AT&T Developer Program

Getting to Know the AT&T M2X Data Service and AT&T Flow Designer APIs

M2X

Time-Series Data Store and Device Management

Flow

Rapid development and deployment platform

Table of Contents

Introduction

AT&T M2X Data Service

AT&T Flow Designer

AT&T M2X Data Service is a cloud-based, time series data storage solution for network connected Machine-to-Machine (M2M) devices, applications, and services. M2X makes it easier for developers to gather real-time data from various M2M sources and translate it into meaningful information, make operational decisions based on the data, and share the data with collaborative partners.

AT&T Flow Designer is a cloud based, multi-tenant development and execution service for network connected IoT devices. Flow Designer provides support for common IoT protocols. It helps developers create custom business logic and integrates with external services such as analytics, visualization and other services.

The intent of this booklet is to guide you through a series of simple lab exercises to familiarize you with using the AT&T M2X Data Service and AT&T Flow Designer APIs and to greatly simplify its integration into your own applications.

These lab exercises are targeted toward developers and will be shown in curl and a web-based REST client (Postman for Chrome), but you can use any programming language or REST client that you prefer. In addition, the exercises use the TI MSP432 LaunchPad with the CC3100MOD BoosterPack and the AT&T Flow Designer, but again, you can use any device that you prefer, as long as it can connect to a network and make HTTP requests.

The following pages provide additional information about the M2X Data Service.

- About M2X
- Getting Started
- API Docmentation
- Client Libraries
- Tutorials
- Hardware Platforms
- Forums
- FAQ

Prerequisites

In order to complete the exercises in this workbook, you will need:

- A browser and internet connection.
- A TI MSP432 LaunchPad, CC3100MOD BoosterPack and micro-USB cord (micro to regular).
- Optionally: A REST client, such as the Chrome app Postman. See Appendix A on how to download and install Postman.
- Optionally: A command line tool called curl (officially known as cURL). See Appendix B on how to download and install curl.

Conventions Used

The following formatting conventions are used in this workbook:

Type	Style	Example
URL	Monospace	https://api.att.com/speech/v3/speechToText
Parameter	*Italic*	*client_id*
Header	**Bold**	**application/json**
Command line	Monospace	curl -X POST --data-binary...
JSON or XML	Monospace	{ "Recognition": { "Status": "OK",...
User interface elements	**Bold**	**Join Now**
File names	**Bold**	**homeBy6.wav**

Setting Up for the M2X Exercises

Pre-requisites: browser
Length: 10 minutes

In order to call the AT&T M2X Data Service APIs, you will first need to sign up for a free M2X Beta account. Once you have done so, you can create new Devices and begin to push data into the service.

See the Prerequisites section on what you need before starting the exercises.

Sign Up for an M2X Account

Follow these steps to sign up for an M2X account.

Note: If you have already signed up, simply log in.

1. Launch your browser and go to https://m2x.att.com/signup.
2. Either sign in using your GitHub account, AT&T Developer account, or scroll down to create a new username.
3. If you sign up using GitHub, then authorize AT&T to have access to your public data.
4. Enter in your name, email, and password.
5. An activation email is sent to your email address. Click on the link inside the email to activate your account

Congratulations! You're signed up.

You will also need to set up the TI LaunchPad development kit, but you will do that in Lab #2.

Lab #1: Creating and Using an M2X Device

Pre-requisites: M2X account, curl or REST client
Length: 20 minutes

In this exercise:

An M2X Device defines the attributes associated with an IP enabled device, application, or service that will be sending data to the M2X service. Once the Device has been tested, it can be used with multiple devices of the same type, which which is accomplished through creating a Distribution and launching it.

Note: Distributions are available for Professional accounts, and are note included in the free developer tier. Please see https://m2x.att.com/pricing for details.

In this exercise, you will learn how to set up an M2X Device and then send data to it. Finally, you will make a request to obtain all of the data that the Device has collected.

To make HTTP requests, you can use either a graphical client such as the Postman add-on for Chrome, or you can use the curl command line tool.

API Details

Sending Data

To send one piece of data from a Device, make an HTTP PUT request to this URL:

```
http://api-m2x.att.com/v2/devices/{device-ID}/streams/speed/value
```

where {device-ID} is the ID of the Device you created. The request contains the following headers:

Header Name	Description
Content-Type	application/json
X-M2X-KEY	Your Device's API Key

The POST body contains JSON with the following key/value pairs:

Key	Required	Value
timestamp	Required	A string that contains the time-stamp in ISO 8601 format. If omitted, the current time of the server is used.
value	Required	A number or string that contains the value

If successful, no data is returned. For more information, see the documentation at https://m2x.att.com/developer/documentation/v2/device#Update-Data-Stream-Value.

