEDriver": {
    "connections": [
      {
        "name": "Ethernet",
        "type": "ethernet",
        "parameters": {
          "port": "n1"
        }
      }
    ],
    "nodes": [
      {
        "name": "Mq",
        "connection": "Ethernet",
        "broker": "mqtt://broker.hivemq.com:1883",
        "keepalive": 60,
        "clientId": "example_client_id",
        "rejectUnauthorized": false,
        "username": "",
        "password": "",
        "clean": true,
        "cert": "",
        "key": "",
        "ca": ""
      }
    ],
    "tasks": [
      {
        "name": "SubTest",
        "node": "Mq",
        "scanInterval": 30,
        "topic": "chipkin/test/1",
        "type": "subscribe",
        "qos": 0,
        "dataBroker": {
          "pe": {
            "name": "DA_AI",
            "offset": 0,
            "length": 1
          }
        }
      },
      {
        "name": "PubTest",
        "node": "Mq",
```

```
        "scanInterval": 30,
        "topic": "chipkin/test/2",
        "type": "publish",
        "qos": 0,
        "dataBroker": {
          "pe": {
            "name": "DA_AI",
            "offset": 1,
            "length": 1
          }
        },
        "lastModified": {
          "display": "Unix Epoch"
        }
      }
    ]
  }
}
```

Appendix C. Example publish payloads

The example ‘publish’ payloads from the MQTT driver to the MQTT broker.

Single value


```
{
  "value": 0,
  "lastModified": 1656366424332,
  "id": "pe:DA_U16_M1_1/0"
}
```


Multiple values


```
{
  "value": { "0": 1, "1": 2, "2": 3.33, "3": "4.58", "4": 0, "5": 0, "6": 0, "7": 0,
"8": 0, "9": 0, "10": 0, "11": 0, "12": 0, "13": 0, "14": 0, "15": 0, "16": 0, "17":
0, "18": 0, "19": 0, "20": 0, "21": 0, "22": 0, "23": 0, "24": 0, "25": 0, "26": 0,
"27": 0, "28": 0, "29": 0, "30": 0, "31": 0, "32": 0, "33": 0, "34": 0, "35": 0,
"36": 0, "37": 0, "38": 0, "39": 0, "40": 0, "41": 0, "42": 0, "43": 0, "44": 0,
"45": 0, "46": 0, "47": 0, "48": 0, "49": 0, "50": 0, "51": 0, "52": 0, "53": 0,
"54": 0, "55": 0, "56": 0, "57": 0, "58": 0, "59": 0, "60": 0, "61": 0, "62": 0,
"63": 0, "64": 0, "65": 0, "66": 0, "67": 0, "68": 0, "69": 0, "70": 0, "71": 0,
"72": 0, "73": 0, "74": 0, "75": 0, "76": 0, "77": 0, "78": 0, "79": 0, "80": 0,
"81": 0, "82": 0, "83": 0, "84": 0, "85": 0, "86": 0, "87": 0, "88": 0, "89": 0,
"90": 0, "91": 0, "92": 0, "93": 0, "94": 0, "95": 0, "96": 0, "97": 0, "98": 0,
"99": 0, "100": 0, "101": 0, "102": 0, "103": 0, "104": 0, "105": 0, "106": 0, "107":
0, "108": 0, "109": 0, "110": 0, "111": 0, "112": 0, "113": 0, "114": 0, "115": 0,
"116": 0, "117": 0, "118": 0, "119": 0, "120": 0, "121": 0, "122": 0, "123": 0,
"124": 0, "125": 0, "126": 0, "127": 0, "128": 0, "129": 0, "130": 0, "131": 0,
"132": 0, "133": 0, "134": 0, "135": 0, "136": 0, "137": 0, "138": 0, "139": 0,
"140": 0, "141": 0, "142": 0, "143": 0, "144": 0, "145": 0, "146": 0, "147": 0,
"148": 0, "149": 0, "150": 0, "151": 0, "152": 0, "153": 0, "154": 0, "155": 0,
"156": 0, "157": 0, "158": 0, "159": 0, "160": 0, "161": 0, "162": 0, "163": 0,
"164": 0, "165": 0, "166": 0, "167": 0, "168": 0, "169": 0, "170": 0, "171": 0,
"172": 0, "173": 0, "174": 0, "175": 0, "176": 0, "177": 0, "178": 0, "179": 0,
"180": 0, "181": 0, "182": 0, "183": 0, "184": 0, "185": 0, "186": 0, "187": 0,
"188": 0, "189": 0, "190": 0, "191": 0, "192": 0, "193": 0, "194": 0, "195": 0,
"196": 0, "197": 0, "198": 0, "199": 0 },
  "lastModified": 1656366424332,
  "id": "pe:DA_U16_M1_1/0"
}
```

Appendix D. NTP Settings

Time synchronization can be configured with custom NTP servers.
These settings are not available in the UI and must be configured directly in the JSON configuration file. Which can be downloaded using instructions from Section 6 - Importing and Exporting Configurations

You can provide multiple NTP servers. The system tries them in the order listed and stops at the first successful response.
If all configured NTP servers fail, the system falls back to PE time sync (fieldpop bridge time sync).
If PE time sync also fails, the system continues using the current device/system clock.

Accurate time is strongly recommended. An incorrect clock can cause certificate validation failures, MQTT/TLS connection issues, and incorrect timestamps.

Configuration Structure

- ChipkinCore - Root object for Chipkin core runtime settings.
    - NtpSettings - Object containing NTP configuration.
        - Resync Interval Hours – Resync with the NTP servers every N hours. Off by default. Must be greater than or equal to 1 if used, otherwise ignored.
        - Servers - Ordered array of NTP servers to try.
            - Address - Required hostname or IP of the NTP server.
            - Port - Optional UDP port for NTP (default: 123)

Server entries are evaluated sequentially from top to bottom. Place your preferred/local gateway NTP server first to minimize startup delay and improve reliability.

D.1.1. Example

