MiddleBridge user manual

Compile from sources

To create the ".jar " program executable, open the Eclipse IDE and right-click on the project. Then click on "Export" as shown in Figure 1.

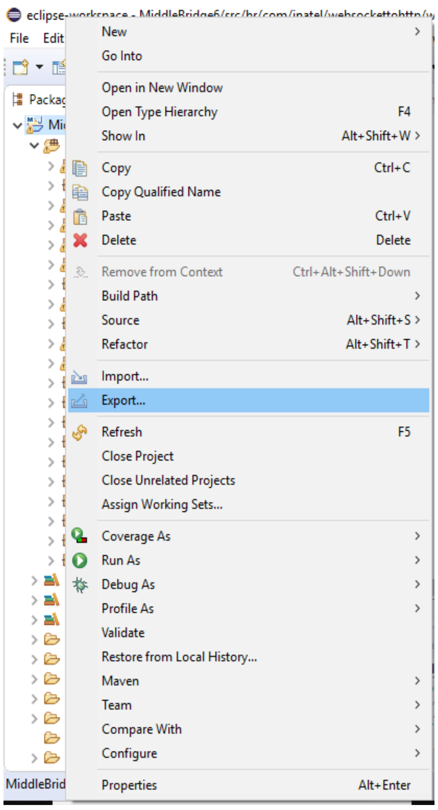
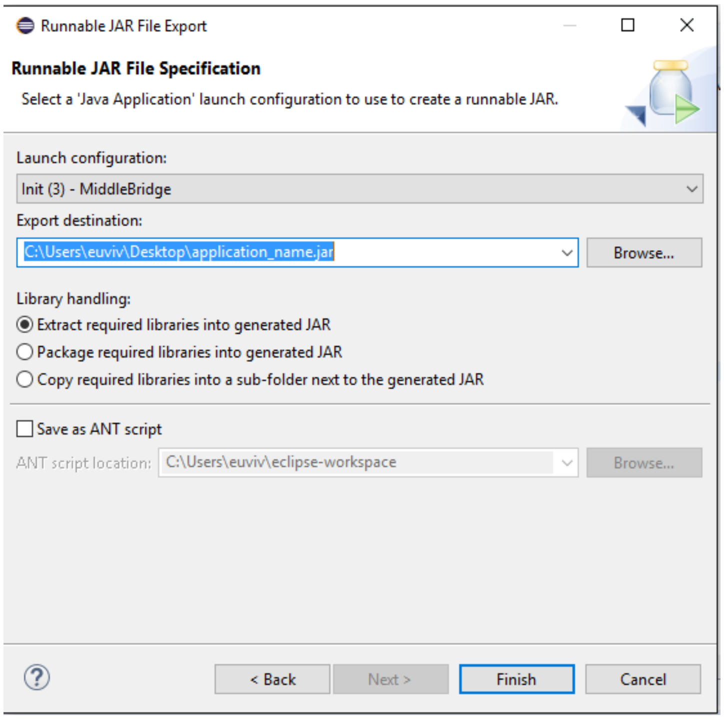


Figure 1 - Export project.

After that, click on "Runnable jar File" and "next". Then, as shown in Figure 2, choose the Init class in Launch Configuration, set the target directory, and click "Finish".


Runnable JAR File Export

Runnable JAR File Specification

Select a 'Java Application' launch configuration to use to create a runnable JAR.

Launch configuration: Init (3) - MiddleBridge

Export destination: C:\Users\euviv\Desktop\application_name.jar

Library handling:

(●) Extract required libraries into generated JAR
( ) Package required libraries into generated JAR
( ) Copy required libraries into a sub-folder next to the generated JAR

[ ] Save as ANT script

ANT script location: C:\Users\euviv\eclipse-workspace

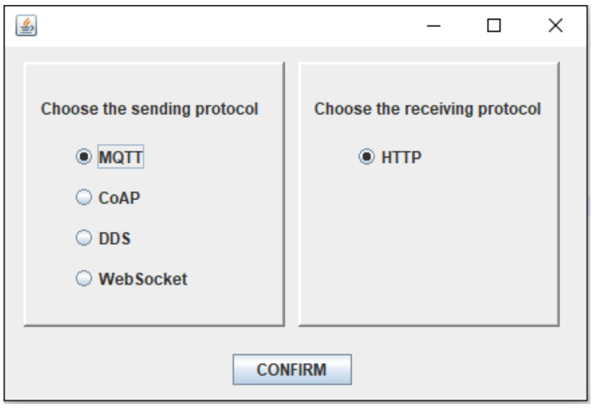
Figure 2 – Choose destination folder.

Start application

To start the application, run it from the command prompt using the following command:

java -jar application_name.jar

Figure 3 shows the initial screen with the protocol options that can be chosen to perform message conversion.