Note that you can send multiple pieces of data using the POST request documented at https://m2x.att.com/developer/documentation/v2/device#Post-Data-Stream-Values.

Retrieving Data

To retrieve data that was sent from a Device, make an HTTP GET request to this URL:

```
http://api-m2x.att.com/v2/devices/{device-ID}/streams/speed/values
```

where {device-ID} is the ID of the feed you created when creating the Device. The request should contain the following header:

Header Name	Description
X-M2X-KEY	Your Device's API Key

You can use the following optional query parameters to filter the returned data:

Parameter	Required	Description
start	Optional	The starting date and time in ISO 8601 format for data to be returned. Default is time when stream was created.
end	Optional	The ending date and time in ISO 8601 format for data to be returned. Default is current time.
limit	Optional	The maximum number of values to return. Default is 100.

If successful, JSON data is returned with the following key/value pairs:

Key	Value
start	Specified start time. null, if not specified.
end	Specified end time.
limit	Maximum number of values returned
values	An array with values and timestamps
timestamp	A string that contains the time-stamp in ISO 8601 format.
value	A number or string that contains the value

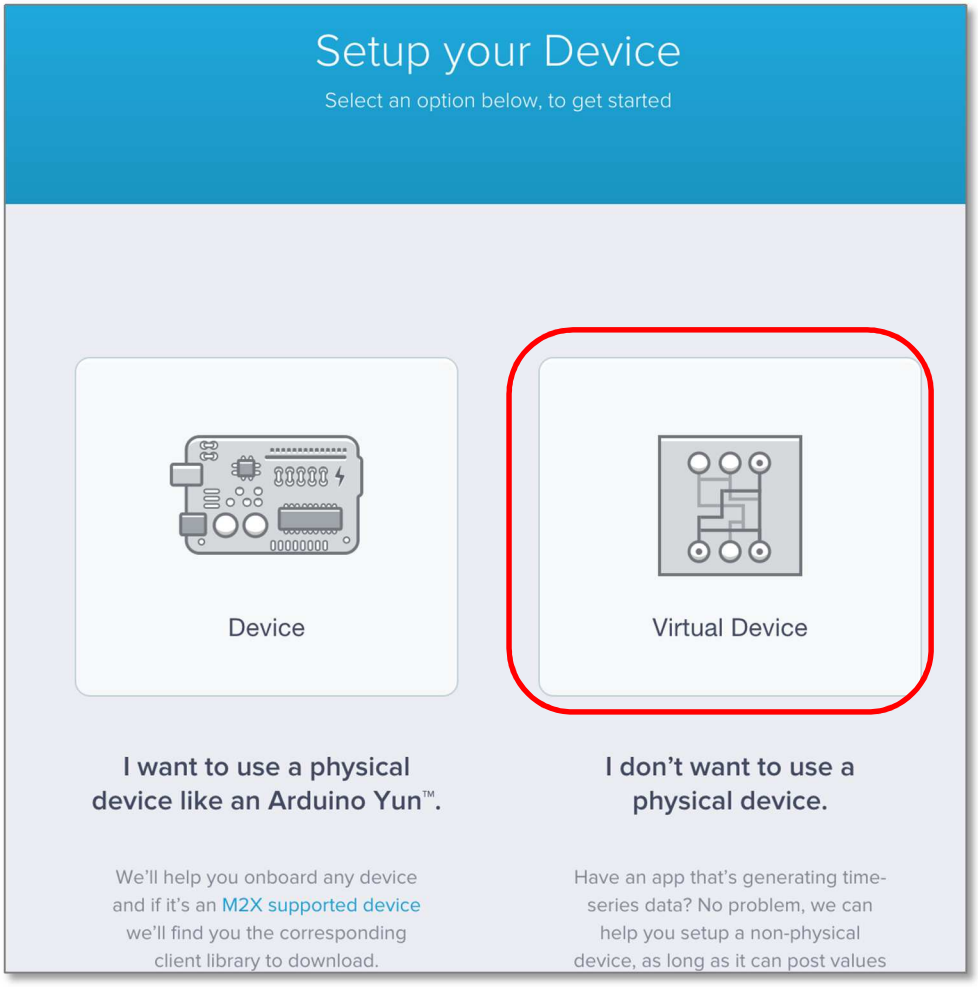
For more information, see the documentation at https://m2x.att.com/developer/documentation/v2/device#List-Data-Stream-Values.

