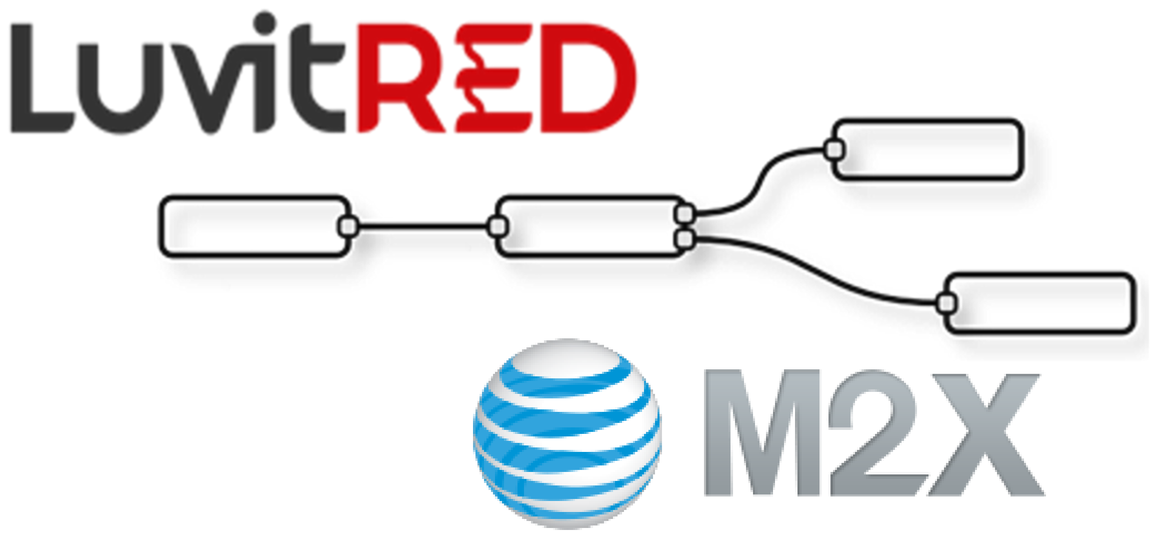


Connecting to M2X using LuvitRED

Franco Arboleda

5-May-16

OPTION

Table of Contents

1 Introduction

This document explains how to send and receive data to and from AT&T's M2X platform using LuvitRED.

This document was written using CloudGate's firmware version 2.59.0 and LuvitRED version 2.7.5. Although older versions of firmware and LuvitRED might work in the same way, we strongly recommend to upgrade to the above mentioned versions or newer in order to ensure the same results.

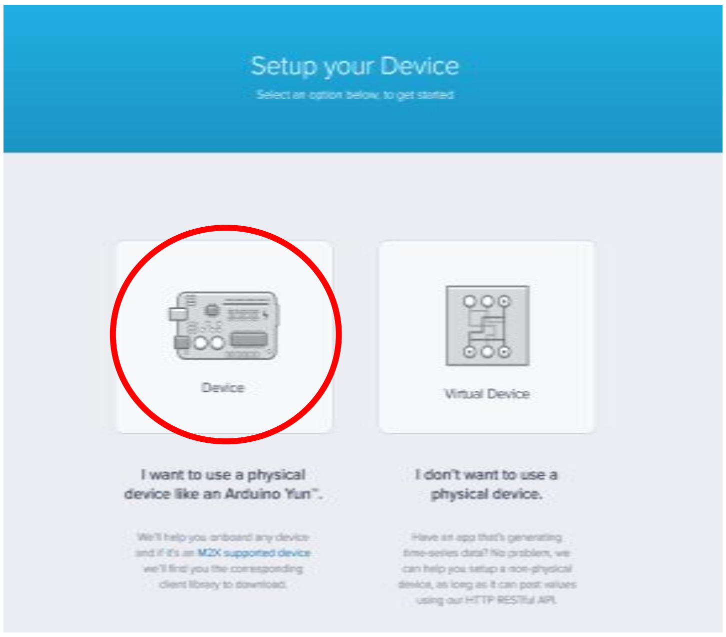
For this explanation we are using a free account on AT&T's M2X platform (https://m2x.att.com/).

2 Configuring the M2X platform.

2.1 Creating a new device on a new account

When creating a new account on the M2X platform, the user is prompted to create a new device. This section explains how to setup a new device in this case.

1. On the left hand side, click on "Device":



Setup your Device

Select an option below, to get started

I want to use a physical device like an Arduino Yun™.

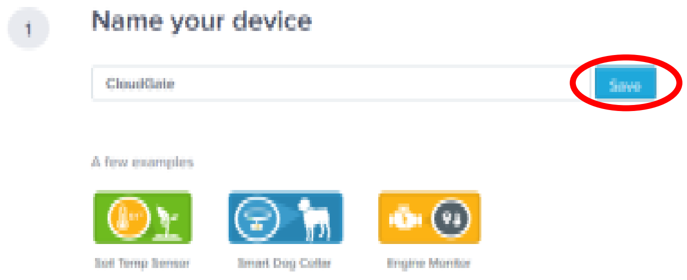
We'll help you onboard any device and if it's an M2X supported device we'll find you the corresponding client library to download.

I don't want to use a physical device.

Have an app that's generating time-series data? No problem, we can help you setup a non-physical device, as long as it can post values using our HTTP RESTful API.

Figure 1: M2X: Device.

2. Name your device (e.g. CloudGate) and click on "Save":



1 Name your device

CloudGate

Figure 2: M2X: Name your device.

3. Do not set any data stream yet, instead, go to the bottom of the page and click on "Go to Device":

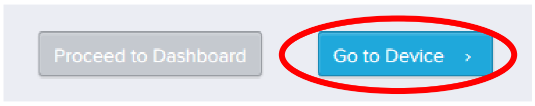
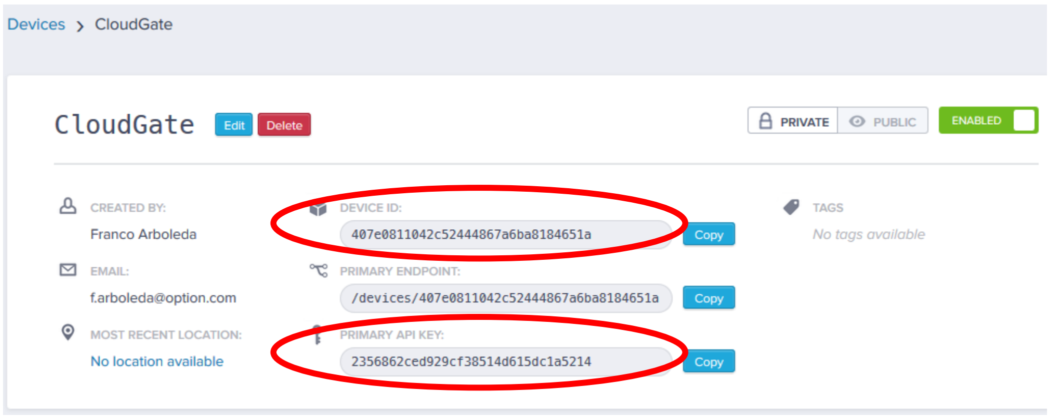


Figure 3: M2X: Go to Device.