Choose the sending protocol

(●) MQTT
( ) CoAP
( ) DDS
( ) WebSocket

Choose the receiving protocol

(●) HTTP

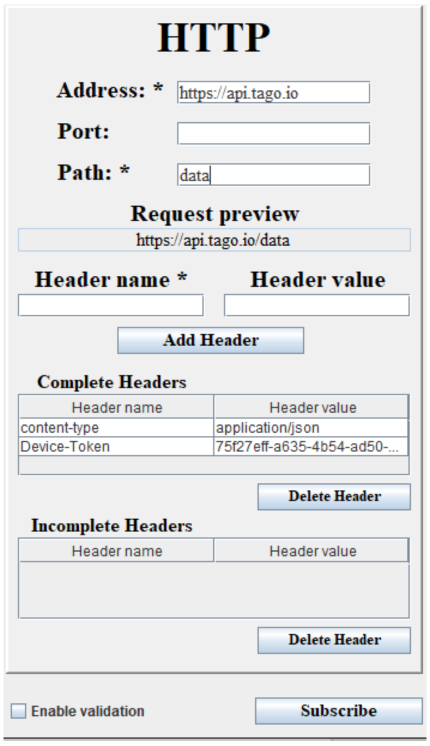
Figure 3 - Choose the protocols.

HTTP

The required HTTP data is the server's address, port, and path. Also, the message header data can be entered completely (name and value) or just the name (the value can be entered in the message).

When validation is enabled, MiddleBridge verifies that the received message size is compatible with the number of payload variables before forwarding the message to the HTTP server. Figure 4 shows an example of using the free Tago server for testing, available at the https://api.tago.io/. On this platform, users can create a new device and associate it with a widget in a Dashboard.

The tutorial on how to create devices in Tago is available at this link: https://tago.elevio.help/en/articles/1-getting-started.



Address: * https://api.tago.io
Port:
Path: * data
Request preview
https://api.tago.io/data
Header name * | Header value

Complete Headers

Header name	Header value
content-type	application/json
Device-Token	75f27eff-a635-4b54-ad50-...

Incomplete Headers

Header name	Header value

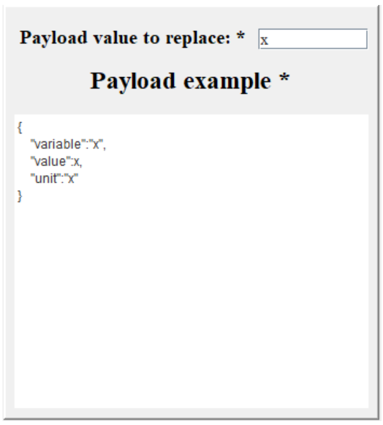
☐ Enable validation

Figure 4 - HTTP data.

As shown in Figure 4, the device token created in Tago must be inserted into the HTTP header in addition to the message type.

Payload example

The payload contains a message template. When the MQTT message arrives at the broker, the message values are docked to the model according to the position of the variable (payload value to replace).



```
{
  "variable":"x",
  "value":x,
  "unit":"x"
}
```

Figure 5 – Payload example.

For this payload model, an example of the MQTT message would be:

temperature,34,C

The payload “**x**” value would be replaced with **"Temperature**", "**34**", and "**C**", respectively.

