FIWARE
Global
Summit

From Data
to Value

OPEN SOURCE
OPEN STANDARDS
OPEN COMMUNITY

# Code Walkthrough: Strategies for creating custom IoT Agents

Jason Fox, Technical Evangelist, FIWARE Foundation

Vienna, Austria
12-13 June, 2023
#FIWARESummit

FIWARE
Open APIs for Open Minds

# Existing IoT Agents

- Open Source IoT Agents available in the **FIWARE Catalogue** https://github.com/FIWARE/catalogue

IoT Agent for **JSON**
- **JSON** payload over **HTTP/MQTT/AMPQ** transport

IoT Agent for **Ultralight**
- **Character-based** payload over **HTTP/MQTT/AMPQ** transport

- IoT Agent for **LWM2M**
  - **JSON**, **Plain Text** etc. payloads over **UDP** transport

IoT Agent for **LoRaWAN**
- **CBOR/CayenneLPP** payload over **TTN/ChirpStack** transport

IoT Agent for **OPC-UA**
- **OPC-UA** objects/variables/methods over **OPC-UA** transport

IoT Agent for **Sigfox**
- **Binary** payload over **HTTP** transport

- Additional Custom and Proprietary IoT Agents are available in the **FIWARE Marketplace** http://marketplace.fiware.org/



FIWARE

# Consider Transport, Payload and Protocol/Handshake

- I want an IoT Agent listening over HTTP
- I want an IoT Agent that only accepts messages of the correct **`Content-Type`**
- I want an IoT Agent that rejects payloads missing an API Key
- I want an IoT Agent that processes both measures and commands
- Commands must respond with a success/fail value

# Custom XML Payload in Detail

- The IoT Agent library can deal with attribute mappings, NGSI-v2/NGSI-LD payloads and communication. Just add common middleware functions using i**otAgent.addUpdateMiddleware()**
- **Northbound** will need a custom parser to translate the payload into an in-memory data object and eventually attribute/value data objects for the **iotagent.update()** function
- **Southbound** will need handler code to process NGSI. This is set in **iotagent.setCommandHandler()** and translates to the proprietary payload
- Use standard parsing libraries where available or write your own if necessary.

**Content-Type:** application/xml

**Northbound** Measures are:

```
<measure device="lamp002" key="xxx">
 <c value="1"/>
</measure>
```

**Southbound** Commands are:

```
<turn device="Robot1">
 <left>30</left>
</turn>
```

**Northbound** Command response is:

```
<success/>
```

or

```
<failure/>
```



FIWARE

# IoT Agent for ADAPT/ISOXML

- **ISO 11783** is an electronics communications protocol for agricultural equipment.
- Well defined ISO standard. Full specification runs for over 100 pages.
- XML based messaging used as an interchange between devices from multiple manufacturers
- Multiple messages within a single payload. Data typically uploaded to an MICS using a USB Stick
- Context data is usually a `<TSK>` within a `<ISO11783_TaskData>`

**`Content-Type:`** application/xml

```
<?xml version="1.0" encoding="utf-8" ?>
<ISO11783_TaskData>
    <FRM A="FRM3" B="Manor Farm" C="Street2"
         D="PO Box2" E="PostalCode2" F="City2"
         G="State2" H="Country2" I="CTR2"/>
    <CTR A="CTR2" B="Mr. Jones" />
    <TSK A="TSK11" B="mass-based product allocation">
        <PAN A="PDT2" B="004B" C="20000" D="1"
             E="DET3" F="VPN1">
            <ASP A="2019-11-12T08:00:00" D="4"/>
        </PAN>
    </TSK>
    <PGP A="PGP1" B="Herbicides"/>
    <PDT A="PDT1" B="agent 1" C="PGP1"/>
    <PGP A="PGP2" B="Potato" C="2" />
    <PDT A="PDT2" B="Asterix" C="PGP2"/>
    <VPN A="VPN1" B="0.001" C="1.0" D="0" E="kg"/>
</ISO11783_TaskData>
```



FIWARE

# Custom IoT Agent for CSV and Excel

- **Measures** only
- CSV and Excel files parsers exist
- Data is always in a proprietary format. The customer needs to define what the columns actually mean