4. Once on the newly created device page, take note of the DEVICE ID and PRIMARY API KEY of the device:



CREATED BY: Franco Arboleda

EMAIL: f.arboleda@option.com

MOST RECENT LOCATION: No location available

DEVICE ID: 407e0811042c52444867a6ba8184651a

PRIMARY ENDPOINT: /devices/407e0811042c52444867a6ba8184651a

PRIMARY API KEY: 2356862ced929cf38514d615dc1a5214

TAGS: No tags available

Figure 4: M2X: DEVICE ID and PRIMARY API KEY.

2.2 Creating a new device on an old account

1. Login to your account and go to "Devices". Then click on "Create New" and select "Device":

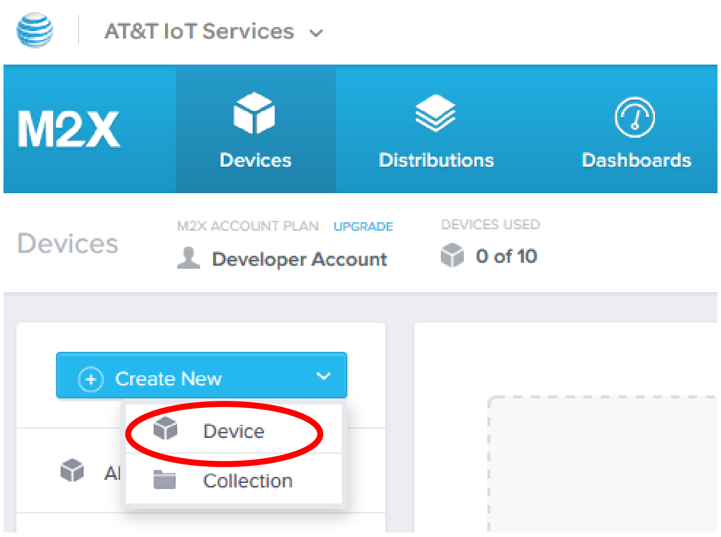
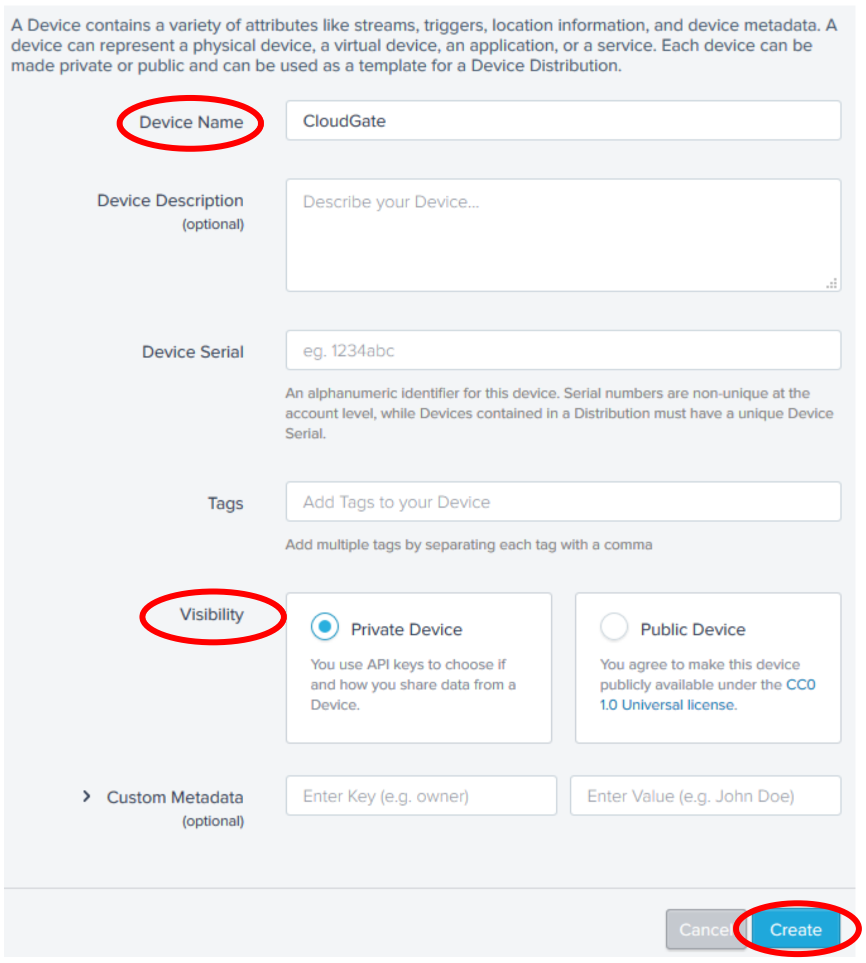


Figure 5: M2X: Create New - Device.

2. Provide a Device Name and make it a Private Device. Click on "Create" to finalize:



A Device contains a variety of attributes like streams, triggers, location information, and device metadata. A device can represent a physical device, a virtual device, an application, or a service. Each device can be made private or public and can be used as a template for a Device Distribution.

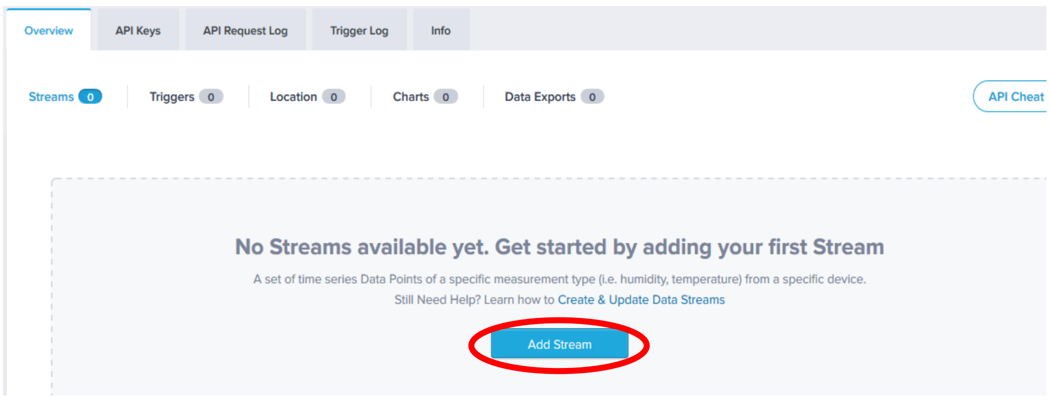
Device Name	CloudGate
Device Description (optional)	Describe your Device...
Device Serial	eg. 1234abc
	An alphanumeric identifier for this device. Serial numbers are non-unique at the account level, while Devices contained in a Distribution must have a unique Device Serial.
Tags	Add Tags to your Device
	Add multiple tags by separating each tag with a comma
Visibility	[✓ Private Device] You use API keys to choose if and how you share data from a Device. / Public Device You agree to make this device publicly available under the CC0 1.0 Universal license.
Custom Metadata (optional)	Enter Key (e.g. owner) / Enter Value (e.g. John Doe)

Figure 6: M2X: Device setup.