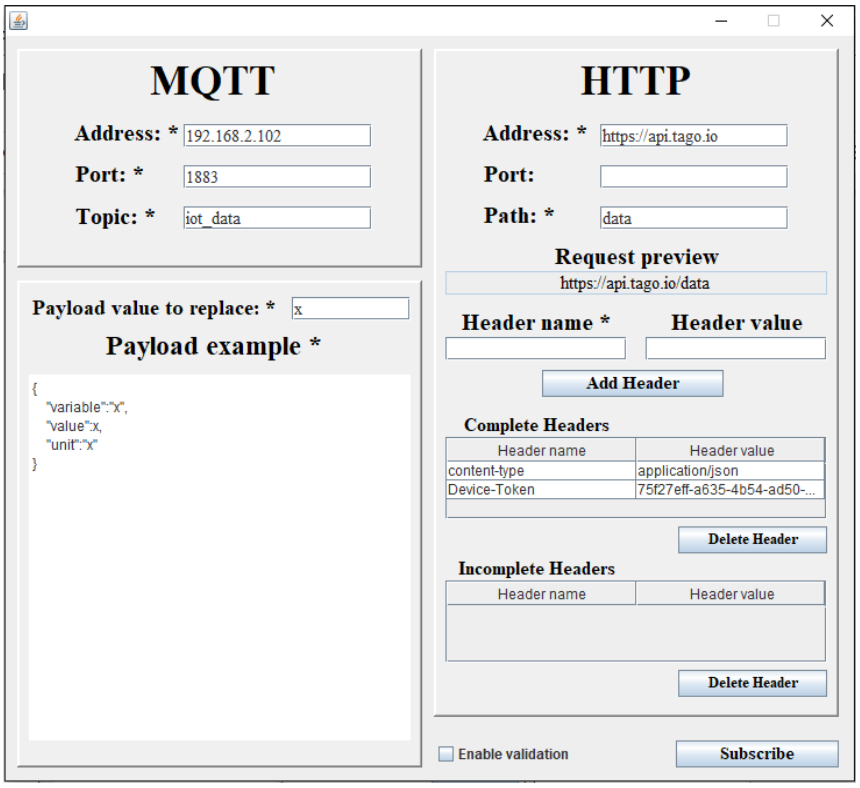
From MQTT to HTTP

To use the Message Queuing Telemetry Transport (MQTT) protocol, provide the IP address of the machine running the program because it will serve as a broker. In addition, insert the communication port and message topic.

The message topic is to identify it. When any device performs a post on a topic, this message is forwarded to the broker, which sends the message to all devices that are subscribed to the same topic.

The TCP/IP port 1883 is reserved with Internet Assigned Numbers Authority (IANA) for use with MQTT. However, any other port can be used as well.

Figure 6 illustrates an example:



MQTT

Address: * 192.168.2.102

Port: * 1883

Topic: * iot_data

Payload value to replace: * x

Payload example *

```
{
  "variable":"x",
  "value":x,
  "unit":"x"
}
```

HTTP

Address: * https://api.tago.io

Port:

Path: * data

Request preview

https://api.tago.io/data

Header name * | Header value

Add Header

Complete Headers

Header name	Header value
content-type	application/json
Device-Token	75f27eff-a635-4b54-ad50-...

Delete Header

Incomplete Headers

Header name	Header value

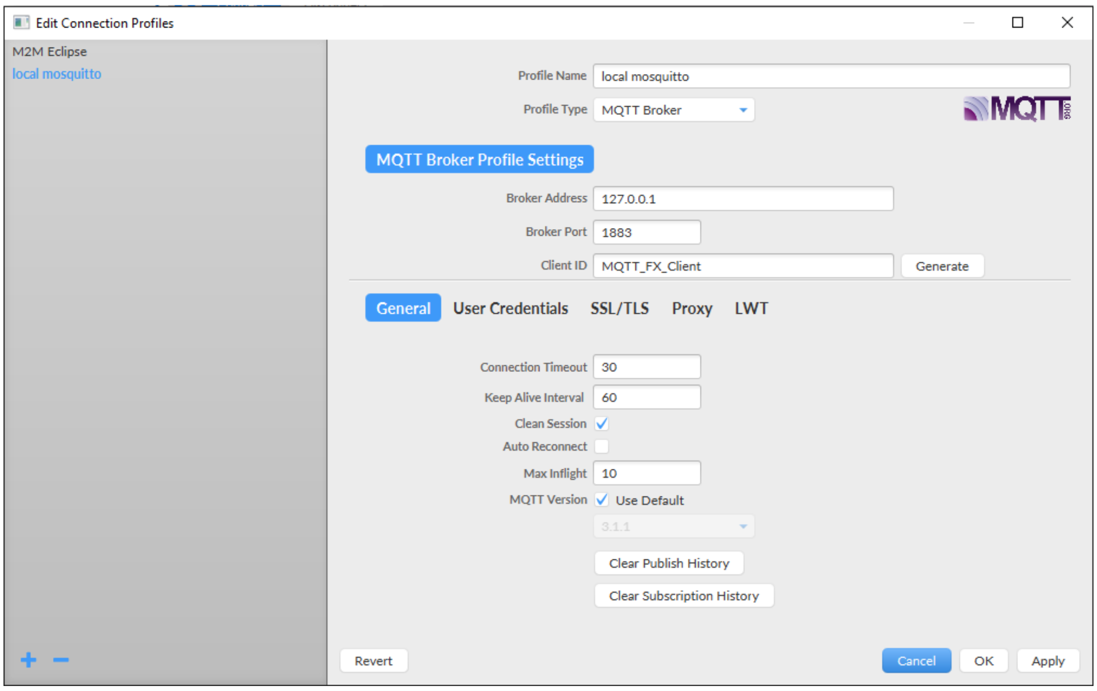
Delete Header

☐ Enable validation

Subscribe

Figure 6 - MQTT example.

The connection is started by clicking Subscribe. This example can be tested by using a program called MQTT.fx, which can be downloaded for free at the https://mqttfx.jensd.de/address. In MQTT.fx, users must initially configure the connection. The same IP address and port entered into the MiddleBridge must also be entered in the MQTT.fx settings. Figure 7 displays an example of MQTT.fx configuration.