Exercise Variant: REST Client (Postman)

The AT&T M2X Data Service can receive, store, and react to data from a data source. In order to prepare the service for your data source, the first step is to set up a Device, which is a way to prototype your Device before launch.

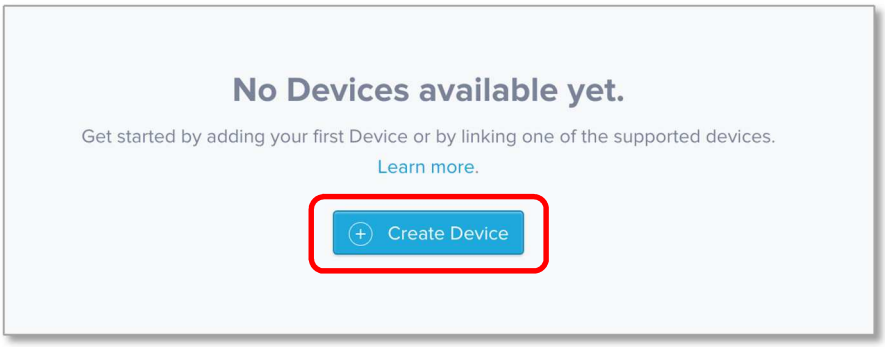
Follow these steps to set up a Device:

1. Go to https://m2x.att.com/devices.
2. You will be asked to set up your first Device as either a physical device or a virtual device. If you choose physical, it will assist you in finding the correct client library. However, we will provide you with client library for this event. Therefore, click on **Virtual Device**.

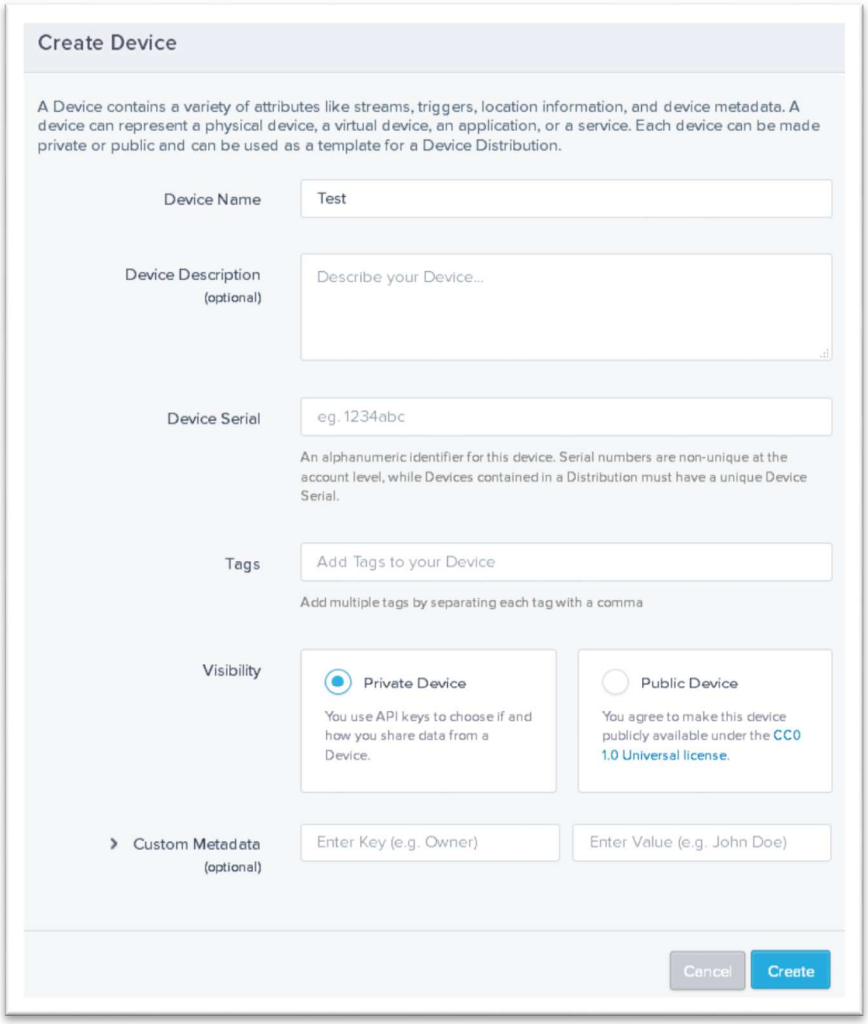


3. The first time you log in to the M2X website, you can be led through the first-time process of creating a new Device and Stream. However, for this exercise, we will use the more standard procedure. In the top right corner, click on Skip Setup.