3. Take note of the DEVICE ID and PRIMARY API KEY as explained on section 2.1, step 4.

2.3 Adding a data stream

1. Add a new Stream under the Overview section by clicking on "Add Stream":



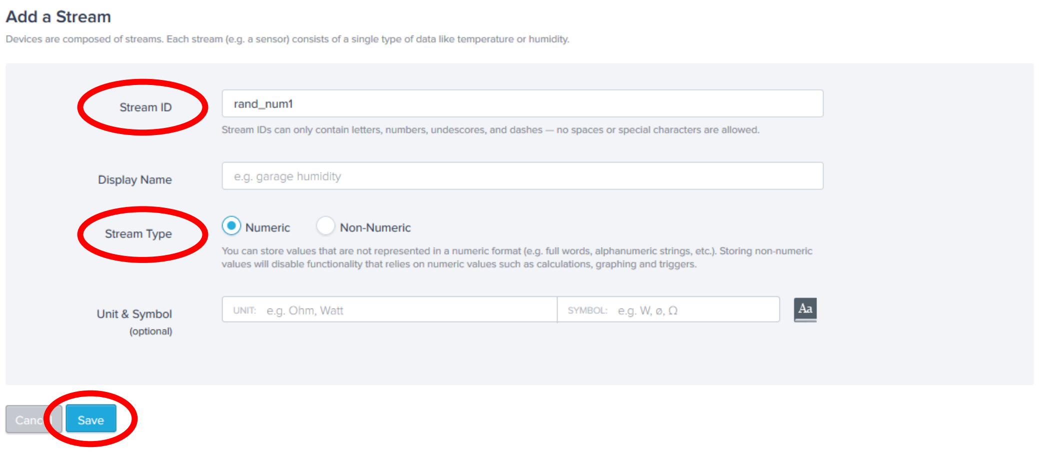
No Streams available yet. Get started by adding your first Stream

A set of time series Data Points of a specific measurement type (i.e. humidity, temperature) from a specific device.

Still Need Help? Learn how to Create & Update Data Streams

Figure 7: M2X: Add Stream.

2. Provide a Stream ID (rand_num1) and Stream Type (Numeric) to the new stream and click on "Save":



Add a Stream

Devices are composed of streams. Each stream (e.g. a sensor) consists of a single type of data like temperature or humidity.

Stream ID: rand_num1

Stream IDs can only contain letters, numbers, undescores, and dashes — no spaces or special characters are allowed.

Display Name: e.g. garage humidity

Stream Type: [✓ Numeric] Non-Numeric

You can store values that are not represented in a numeric format (e.g. full words, alphanumeric strings, etc.). Storing non-numeric values will disable functionality that relies on numeric values such as calculations, graphing and triggers.

Unit & Symbol (optional): UNIT: e.g. Ohm, Watt SYMBOL: e.g. W, ø, Ω

Figure 8: M2X: Add a Stream.

3. A new Stream called ***rand_num1*** is created under the Stream section:

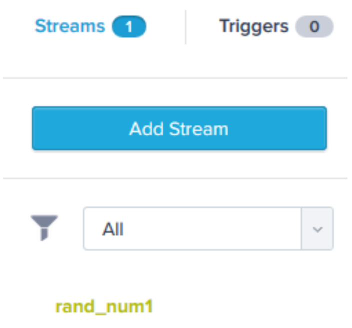


Figure 9: M2X: New Stream.

3 LuvitRED configuration.

Go to the web interface of the CloudGate and then to the "Plugin" tab and sub-tab called "LuvitRED"

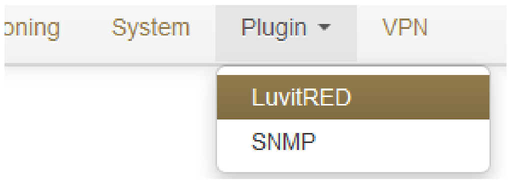


Figure 10: Plugin tab, LuvitRED.

<u>NOTE</u>: Do not focus on the SNMP (Simple Network Management Protocol) sub-tab. This tab is not going to be used on this document.

Under the "Advanced Editor" of LuvitRED, there are two nodes that are related to M2X (See Figure 11):

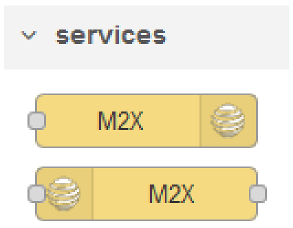


Figure 11: M2X nodes.

3.1 Sending data to the M2X platform.

1. Drop a M2X out node:

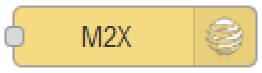
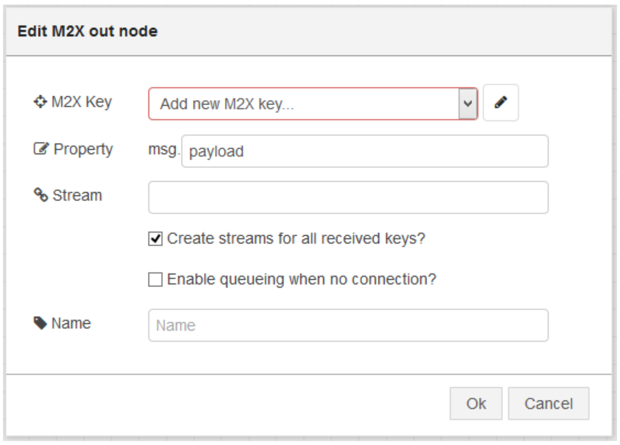


Figure 12: M2X out node.

2. Configure the M2X out node in the following way:

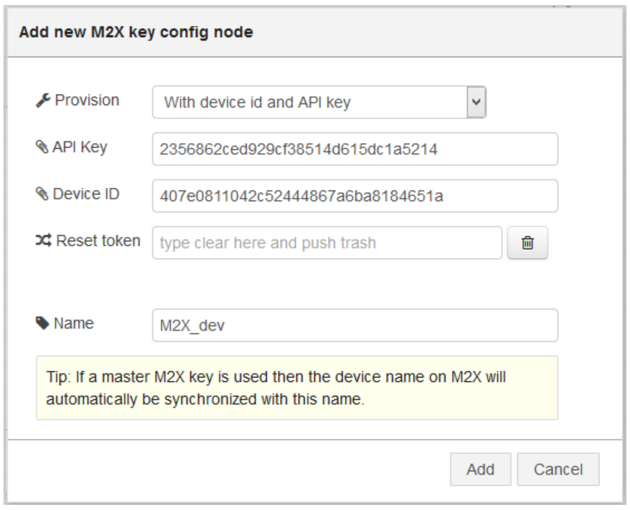


Edit M2X out node

M2X Key: Add new M2X key...
Property: msg. payload
Stream:
☑ Create streams for all received keys?
☐ Enable queueing when no connection?
Name: Name

Figure 13: Default M2X out node.

a. Add a **new** M2X Key by clicking on the pencil icon and configure it for the DEVICE ID (Device ID) and PRIMARY API KEY (API Key) noted on the previous section. Change the "Name" of the configuration node and click on "Add":