Edit Connection Profiles

M2M Eclipse
local mosquitto

Profile Name: local mosquitto
Profile Type: MQTT Broker

MQTT Broker Profile Settings

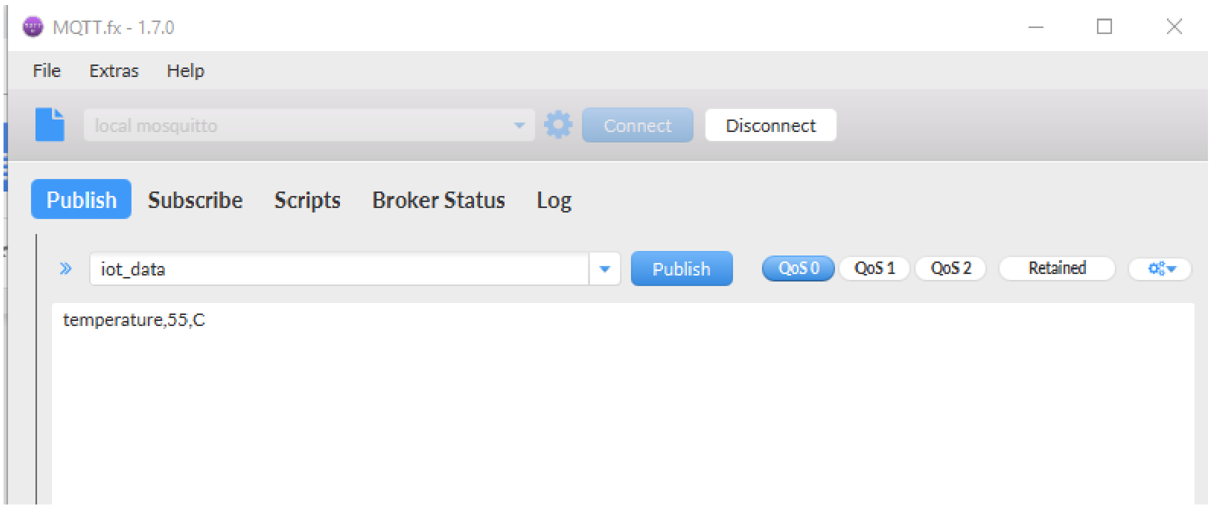
Broker Address: 127.0.0.1
Broker Port: 1883
Client ID: MQTT_FX_Client

General | User Credentials | SSL/TLS | Proxy | LWT

Connection Timeout: 30
Keep Alive Interval: 60
Clean Session: ✓
Auto Reconnect:
Max Inflight: 10
MQTT Version: ✓ Use Default
3.1.1

Figure 7 - Mqtt.fx configuration

After applying the settings above, click Connect. Then, enter the message topic, which in the example is "iot_data", and enter the data to be published according to the message template defined in the payload example. Figure 8 shows an example according to the Tago model: Device variable name (temperature), value (55), and data type (Celcius degrees). Then, publish the message.



MQTT.fx - 1.7.0

iot_data

```
temperature,55,C
```

Figure 8 - Sending MQTT message

The published information is converted to HTTP by the Middlebridge program and can be viewed in the widget associated with the device in Tago, as illustrated in Figure 9.