Process data step-by-step:

1. Sanity checks on data input
2. Upload raw datafile
3. Read raw data rows from file
4. Manipulate in-memory data objects
5. Convert to NGSI data
6. Send NGSI Upsert requests
7. Process responses from Context Broker

Store additional metadata in a database and retrieve and reuse if necessary

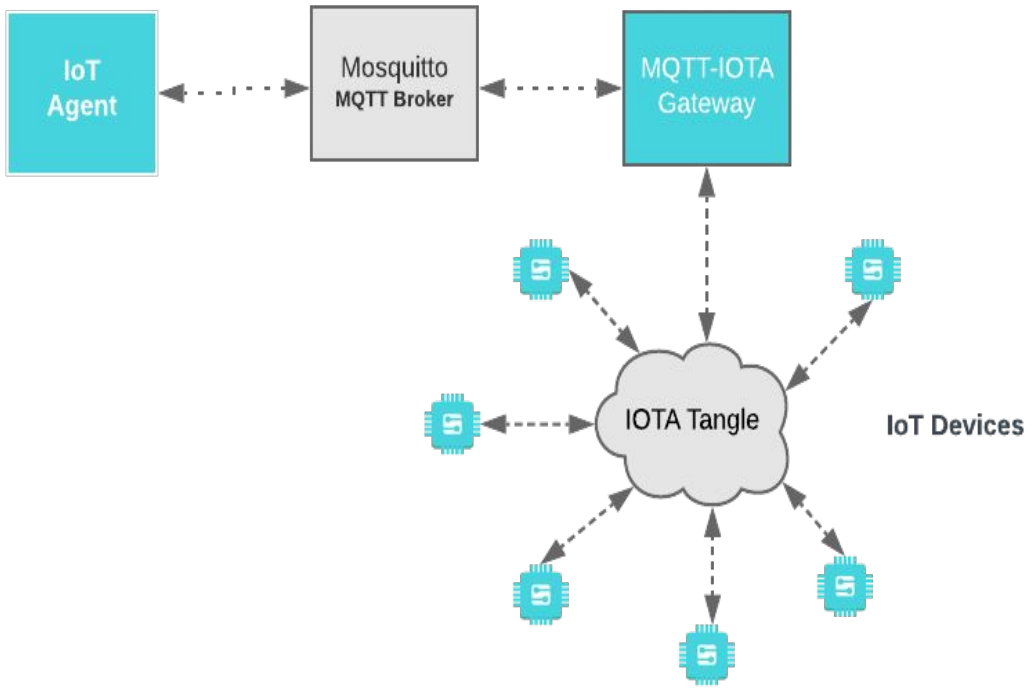
**Content-Type:** text/csv

```
"timestamp","0ANAB7|Value","0ANAB7|Quality",
  "0BNAB9|Value", "0BNAB9|Quality"
"2020-04-28 14:00:00+03:00","2.9985","1","-.75","524288"
"2020-04-28 16:00:00+03:00","2.9445","1","-.75","524288"
"2020-04-28 18:00:00+03:00","2.8900","1","-.75","524288"
"2020-04-28 20:00:00+03:00","2.7450","1","-.75","524288"
"2020-04-28 22:00:00+03:00","2.6331","1","-.75","524288"
```

**Content-Type:**
application/vnd.openxmlformats-officedocument.spreadsheetml.sheet

```
504b 0304 1400 0600 0800 0000 2100 aaf7 58a4 7901 0000
1406 0000 1300 0802 5b43 6f6e 7465 6e74 5f54 7970 6573
5d2e 786d 6c20 a204 0228 a000 0200 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000
```

# Architectural Overview of a FIWARE-IOTA Gateway




- IoT Devices interact with **the Tangle** directly.
- Expand and reuse an IoT Agent's existing **MQTT** Binding - the IoT Agent communicates with IoT devices indirectly via an MQTT Broker and a custom Gateway.
- For Southbound Traffic:
  - the Gateway intercepts **MQTT** topics and sends them to **the Tangle**.
- For Northbound Traffic:
  - **the Tangle** informs the Gateway of incoming data and the Gateway forwards this to **MQTT**.