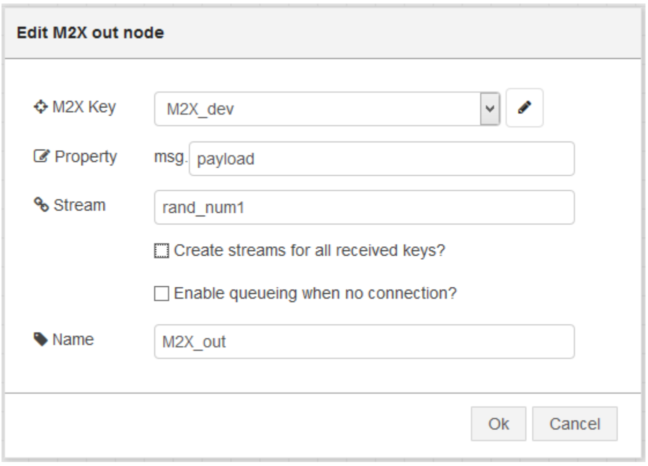
Add new M2X key config node

Provision: With device id and API key
API Key: 2356862ced929cf38514d615dc1a5214
Device ID: 407e0811042c52444867a6ba8184651a
Reset token: type clear here and push trash
Name: M2X_dev

Tip: If a master M2X key is used then the device name on M2X will automatically be synchronized with this name.

Figure 14: Adding Device's ID and API Key.

b. On the node configuration write the "Stream" name created on the M2X platform (***rand_num1***), uncheck the "Create streams for all received keys?" and change the name of the node. Click on "OK" when finished:

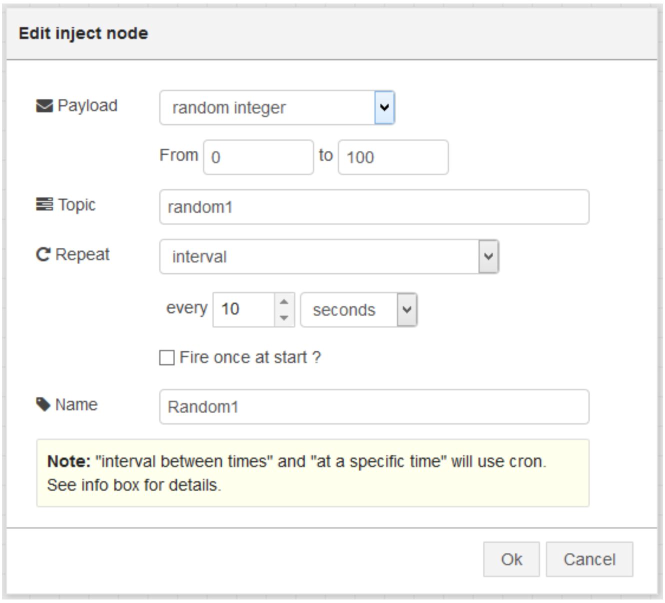

Edit M2X out node

M2X Key: M2X_dev
Property: msg. payload
Stream: rand_num1
☐ Create streams for all received keys?
☐ Enable queueing when no connection?
Name: M2X_out

Figure 15: Final node configuration.

This node is now configured to send one data stream that will be called ***rand_num1***. Now we need to actually send data.

1. Drop an inject node and configure it so that it outputs a random integer (0-100) every 10 seconds (give the node a name):


Edit inject node

Payload: random integer
From 0 to 100
Topic: random1
Repeat: interval
every 10 seconds
☐ Fire once at start ?
Name: Random1

Note: "interval between times" and "at a specific time" will use cron.
See info box for details.

Figure 16: Inject node configuration.

2. Let's connect the nodes together in the following way:

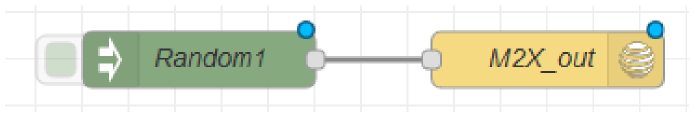


Figure 17: Final Configuration.

3. Click on "Deploy"

If we now go back to our device on the M2X platform, we should be able to see that the rand_num1 stream is being updated with the values that our CloudGate is sending:

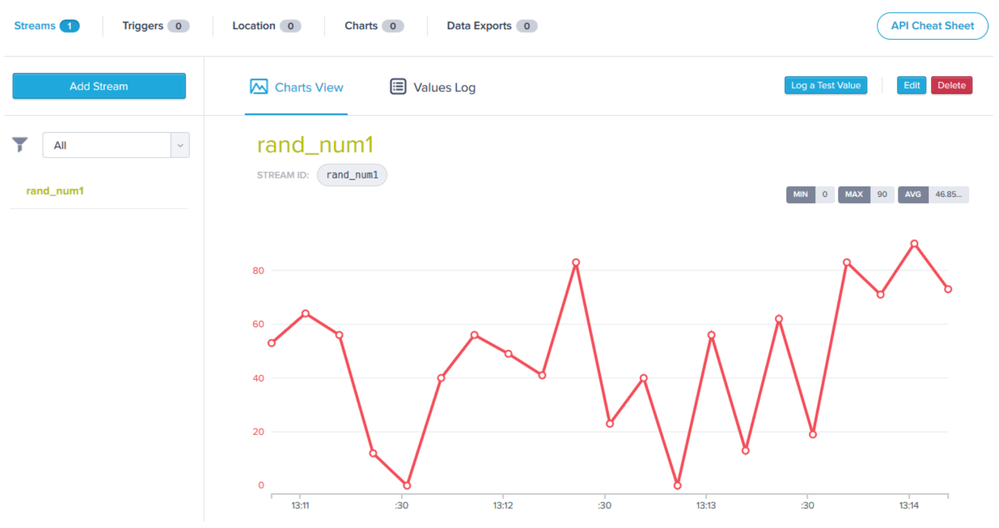
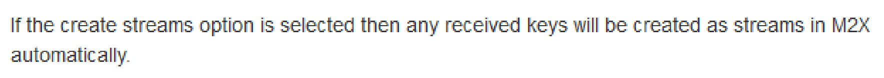


Figure 18: M2X: rand_num1 stream item updated.

Multiple streams can be added by replicating the above steps we used to add ***rand_num1***, but this is only a good solution for a reduced amount of streams. If we need to post data for a hundred streams, this solution is not the best. The M2X out node can send information for multiple streams as long as this information is provided to the node as a table in which case each key of the table need to be the name of the stream we want to update (See Figure 19):

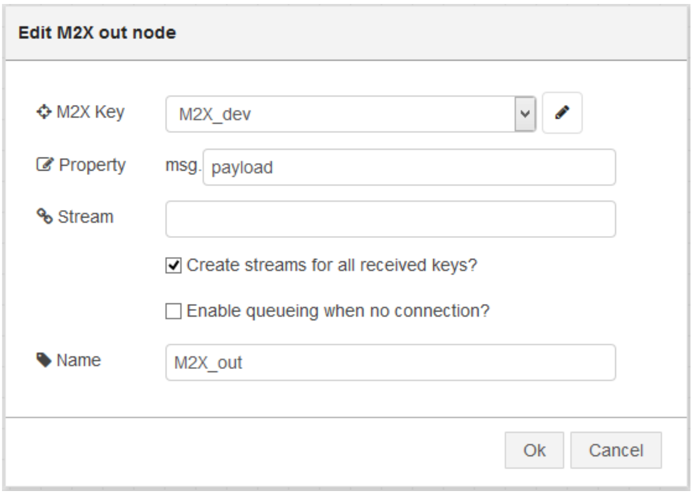


If the create streams option is selected then any received keys will be created as streams in M2X automatically.