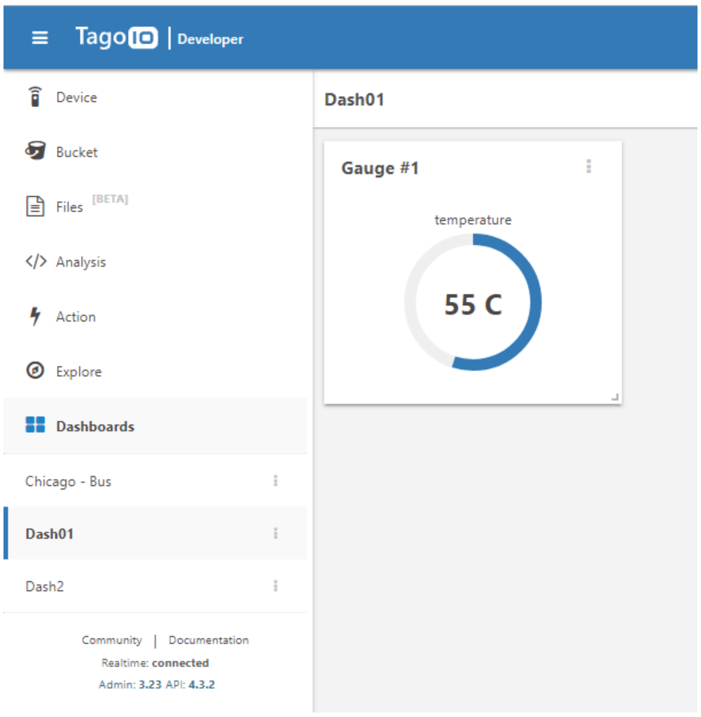


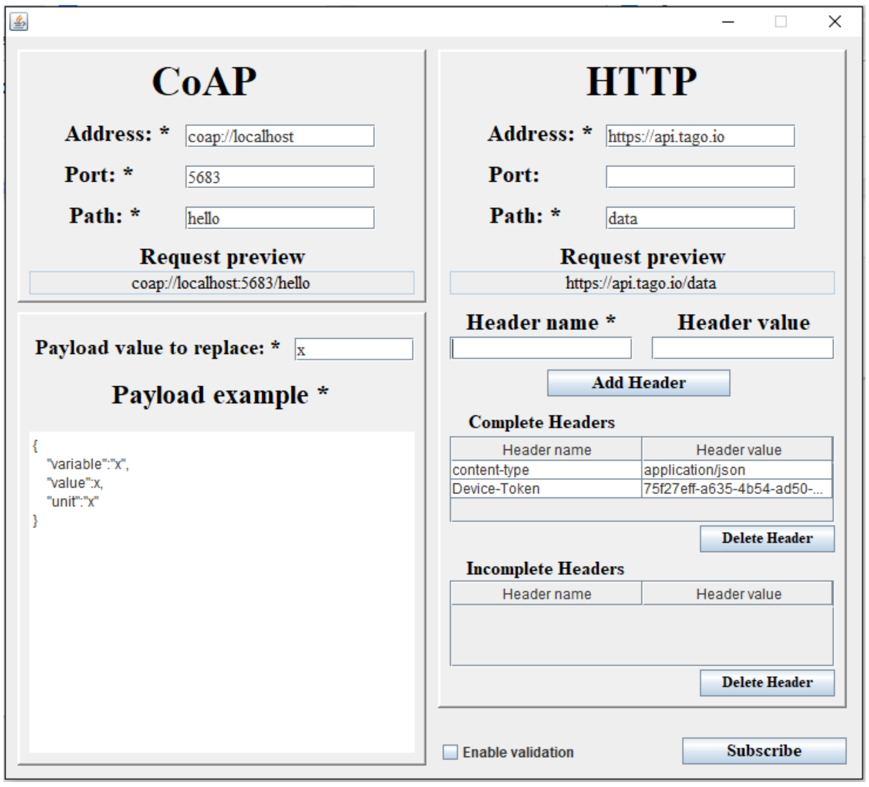
Figure 9 - MQTT message values shown in Tago

More information regarding MQTT can be found in the link https://github.com/mqtt/mqtt.github.io/wiki.

From CoAP to HTTP

Similarly, to the previous protocol, when selecting Constrained Application Protocol (COAP), set the server address, the connection port, and the path of the messages. Servers provide resources in a URI, and clients access these resources using methods such as GET, PUT, POST, and DELETE. The port allows UDP communication between client and server (by default it is used at 5683) and the path identifies the resource within the URI.

Figure 10 shows an example.



CoAP

Address: * coap://localhost

Port: * 5683

Path: * hello

Request preview: coap://localhost:5683/hello

Payload value to replace: * x

```
{
  "variable":"x",
  "value":x,
  "unit":"x"
}
```

HTTP

Address: * https://api.tago.io

Port:

Path: * data

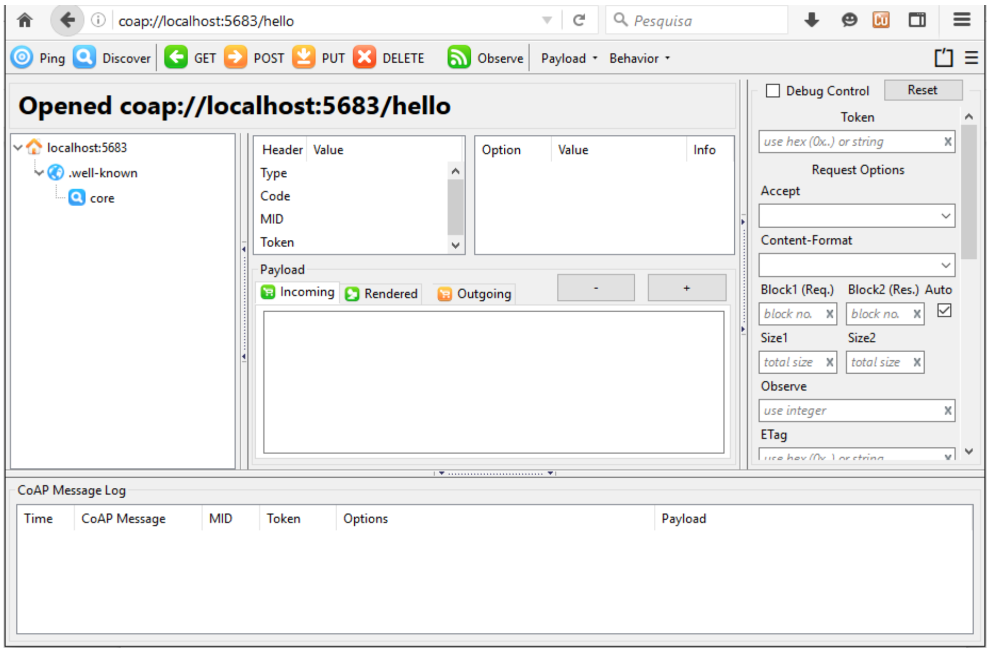
Request preview: https://api.tago.io/data