More info: https://github.com/FIWARE/tutorials.IoT-over-IOTA

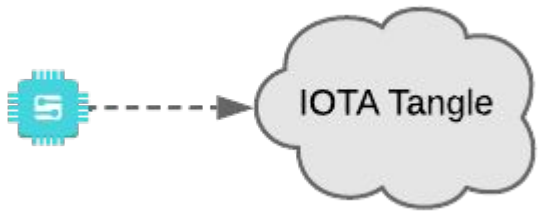
FIWARE

# Connecting Devices directly to the Tangle

FIWARE

# **Northbound** Measures from Device to the Tangle

A simple transaction pushing data onto **the Tangle**




- Connect to an IOTA node.
- Pushes data to the tangle using a common message index (e.g. `fiware/attrs`) for the Gateway to recognise upstream.

see https://wiki.iota.org/iota.rs/libraries/nodejs/examples#messages

FIWARE

# **Code - Northbound** Measures from Device to the Tangle

## Ultralight payload

```
const async = require('async');
const iota = require('@iota/client');
const IOTA_ATTRS_TOPIC = 'fiware/attrs';
const IOTA_CLIENT = new iota.ClientBuilder().node
   ('https://chrysalis-nodes.iota.cafe').build();

const queue = async.queue((payload, callback) => {
   IOTA_CLIENT.message()
      .index(IOTA_ATTRS_TOPIC)
      .data(payload)
      .submit()
      .then((message) => {
         callback();
      })
      .catch((err) => {
         setTimeout(() => {
            // resending failed measure
            queue.push(payload);
         }, 1000);
         callback(err);
      }, 8);
});
```

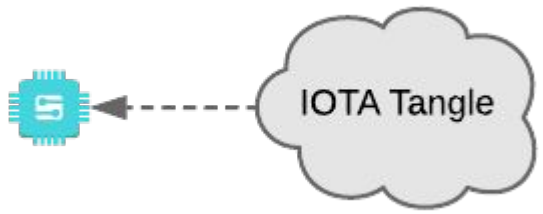
```
function sendToTangle(deviceId, apiKey, state) {
   // Sending an Ultralight measure to the
   // Tangle with a timestamp
   const payload =
      'i=' + deviceId +
      '&k=' + apiKey +
      '&d=' + state +
      '|t|' + new Date().toISOString();
   queue.push(payload);
}
```

- Connect to an IOTA node.
- Define a common message index **`fiware/attrs`** for the Gateway to be able to receive the measure upstream.
- Use a **defined payload format**, and remember settlement takes time, include an **`observedAt`**.
- Include retry for failure.

FIWARE

# **Southbound** Commands from the Tangle to Device

Listen for data settled on **the Tangle** read it and decode the command




- Device connects to an IOTA node.
- Listens to a common message index (e.g. `fiware/cmd`) to be informed that an incoming command has settled on **the Tangle**.
- Read the full command payload direct from **the Tangle** and then process the payload as necessary.

see https://wiki.iota.org/iota.rs/libraries/nodejs/examples#listening-to-mqtt

FIWARE

# **Code - Southbound** Commands from the Tangle to Device

```
const iota = require('@iota/client');
const IOTA_COMMAND_TOPIC = 'fiware/cmd';
const IOTA_CLIENT = new iota.ClientBuilder().node
   ('https://chrysalis-nodes.iota.cafe').build();

IOTA_CLIENT.getInfo()
   .then(() => {
      IOTA_CLIENT.subscriber()
         .topic(IOTA_COMMAND_TOPIC)
         .subscribe((err, data) => {
            if (err) {
               // Error Handling goes here
               return;
            }
            return process.nextTick(() => {
               readFromTangle(data);
            });
         });
   })
   .catch((err) => {
      // More Error Handling goes here
   });
```

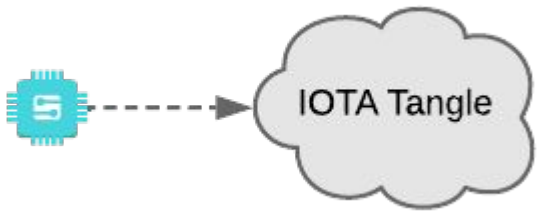
```
function readFromTangle(data) {
   const messageId = IOTA_CLIENT.getMessageId(data.payload);
   IOTA_CLIENT.getMessage()
      .data(messageId)
      .then((messageData) => {
         const commandPayload =
            Buffer.from(messageData.message.payload.data, 'hex')
               .toString('utf8');
         processCommand(commandPayload);
      })
      .catch((err) => {
         // Error Handling goes here
      });
}
```

- Connect to an IOTA node and listen to the message index **`fiware/cmd`** to be informed that an incoming command has settled on **the Tangle**.
- Read the full command payload direct from **the Tangle** interpret it and then process the payload as necessary.