Figure 19: Extract of the information tab of the M2X out node.

Let's adapt the configuration we already have so that we can post data for multiple streams using one single M2X out node:

1. Open the M2X out node and delete "Stream" name (Remember: the stream name is now going to be provided as the key of the table). Check the "Create streams for all received keys?" option (This allows us to create new streams on the M2X platform without the need to manually adding them) and click on OK:

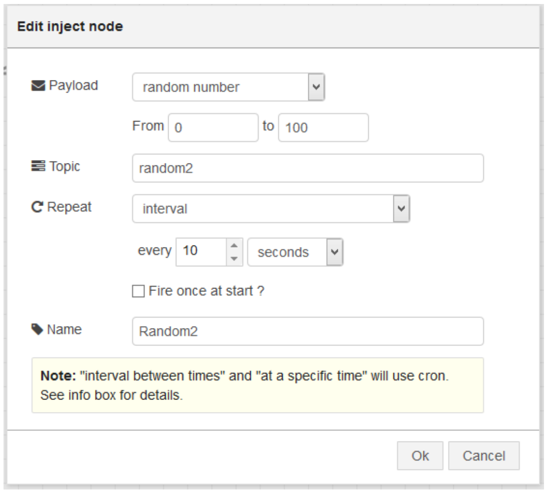


Edit M2X out node

M2X Key: M2X_dev
Property: msg. payload
Stream:
☑ Create streams for all received keys?
☐ Enable queueing when no connection?
Name: M2X_out

Figure 20: M2X out node with no Stream name.

2. Now, lets drag and drop a second inject node and configure it as follows:



Edit inject node

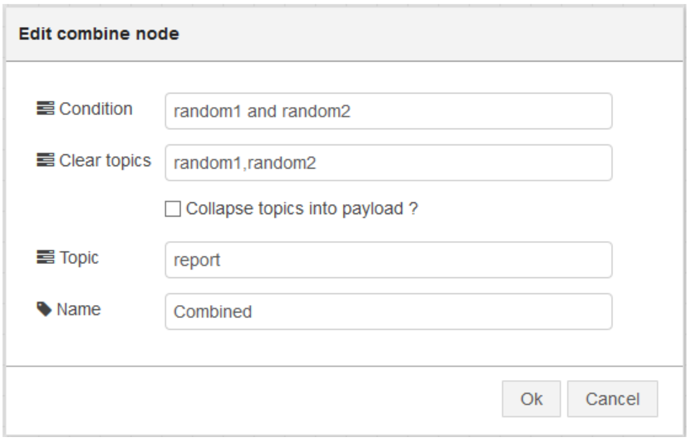
Payload: random number
From 0 to 100
Topic: random2
Repeat: interval
every 10 seconds
☐ Fire once at start ?
Name: Random2

Note: "interval between times" and "at a specific time" will use cron.
See info box for details.

Figure 21: Random2 inject node (random number).

<u>NOTE:</u> We can now use a combine node to combine the messages coming from both nodes into one message and using their topics as keys.

3. Drag and drop a combine node and configure it the following way:


Edit combine node

Condition: random1 and random2
Clear topics: random1,random2
☐ Collapse topics into payload ?
Topic: report
Name: Combined

Figure 22: Combine node.

NOTE: This configuration does the following:

- The condition is to wait for both random1 and random2 topics to arrive.
- After they arrive, clear the topics, so that the node waits for two new values to come.
- It changes the final topic to "report"

4. Drop a debug node (This is just to show the format of the data after the combine node).
5. Let's connect the nodes together in the following way and click on "Deploy":

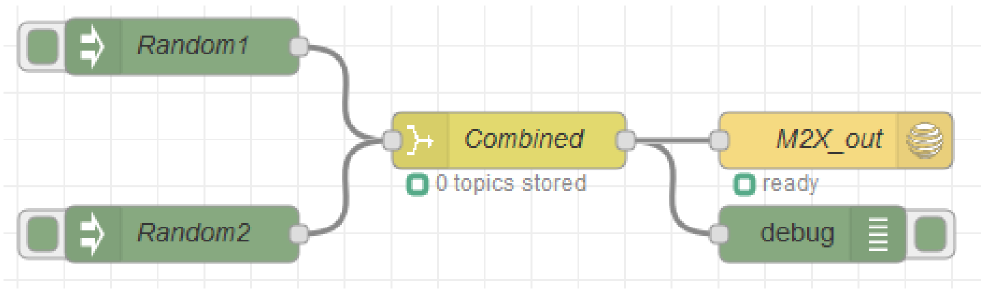
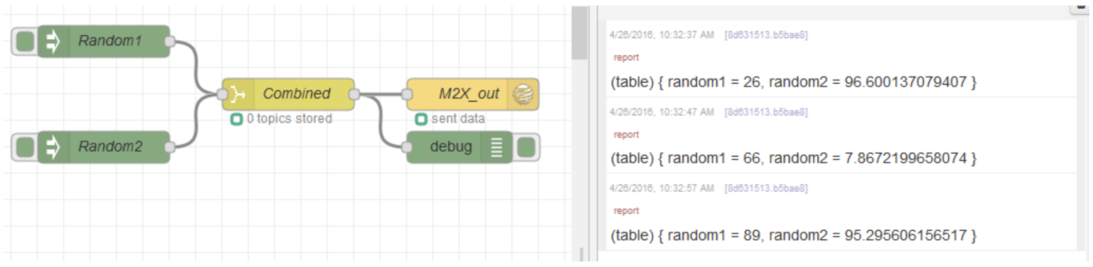


Figure 23: Configuration using combine and one M2X out node.

On the debug node, we should be able to see the table created by the combine node containing both random1 and random2 values:



```
4/26/2016, 10:32:37 AM   [8d631513.b5bae8]
report
(table) { random1 = 26, random2 = 96.600137079407 }
4/26/2016, 10:32:47 AM   [8d631513.b5bae8]
report
(table) { random1 = 66, random2 = 7.8672199658074 }
4/26/2016, 10:32:57 AM   [8d631513.b5bae8]
report
(table) { random1 = 89, random2 = 95.295606156517 }
```

Figure 24: Table of data shown under debug tab.