Complete Headers

Header name	Header value
content-type	application/json
Device-Token	75f27eff-a635-4b54-ad50-...

Figure 10 - CoAP example

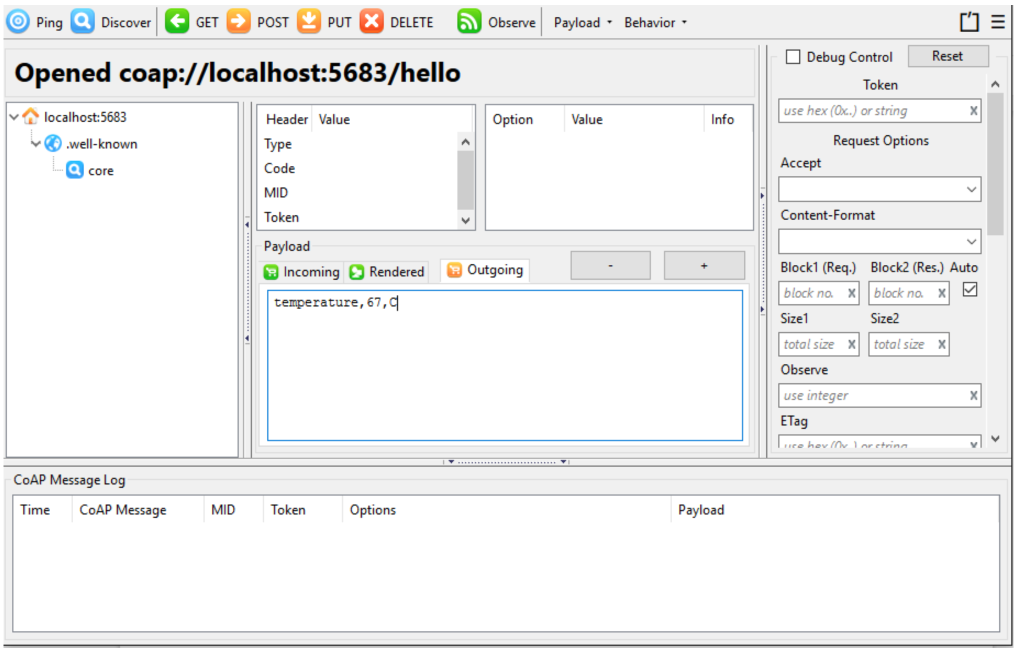
For testing, one must install an extension in the Firefox browser, called Copper (Cu), available only for older versions of the browser, such as version 55.0.3. After that, enter the address Coap://localhost:5683/hello in the browser (note that the port used in this URI is the same as that configured in MiddleBridge). So, the COAP test environment appears as in Figure 11.



Opened coap://localhost:5683/hello

Figure 11 - Firefox extension for CoAP

To send a COAP message using this Firefox tool, click Outgoing, type the message according to the payload model example, and then post.



```
temperature,67,C
```

Figure 12 - Sending CoAP message

In this way, the COAP message is converted to HTTP and sent to the Tago server, as illustrated in Figure 13.