FIWARE

# **Northbound** Command Acknowledgement

Pushing command responses back onto **the Tangle**




- Device connects to an IOTA node.
- Responds back to **the Tangle** on a common message index (e.g. `fiware/cmdExe`) for the Gateway to recognise upstream, queuing the acknowledgement just like a measure.

see https://wiki.iota.org/iota.rs/libraries/nodejs/examples#messages

FIWARE

# **Code -** Command Extraction, Execution and **Northbound** Acknowledgement

## Ultralight payload

```
const myApiKey = '12345';
const myDeviceId = 'x34995bc';
const IOTA_COMMAND_EXE_TOPIC = 'fiware/cmdExe';

// Extract the device id, apiKey and command instructions
function unmarshall(payload) {
   const parts = payload.split('&');
   const obj = {};
   parts.forEach((elem) => {
      const keyValues = elem.split('=');
      obj[keyValues[0]] = keyValues[1];
   });
   return obj;
}

function processCommand(commandPayload) {
   const data = unmarshall(commandPayload);
   // process the command if valid for this device
   if (data.k === myApiKey && data.i === myDeviceId){
      actuateDevice(data.d)
        .then((message) => {
           acktoTangle(myApiKey, deviceId, message);
        })
        .catch((err) => {
           ackToTangle(myApiKey, deviceId, err);
        });
   }
}
```

```
function ackToTangle (deviceId, apiKey, result) {
   // Sending an Ultralight command acknowledgement
   const ackPayload =
      'i=' + deviceId +
      '&k=' + apiKey +
      '&d=' + result;
   process.nextTick(() => {
       queue.push(ackPayload);
   });
}
```

- Command Payloads placed on **the Tangle** should be in a **well defined format** and must include **`deviceId`** and **`apiKey`** along with instructions for actuation.
- Interpret the payload and actuate the device if the keys match.
- Respond back to **the Tangle** using the message index **`fiware/cmdExe`** for the Gateway to receive the acknowledgment upstream, queuing it just like a measure.

FIWARE

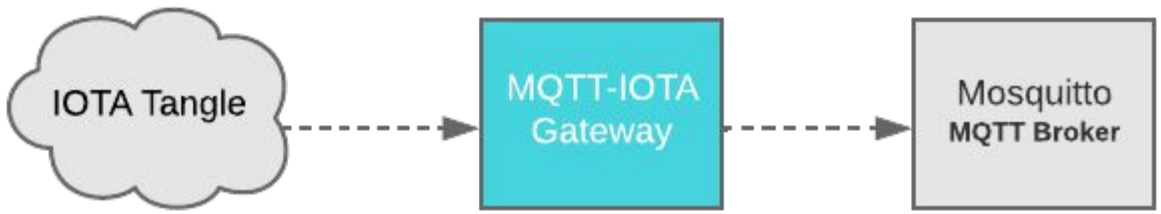
# Creating an IoT Agent Gateway

## Repurposing an IoT Agent's existing **MQTT** Binding

FIWARE

# **Northbound** Measures from the Tangle to the MQTT Broker

Listen for data settled on **the Tangle**, read it, decode it and republish it on an MQTT topic




- Gateway connect to an IOTA node
- Listens to a common message index (e.g. `fiware/attrs`) to be informed that an incoming measure has settled on **the Tangle**.
- The data held on **the Tangle** contains the `apiKey`, the `deviceId` and the payload of the acknowledgement
- Reads the payload direct from **the Tangle** and then publishes it to an MQTT topic `ul/<apiKey>/<deviceId>/attrs` - this also includes the relevant payload format. (e.g. `ul`, `json` etc.)