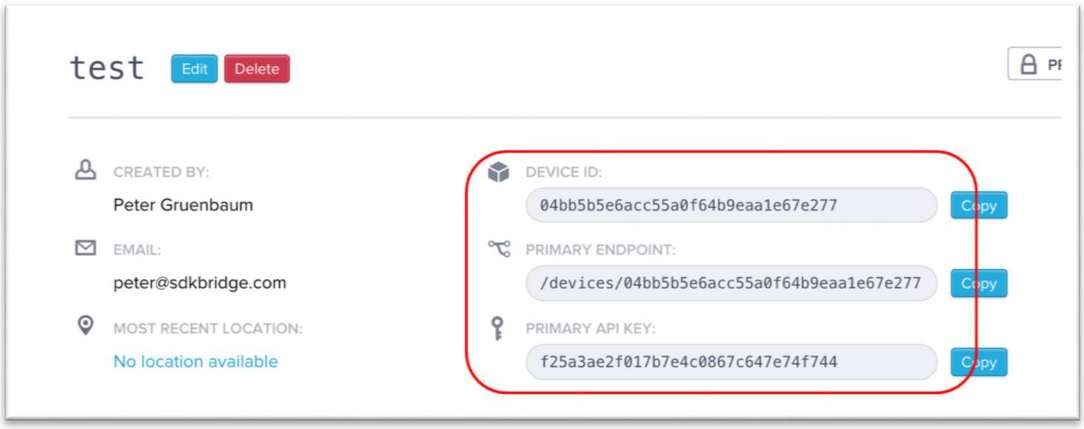
4. Click on **Create Device**.



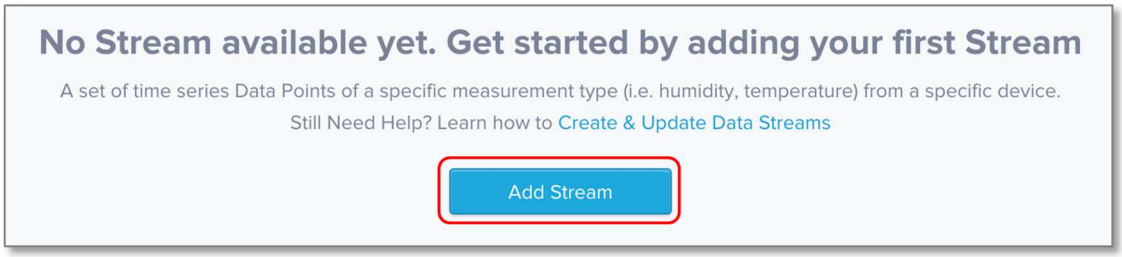
5. For **Device Name**, type "test". Leave everything else the same. Click **Create**.



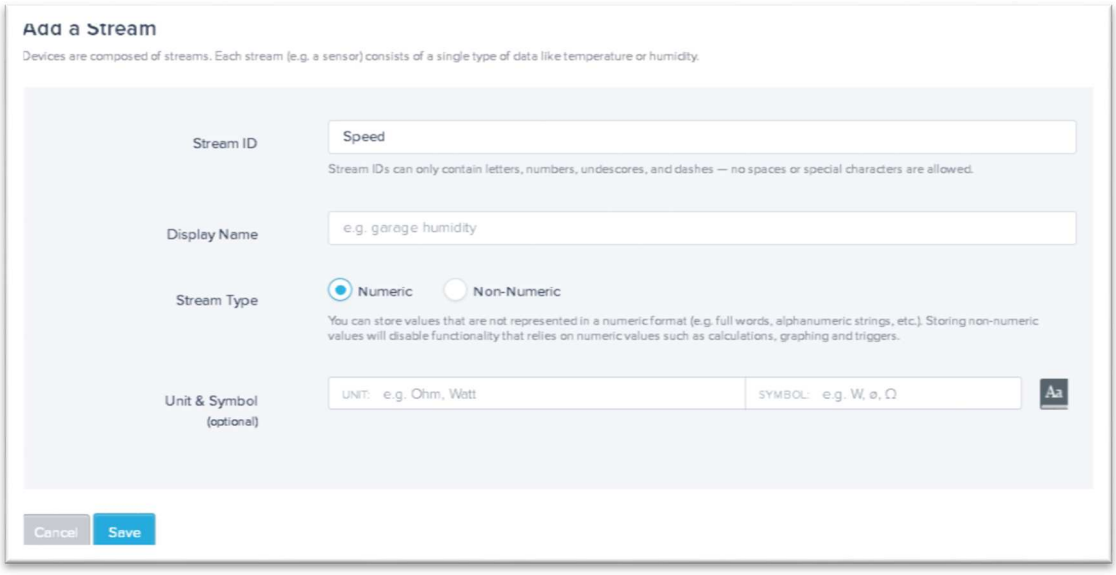
6. Your Device will have a Device ID and API Key associated with it. You will use these later.