On the M2X platform we should see the values being updated (Notice that the two new streams are added into the platform):

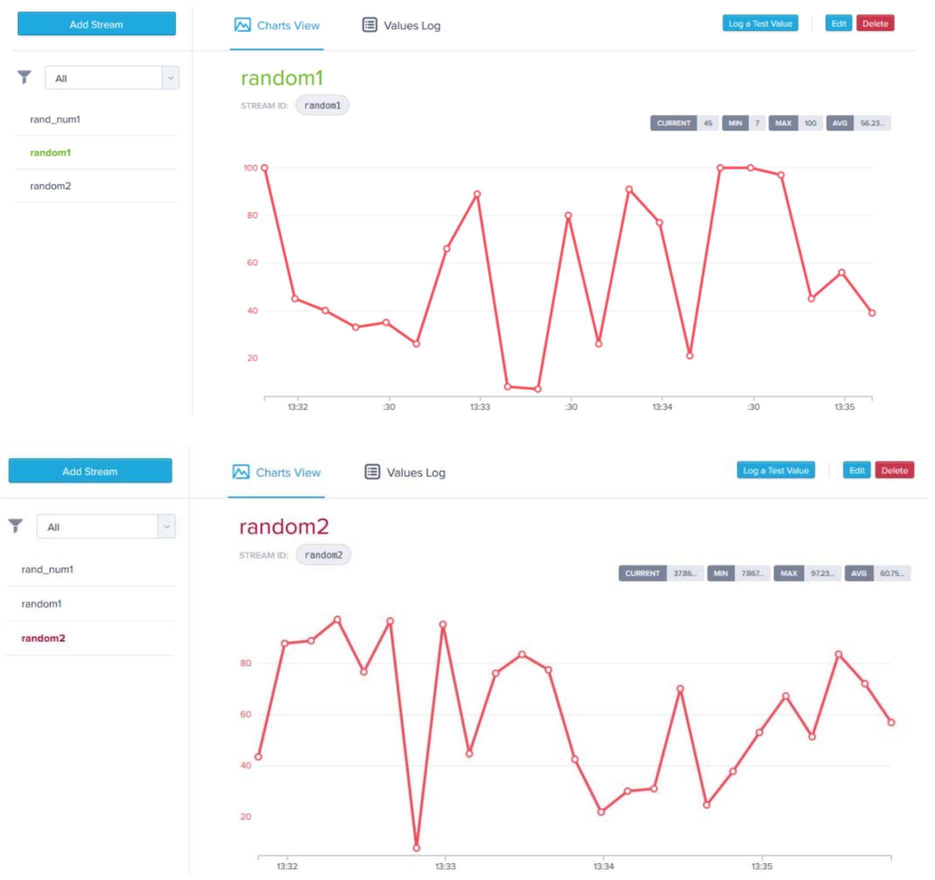


Figure 25: M2X: New Streams created and values updated.

<u>NOTE</u>: Since we are now creating the Streams directly from LuvitRED, we could delete the ***rand_num1*** stream from the platform without affecting the configuration.

3.2 Receiving commands from the M2X platform.

We have been successful in sending data from our device to the M2X platform, but how about receiving commands on our device? M2X provides a method of sending commands to the device. Each received commands needs to be marked accordingly to let the M2X platform know that it has been processed or rejected. For more information about the M2X commands, got to https://m2x.att.com/developer/documentation/v2/commands.

1. Drop an M2X in node:

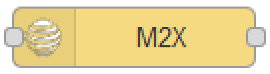
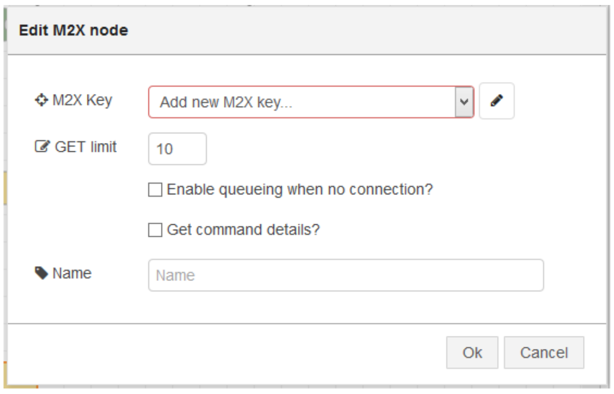


Figure 26: M2X in node.

2. Configure the M2X in node in the following way:



Edit M2X node

M2X Key: Add new M2X key...

GET limit: 10

☐ Enable queueing when no connection?

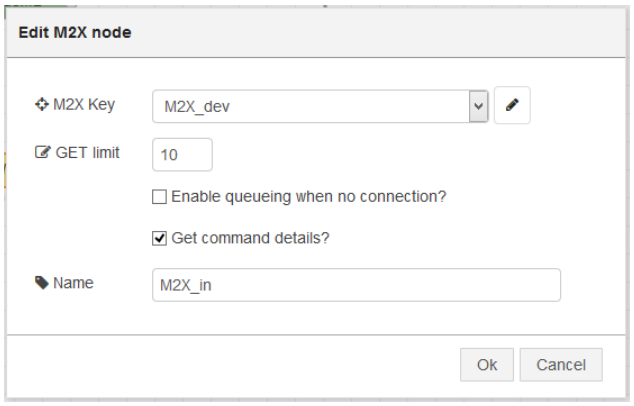
☐ Get command details?

Name: Name

Ok Cancel

Figure 27: Default M2X in node.

a. Select the already configured M2X Device. If no device is already configured, please follow the steps explained on section 3.1. Check the "Get command details?" option and change the name of the node:



Edit M2X node

M2X Key: M2X_dev

GET limit: 10

☐ Enable queueing when no connection?

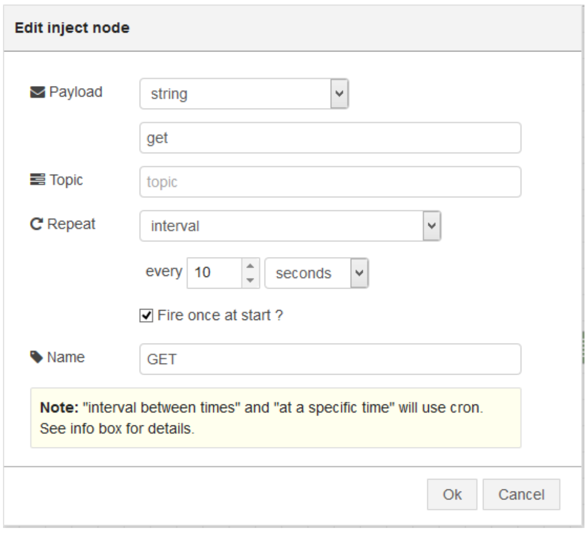
☑ Get command details?

Name: M2X_in

Ok Cancel

Figure 28: M2X in node configuration.

3. Drag and drop an inject node and configure it as follows:



Edit inject node

Payload: string

get

Topic: topic

Repeat: interval

every 10 seconds

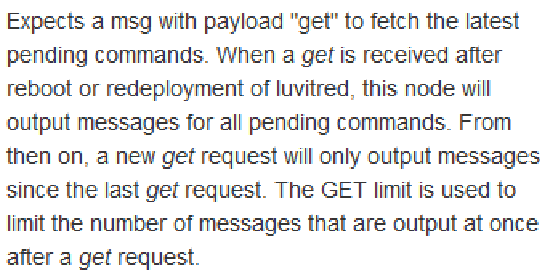
☑ Fire once at start ?

Name: GET

Note: "interval between times" and "at a specific time" will use cron. See info box for details.

Figure 29: Inject node configured to send "get" command every 10 seconds.