FIWARE

# **Code - Northbound** Measures from the Tangle to the MQTT Broker

## Ultralight payload

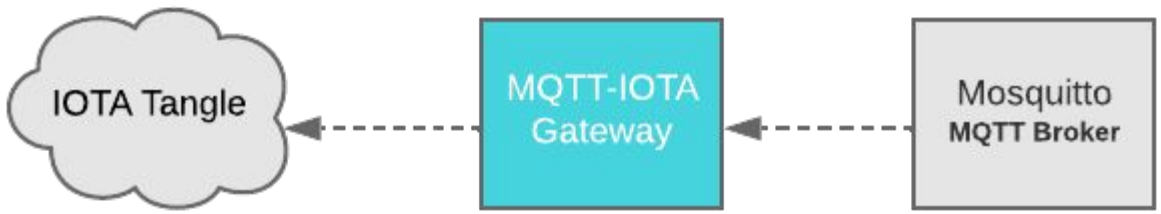
```
const iotaClient = require('@iota/client');
const IOTA_CLIENT = new iotaClient.ClientBuilder().node
    ('https://chrysalis-nodes.iota.cafe').build();
const IOTA_ATTRS_TOPIC = 'fiware/attrs';

IOTA_CLIENT.getInfo()
  .then(() => {
    IOTA_CLIENT.subscriber()
      .topic(IOTA_ATTRS_TOPIC)
      .subscribe((err, data) => {
         if (err) {
           // Error Handling goes here
         } else if (data) {
            const messageId =
                IOTA_CLIENT.getMessageId(data.payload);
            IOTA_CLIENT.getMessage()
               .data(messageId)
               .then((messageData) => {
                  measureReceived(messageData)
               })
               .catch((err) => {
                  // Error Handling goes here
               });
            }
         });
    })
    .catch((err) => {
        // More Error Handling goes here
    });
```

```
const mqtt = require('mqtt');
const MQTT_CLIENT = mqtt.connect('mqtt://mosquitto');
const MQTT_TOPIC_PROTOCOL = 'ul';

function measureReceived(messageData) {
   const payload =
      Buffer.from(messageData.message.payload.data, 'hex')
         .toString('utf8');
   const data = unmarshall(payload);
   process.nextTick(() => {
      forwardAsMQTT(data.k, data.i, data.d, 'attrs')});
   }
}

function forwardAsMQTT(apiKey, deviceId, state, topic) {
    const mqttTopic = '/' + MQTT_TOPIC_PROTOCOL +
      '/' + apiKey +
      '/' + deviceId +
      '/' + topic;

    MQTT_CLIENT.publish(mqttTopic, state);
}
```

FIWARE

# **Southbound** Commands from MQTT Broker to the Tangle

Subscribe to an MQTT topic,and push the transaction to **the Tangle**




- Gateway connects to an IOTA node
- Gateway subscribes to all MQTT topics of the form `/++/++/cmd` which maps to `/<apiKey>/<deviceId>/cmd`
- When notified of a command via MQTT, the Gateway pushes data to **the tangle** using a common message index (e.g. `fiware/cmd`) for the Device to recognise it downstream