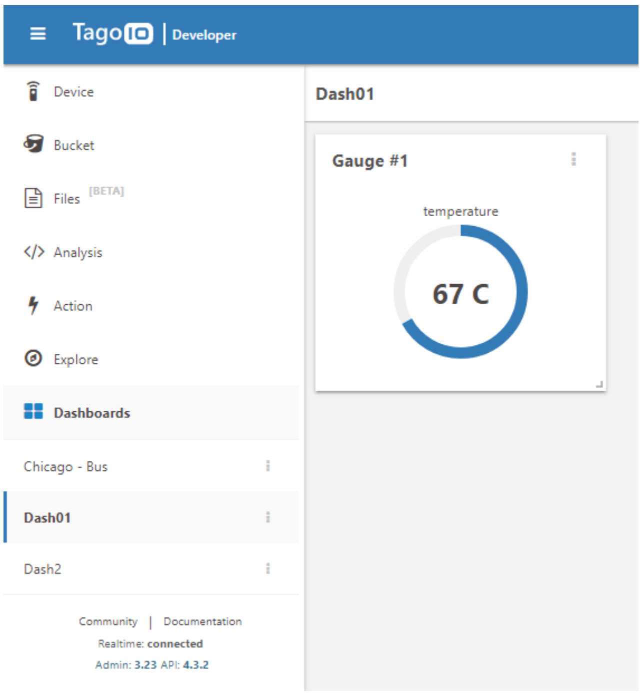
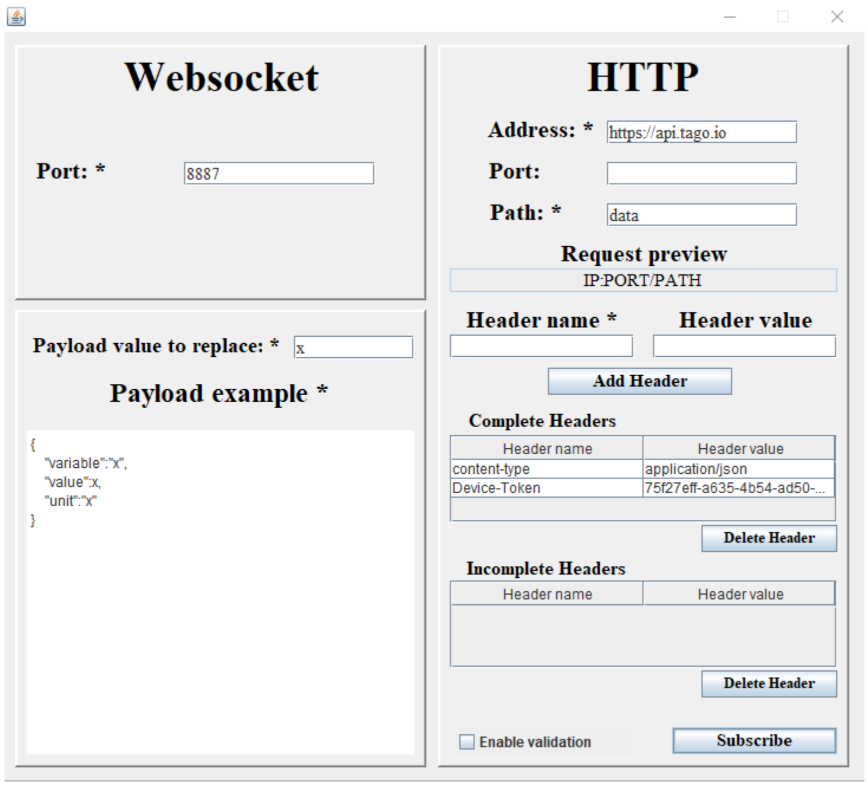


Figure 13 - CoAP message values shown in Tago

In the link https://tools.ietf.org/html/rfc7252, more information regarding the operation of the COAP protocol can be consulted.

From WebSocket to HTTP

To initiate a connection using the WebSocket protocol, configure the server by setting the communication port. Figure 14 shows the example.



Websocket

Port: * 8887

Payload value to replace: * x

Payload example *

```
{
  "variable":"x",
  "value":x,
  "unit":"x"
}
```

HTTP

Address: * https://api.tago.io

Port:

Path: * data

Request preview

IP:PORT/PATH

Header name * | Header value

Add Header

Complete Headers

Header name	Header value
content-type	application/json
Device-Token	75f27eff-a635-4b54-ad50-...

Delete Header

Incomplete Headers

Header name	Header value

Delete Header

Enable validation

Subscribe

Figure 14 - Websocket example

Then, run a client to send the message via chat. In this link there is an application that can be run as a client to perform the tests: https://github.com/TooTallNate/Java-WebSocket.

To run this application, repeat the process of exporting the project, as in Topic 1 from this document (Compile from sources), or run it directly in eclipse. When the program is started, the screen shown in Figure 15 will be displayed. If the port used in the chat is different from the port used in MiddleBridge, modify it to be the same. In the example below, port 8887 is used, as well as the MiddleBridge, shown in Figure 14.