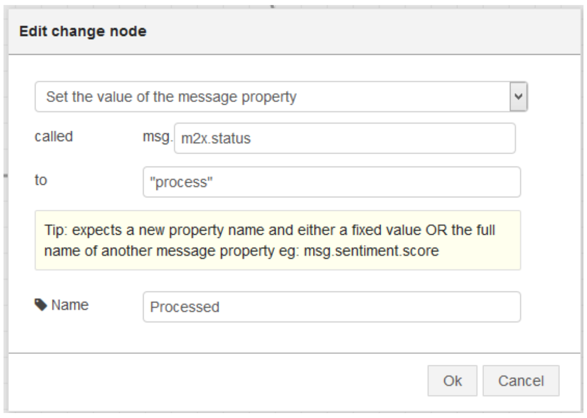
Keep in mind the information on the M2X in node:



Expects a msg with payload "get" to fetch the latest pending commands. When a *get* is received after reboot or redeployment of luvitred, this node will output messages for all pending commands. From then on, a new *get* request will only output messages since the last *get* request. The GET limit is used to limit the number of messages that are output at once after a *get* request.

Figure 30: Extract of the information on the M2X in node.

4. Drag and drop a change node and configure it to set the status of the received command to processed:



Edit change node

Set the value of the message property

called msg. m2x.status

to "process"

Tip: expects a new property name and either a fixed value OR the full name of another message property eg: msg.sentiment.score

Name: Processed

Figure 31: Change node configured to set the status to process.

Keep in mind the information on the M2X in node:

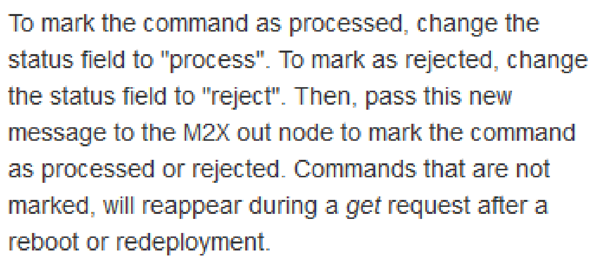
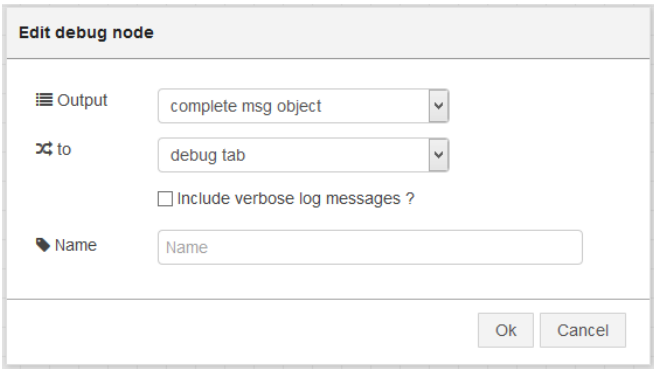

To mark the command as processed, change the status field to "process". To mark as rejected, change the status field to "reject". Then, pass this new message to the M2X out node to mark the command as processed or rejected. Commands that are not marked, will reappear during a *get* request after a reboot or redeployment.

Figure 32: Extract of the information on the M2X in node.

5. Drag and drop two debug nodes, but we need to reconfigure the Output from "payload only" to "complete msg object" in order to see the m2x command information:


Edit debug node

Output: complete msg object

to: debug tab

☐ Include verbose log messages ?

Name: Name

Figure 33: Debug node output configuration.

6. Let's connect the nodes together in the following way and click on "Deploy":

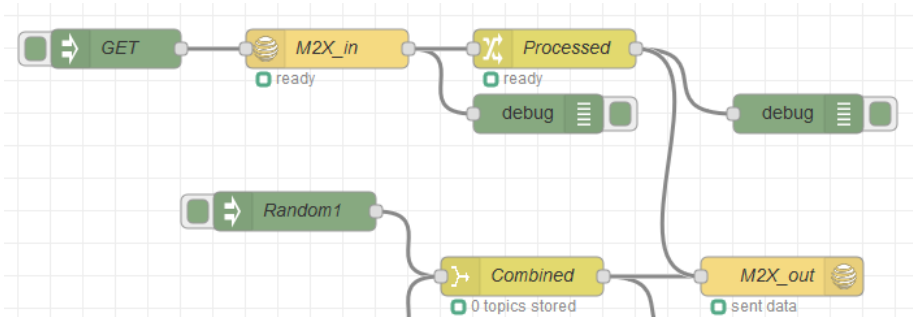


Figure 34: Final configuration.

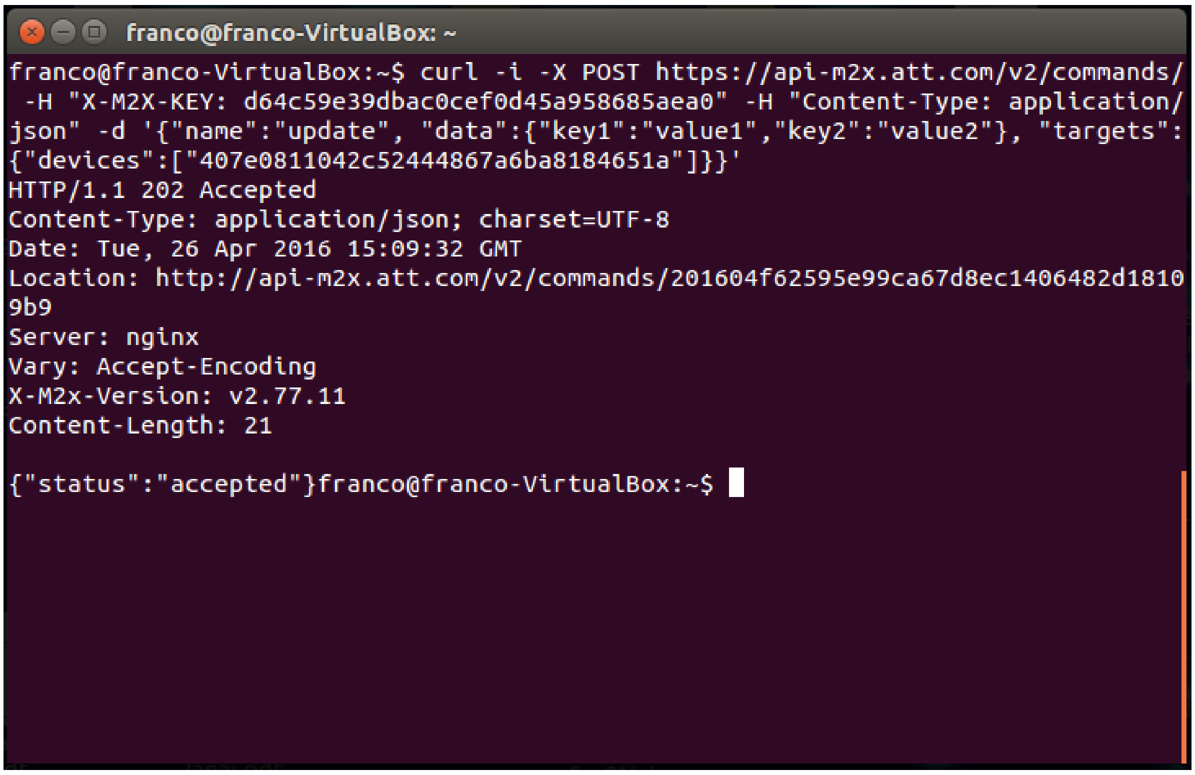
On this configuration we are reusing the M2X_out node from section 3.1 in order to send the "process" command back to the M2X platform.