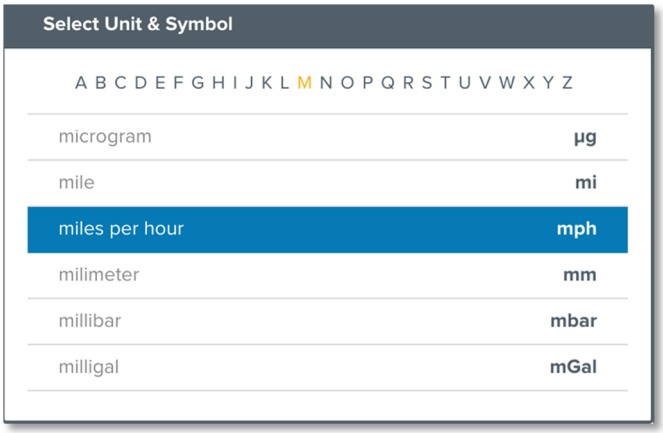
7. Let's say that you were collecting information on a fleet of vehicles, and you wanted to keep track of the speed of each vehicle as a function of time. Let's add a data stream for device. Scroll down and click on the **Add Stream** button.



8. Type in "speed" for the **Stream ID**, and then click on the **Aa** button.



9. Click on the **M**, then scroll down and select "miles per hour".



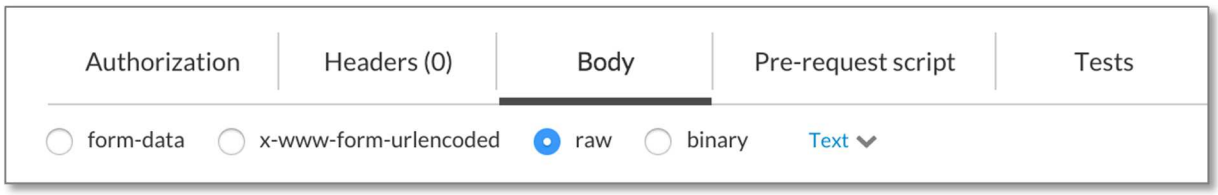
10. Click on **Save** to create the stream.

You now have a Device and a stream. Next, you'll send some data to that Device. Follow these steps to send a value of zero using the Postman add-on for Chrome:

1. Open the Postman add-on for Chrome.
2. In the **Enter request URL here** box type:

http://api-m2x.att.com/v2/devices/{device-id}/streams/speed/value

3. Replace the {device-ID} with the Device ID from the Devices page. You can use the **Copy** button to copy the ID to the clipboard.
4. Set the method dropdown to PUT
5. If the body is not visible, select **Body** tab and then the **raw** button



6. Add the following into the body box:

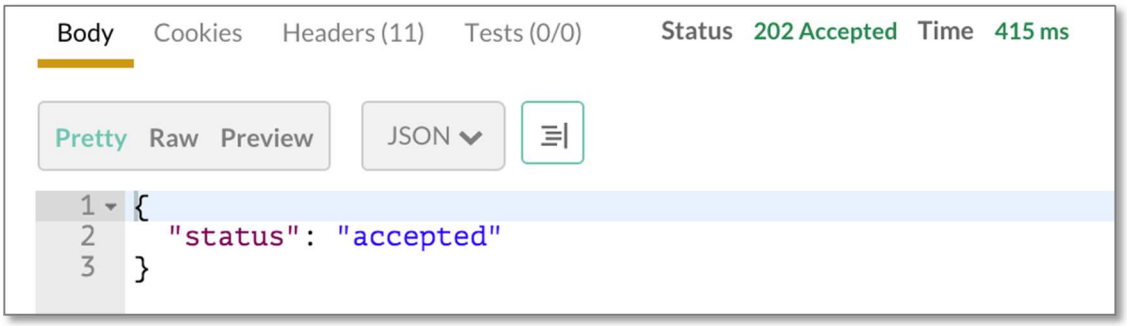
```
{ "value" : 0 }
```

Note: You can add an optional timestamp. If the timestamp is not included, as in the example above, then it uses the date and time that the request is received by the server. See Update Data Stream Value in the documentation.