FIWARE

# **Code - Southbound** Commands from MQTT Broker to the Tangle

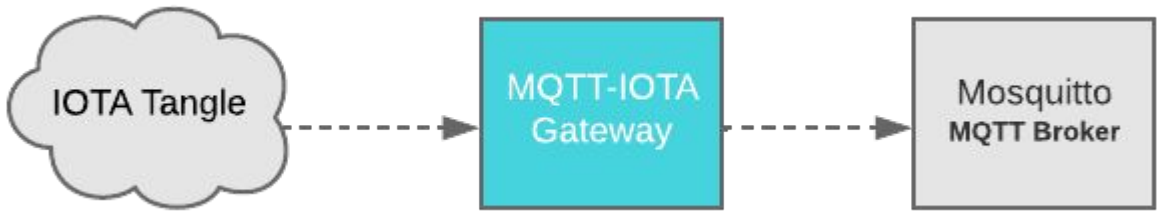
```
const mqtt = require('mqtt');
const MQTT_CLIENT =
mqtt.connect('mqtt://mosquitto');
MQTT_CLIENT.on('connect', () => {
   MQTT_CLIENT.subscribe('/+/+/cmd');
});
MQTT_CLIENT.on('message', command);


function command(topic, message) {
   const parts = topic.toString().split('/');
   const apiKey = parts[1];
   const deviceId = parts[2];
   const action = parts[3];
   process.nextTick(() => {
      forwardAsIOTATangle(apiKey, deviceId,
          message.toString());
   });
}
```

```
const iotaClient = require('@iota/client');
const IOTA_CLIENT = new iotaClient.ClientBuilder().node
    ('https://chrysalis-nodes.iota.cafe').build();
const IOTA_CMD_TOPIC = 'fiware/cmd';

function forwardAsIOTATangle(apiKey, deviceId, state) {
   const payload = 'i=' +
      deviceId + '&k=' +
      apiKey + '&d=' +
      state;

   IOTA_CLIENT.message()
        .index(IOTA_CMD_TOPIC)
        .data(payload)
        .submit()
        .then((message) => {
           // Push to tangle was successful
        })
        .catch((err) => {
           // Error Handling goes here
           forwardAsMQTT(apiKey, deviceId, err, 'cmdexe');
        });
}
```

FIWARE

# **Northbound** Acknowledgement from the Tangle to the MQTT Broker

Listen for data settled on **the Tangle**, read it, decode it and republish it on an MQTT topic




- Gateway connects to an IOTA node
- Listens to a common message index (e.g. `fiware/cmdExe`) to be informed that a command acknowledgement has settled on **the Tangle**.
- The data held on **the Tangle** contains the `apiKey`, the `deviceId` and the payload of the acknowledgement
- Read the payload direct from **the Tangle** and then publish it to an MQTT topic `ul/<apiKey>/<deviceId>/cmdExe`

FIWARE

# **Code - Northbound** Acknowledgement from the Tangle to the MQTT Broker

```
const iotaClient = require('@iota/client');
const IOTA_CLIENT = new iotaClient.ClientBuilder().node
    ('https://chrysalis-nodes.iota.cafe').build();
const IOTA_CMD_EXE_TOPIC = 'fiware/cmdExe';

IOTA_CLIENT.getInfo()
  .then(() => {
    IOTA_CLIENT.subscriber()
      .topic(IOTA_CMD_EXE_TOPIC)
      .subscribe((err, data) => {
         if (err) {
           // Error Handling goes here
         } else if (data) {
            const messageId =
                IOTA_CLIENT.getMessageId(data.payload);
            IOTA_CLIENT.getMessage()
               .data(messageId)
               .then((messageData) => {
                  ackReceived(messageData)
               })
               .catch((err) => {
                  // Error Handling goes here
               });
            }
         });
    })
    .catch((err) => {
        // More Error Handling goes here
    });
```

```
const mqtt = require('mqtt');
const MQTT_CLIENT = mqtt.connect('mqtt://mosquitto');
const MQTT_TOPIC_PROTOCOL = 'ul';

function ackReceived(messageData) {
   const payload =
      Buffer.from(messageData.message.payload.data, 'hex')
         .toString('utf8');
   const data = unmarshall(payload);
   process.nextTick(() => {
      forwardAsMQTT(data.k, data.i, data.d, 'cmdExe')});
   }
}

function forwardAsMQTT(apiKey, deviceId, state, topic) {
    const mqttTopic = '/' + MQTT_TOPIC_PROTOCOL +
      '/' + apiKey +
      '/' + deviceId +
      '/' + topic;

    MQTT_CLIENT.publish(mqttTopic, state);
}
```

FIWARE

FIWARE Global Summit

Find Us On

Stay up to date

JOIN OUR NEWSLETTER

Be certified and featured

FIWAREMarketplace

## Hosting Partner

City of Vienna

vienna business agency

## Keystone Sponsors

aws

ms.gis

Raseel

Red Hat

## Media Partners

bee smart city

CITIES FORUM

CitiesToday

Our Future Water

RM RENEWABLE MATTER

Smart Cities World

the smartcity journal

FIWARE
Global
Summit

# Thanks!

Vienna, Austria
12-13 June, 2023
#FIWARESummit

FIWARE
Open APIs for Open Minds