Now, let's send a command to the CloudGate. For this, we can use a command line tool called "cURL" to send commands to a Device using the M2X platform. We are going to test using cURL on from a Linux machine. The following command will be send:

curl -i -X POST https://api-m2x.att.com/v2/commands/ -H X-M2X-KEY: d64c59e39dbac0cef0d45a958685aea0 -H Content-Type: application/json -d {"name":"update", "data":{"key1":"value1","key2":"value2"}, "targets":{"devices":["407e0811042c52444867a6ba8184651a"]}}

- X-M2X-KEY: Is the account MASTER KEY (found under the account settings).
- data: is the JSON formatted data sent to the devices
- devices: Is a list of the DEVICE IDs that should receive the command.

<u>NOTE</u>: For more information about the structure of the command and the information contained on it, please refer to the M2X documentation.

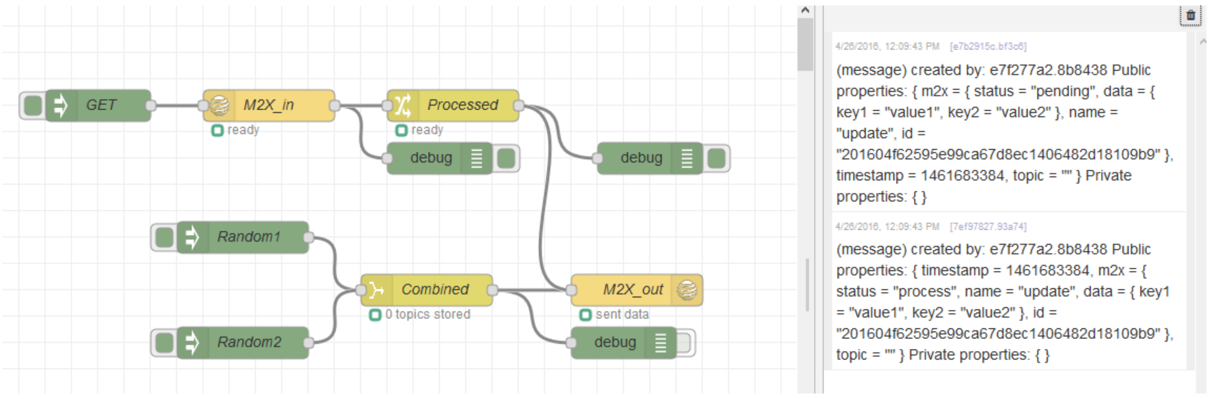


```
franco@franco-VirtualBox: ~
franco@franco-VirtualBox:~$ curl -i -X POST https://api-m2x.att.com/v2/commands/
 -H "X-M2X-KEY: d64c59e39dbac0cef0d45a958685aea0" -H "Content-Type: application/
json" -d '{"name":"update", "data":{"key1":"value1","key2":"value2"}, "targets":
{"devices":["407e0811042c52444867a6ba8184651a"]}}'
HTTP/1.1 202 Accepted
Content-Type: application/json; charset=UTF-8
Date: Tue, 26 Apr 2016 15:09:32 GMT
Location: http://api-m2x.att.com/v2/commands/201604f62595e99ca67d8ec1406482d1810
9b9
Server: nginx
Vary: Accept-Encoding
X-M2x-Version: v2.77.11
Content-Length: 21

{"status":"accepted"}franco@franco-VirtualBox:~$
```

Figure 35: M2X command sent from a Linux machine.

The new data should be shown on the Debug tab in LuvitRED:



4/26/2016, 12:09:43 PM [e7b2915c.bf3c6]

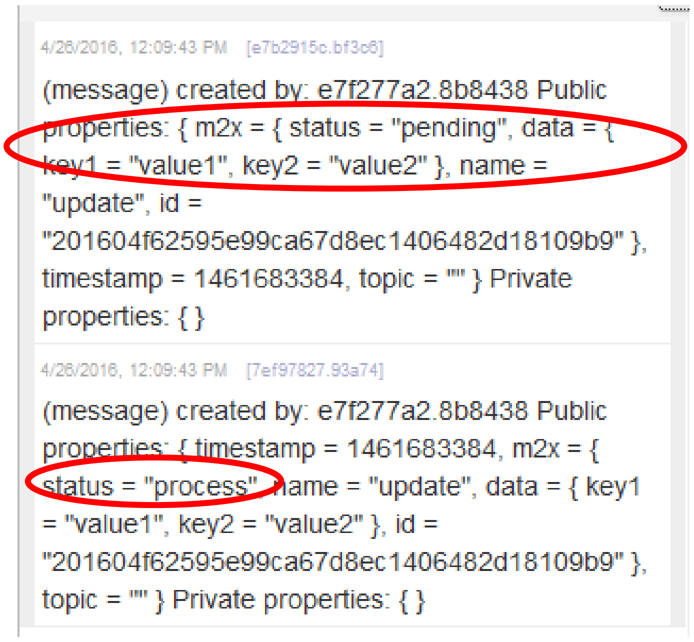
(message) created by: e7f277a2.8b8438 Public properties: { m2x = { status = "pending", data = { key1 = "value1", key2 = "value2" }, name = "update", id = "201604f62595e99ca67d8ec1406482d18109b9" }, timestamp = 1461683384, topic = "" } Private properties: { }

4/26/2016, 12:09:43 PM [7ef97827.93a74]

(message) created by: e7f277a2.8b8438 Public properties: { timestamp = 1461683384, m2x = { status = "process", name = "update", data = { key1 = "value1", key2 = "value2" }, id = "201604f62595e99ca67d8ec1406482d18109b9" }, topic = "" } Private properties: { }

Figure 36: Command coming from M2X.

The actual data looks like this:



4/26/2016, 12:09:43 PM [e7b2915c.bf3c6]

(message) created by: e7f277a2.8b8438 Public properties: { m2x = { status = "pending", data = { key1 = "value1", key2 = "value2" }, name = "update", id = "201604f62595e99ca67d8ec1406482d18109b9" }, timestamp = 1461683384, topic = "" } Private properties: { }

4/26/2016, 12:09:43 PM [7ef97827.93a74]

(message) created by: e7f277a2.8b8438 Public properties: { timestamp = 1461683384, m2x = { status = "process", name = "update", data = { key1 = "value1", key2 = "value2" }, id = "201604f62595e99ca67d8ec1406482d18109b9" }, topic = "" } Private properties: { }

Figure 37: Commands coming in and going out.

The first line on the debug is the command coming from the Linux computer and it contains the M2X command status set to ***pending***, and the data set to ***Key1="value1"*** and ***Key2="value2"*** (these items are setup on the command we sent from the Linux computer).

The second line on the debug tab is the command response we are sending out on the M2X_out node, and it contains the M2X status set to ***process***. In this way, we are telling the M2X platform that this specific command was processed by us. Of course, in this example, we are not using the command information to perform any action (we are just printing the command), but we could do something with the information first and then set the command to processed.

OPTION