Note: It is also possible to send multiple values with one request where each value can have a time-stamp. See Post Data Stream Values in the documentation.

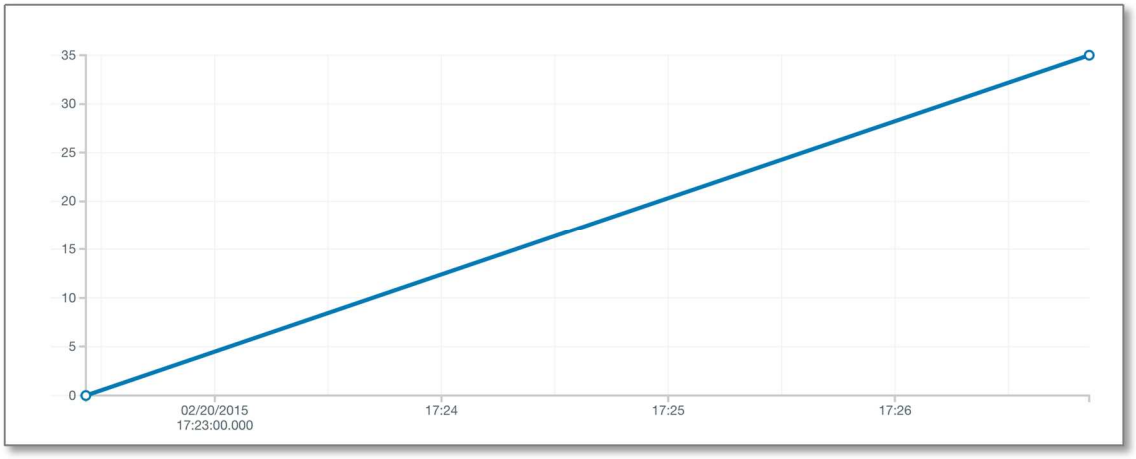
7. Click the **Headers** tab to display the headers section.
8. Add a header with a name of **Content-Type** and value of **application/json**.
9. Add a header with a name of **X-M2X-KEY** and value of your API key from the Device page. You can use the **Copy** button to copy the ID to the clipboard.

10. Click Send in the top left corner. If successful, then an HTTP 202 response will be returned that says it was accepted.

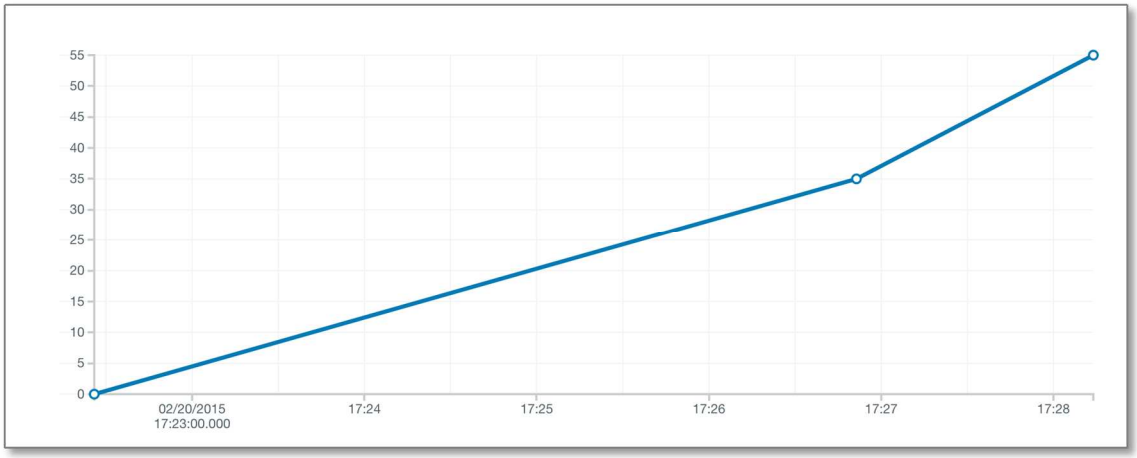


11. Go back to your Device page and scroll down until you see the streams. There will be one stream for speed. Take a look at the graph of values. Note that the graph has one data point, which is at zero.