```
{
  "chipkinCore": {
    "ntpSettings": {
      "resyncIntervalHours": 48,
      "servers": [
        {
          "address": "192.168.2.1"
          "port": "123"
        },
        {
          "address": "0.pool.ntp.org"
          "port": "123"
        },
        {
          "address": "time.nist.gov"
        }
      ]
    }
  },
  "mqttAEDriver": {
     ...
  }
}
```

Appendix E. Example Configurations

Config.csv

```
//=============================================================================
//
//
// 1.00 2026 Apr 25 JJK Created
//
//=============================================================================*/

//=============================================================================
//
// Notes : Example configuration for MQTT.
//
//
//=============================================================================

//=============================================================================
//
// Common Information
//

Bridge
Title , System_Node_Id ,
MQTT HiveMQ Example ,97,

//=============================================================================
//
// Data Arrays
//

Data_Arrays
Data_Array_Name , Data_Format , Data_Array_Length
DA_AE_LOADER ,FLOAT,200
DA_AI ,FLOAT,200
```

Config.json

```
{
  "mqttAEDriver": {
    "connections": [
      {
        "name": "Ethernet",
        "type": "ethernet",
```

```
      "parameters": {
        "port": "n1"
      }
    }
  ],
  "nodes": [
    {
      "name": "Mq",
      "connection": "Ethernet",
      "broker": "mqtt://broker.hivemq.com:1883",
      "keepalive": 60,
      "clientId": "example_client_id",
      "rejectUnauthorized": false,
      "username": "",
      "password": "",
      "clean": true,
      "cert": "",
      "key": "",
      "ca": ""
    }
  ],
  "tasks": [
    {
      "name": "SubTest",
      "node": "Mq",
      "scanInterval": 30,
      "topic": "chipkin/test/1",
      "type": "subscribe",
      "qos": 0,
      "dataBroker": {
        "pe": {
          "name": "DA_AI",
          "offset": 0,
          "length": 1
        }
      }
    },
    {
      "name": "PubTest",
      "node": "Mq",
      "scanInterval": 30,
      "topic": "chipkin/test/2",
      "type": "publish",
      "qos": 0,
      "dataBroker": {
        "pe": {
          "name": "DA_AI",
```

```
            "offset": 1,
            "length": 1
          }
        },
        "lastModified": {
          "display": "Unix Epoch"
        }
      }
    ]
  }
}
```

Appendix F. Marketing

Appendix F.1 Case Study

A series of case studies for MQTT can be found here

- BACnet/IP to MQTT Multi-Building Cloud Monitoring Case Study https://docs.chipkin.com/case-study/bacnet-ip-to-mqtt-multi-building-cloud-monitoring/

Appendix F.2 Keyword

MQTT, IOT, Broker, Pub/Sub, Subscriber, Publisher

Appendix G. Glossary of Terms

- **MQTT -** Message Queuing Telemetry Transport publish/subscribe protocol.
- **TCP** - Transmission Control Protocol
- **Broker** - Central MQTT server that receives and distributes messages.
- **CA Certificate** - Trusted certificate used to validate secure TLS connections.
- **Client** - Any device or software connected to an MQTT Broker.
- **Client ID** - Unique name used by an MQTT client when connecting.
- **Data Array** - Numeric FieldServer memory table used for mapped values.
- **KeepAlive** - MQTT timer used to maintain active broker connection.
- **Last Modified** - Timestamp indicating when a value last changed.
- **NTP** - Network Time Protocol used for clock synchronization.
- **Publisher** - Device or software that sends MQTT messages.
- **Subscriber** - Device or software receiving MQTT messages.
- **Topic** - Named MQTT message path such as building1/ahu1/temp.
- **UTC** - Coordinated Universal Time standard used in timestamps.