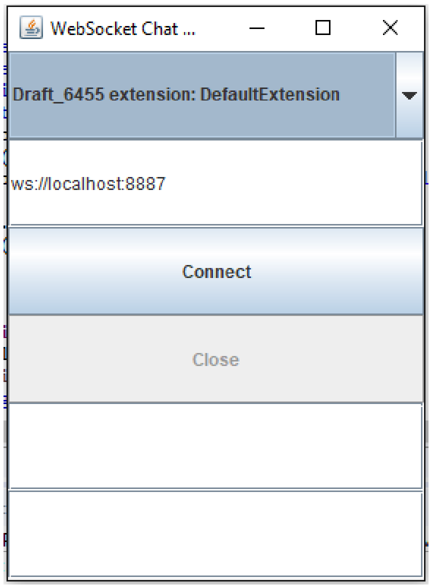
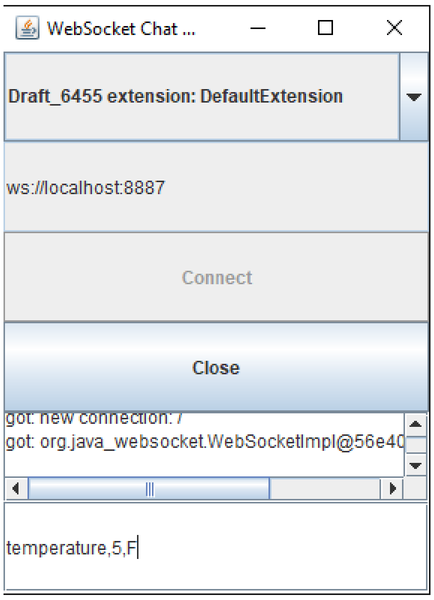


Figure 15 - Chat client Initialization

After starting the application, write the message to be sent to the server, as shown in Figure 16.

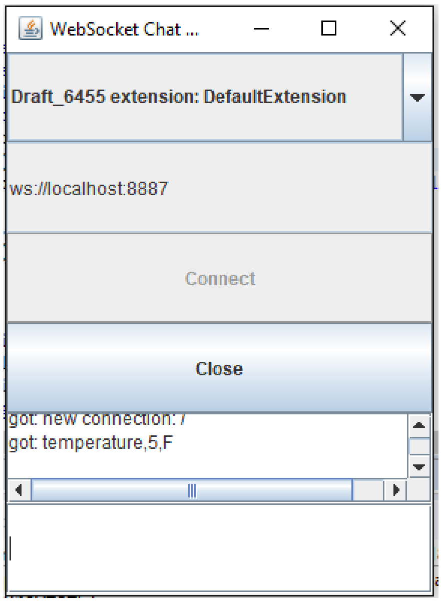


```
got: new connection: /
got: org.java_websocket.WebSocketImpl@56e40
```

```
temperature,5,F
```

Figure 16 - Sending Websocket message

The message is sent by clicking Enter, as shown in Figure 17.



```
got: new connection: /
got: temperature,5,F
```

Figure 17 - Message sent

Soon after, it is possible to check in Tago the message that was sent by the client.

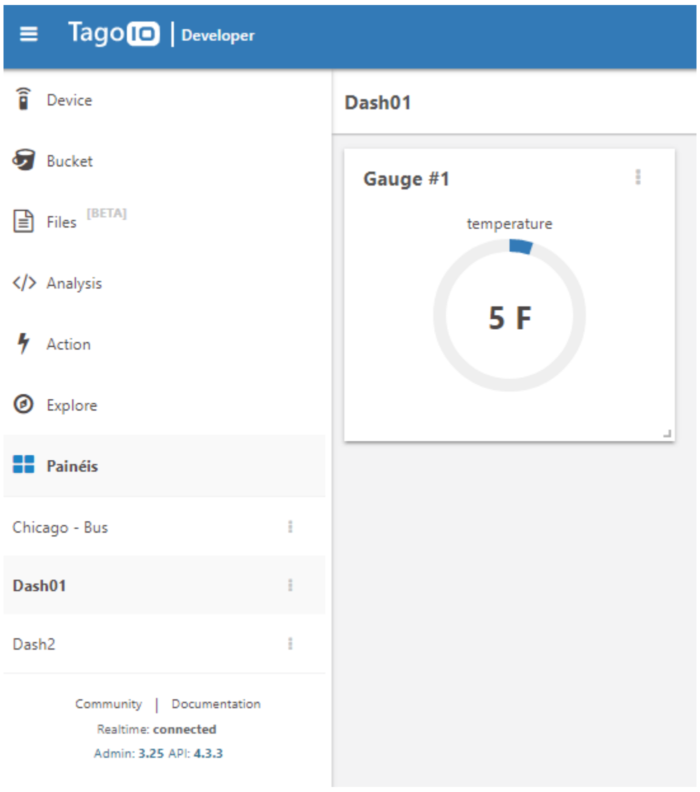


Figure 18 - Websocket message values shown in Tago

To learn more about this protocol, go to: https://websockets.readthedocs.io/en/stable/index.html.