12. Return to Postman. In the POST body, change the value from "0" to "35". Click Send again. Your graph will immediately show a slope from 0 to 35, with a time scale on the x-axis.



13. Try it again, this time putting in a value of 55. You can see that each time it updates the graph.



10. Now, let's use an HTTP request to retrieve those values. Make these changes:

    a. Change the method dropdown from PUT to GET.

    b. Change the URL so that it ends with **/values** instead of **/value**.

Click **Send**. This will return all values so far. Your output should look something like this, with the most recent values displayed first:


```
{
    "limit": 100,
    "end": "2015-02-20T23:18:18.975Z",
    "values": [
        {
            "timestamp": "2015-02-20T17:28:13.880Z",
            "value": 55
        },
        {
            "timestamp": "2015-02-20T17:26:51.375Z",
            "value": 35
```

```
        },
        {
            "timestamp": "2015-02-20T17:22:25.918Z",
            "value": 0
        }
    ]
}
```

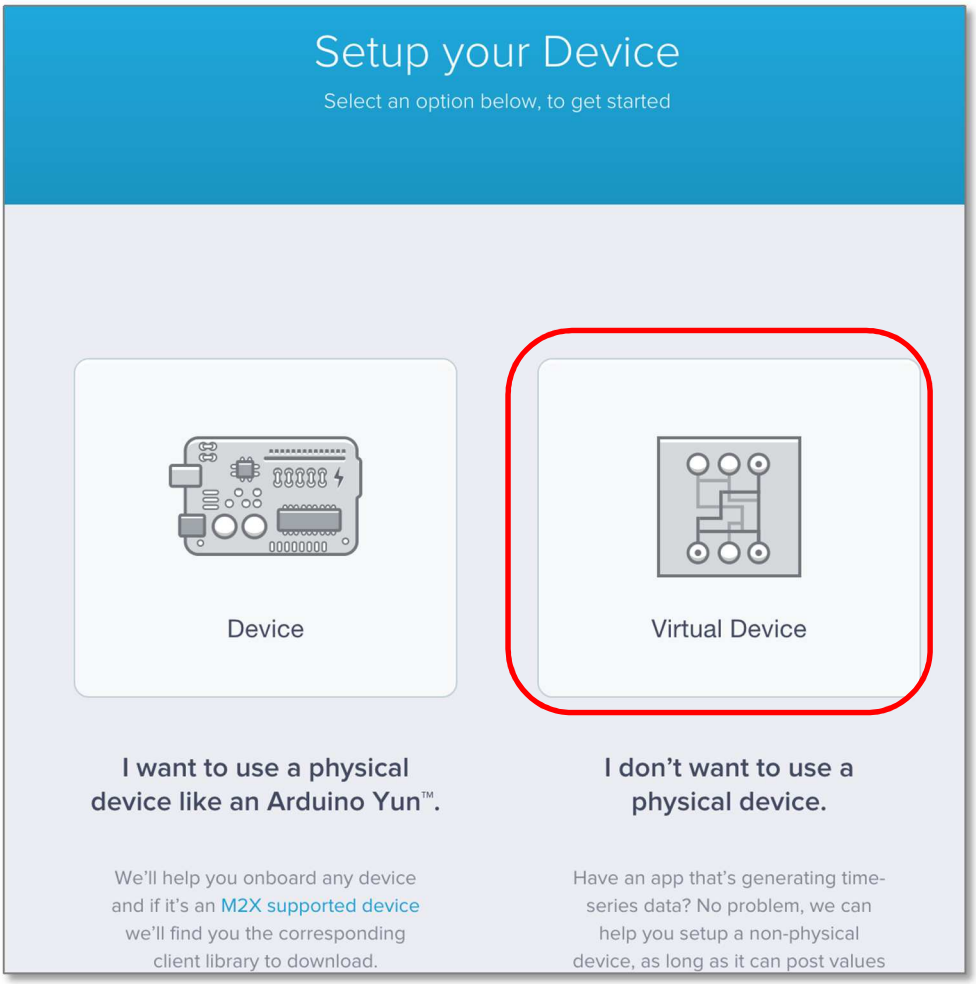
Exercise Variant: Command Line Tool (curl)

Note: The curl exercises are designed for Unix-based terminals, such as Unix, MacOS, and Cygwin. They use the backslash (\) in order to continue onto more than one line. To run them on a Windows command prompt, substitute each backslash for a caret (^), or remove the backslashes and put it all on one line.

The AT&T M2X Data Service can receive, store, and react to data from a data source. In order to prepare the service for your data source, the first step is to set up a Device, which is a way to prototype your Device before launch.

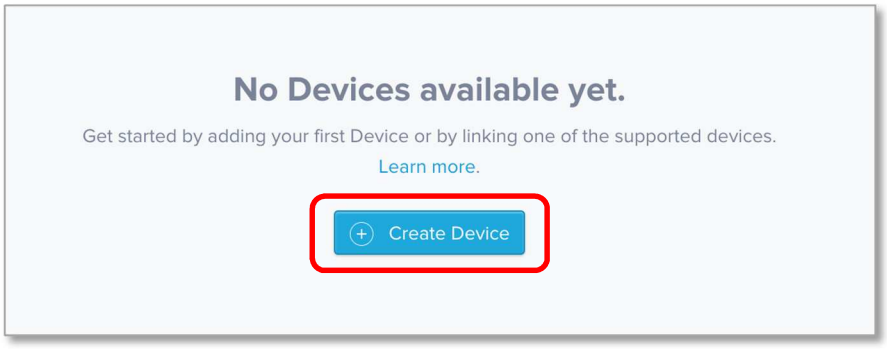
Follow these steps to set up a Device:

1. Go to https://m2x.att.com/devices.
2. You will be asked to set up your first Device as either a physical device or a virtual device. If you choose physical, it will assist you in finding the correct client library. However, we will provide you with client library for this event. Therefore, click on **Virtual Device**.

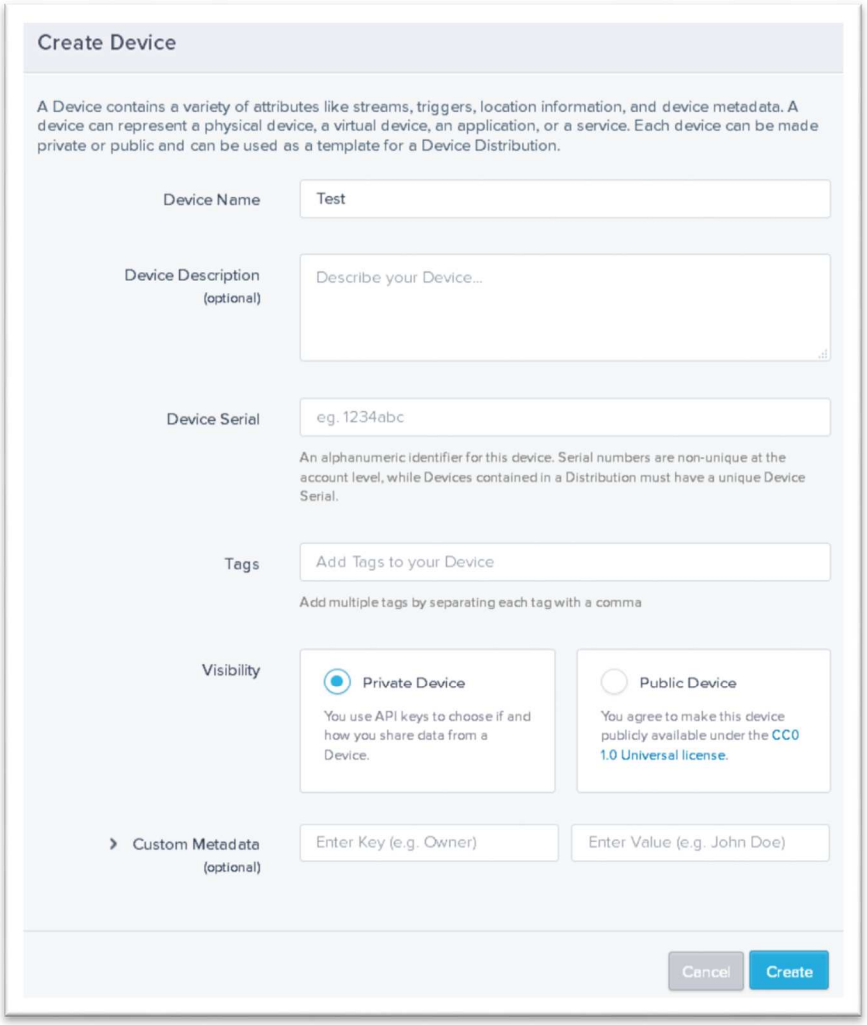


3. The first time you log in to the M2X website, you can be led through the first-time process of creating a new Device and Stream. However, for this exercise, we will use the more standard procedure. In the top right corner, click on Skip Setup.

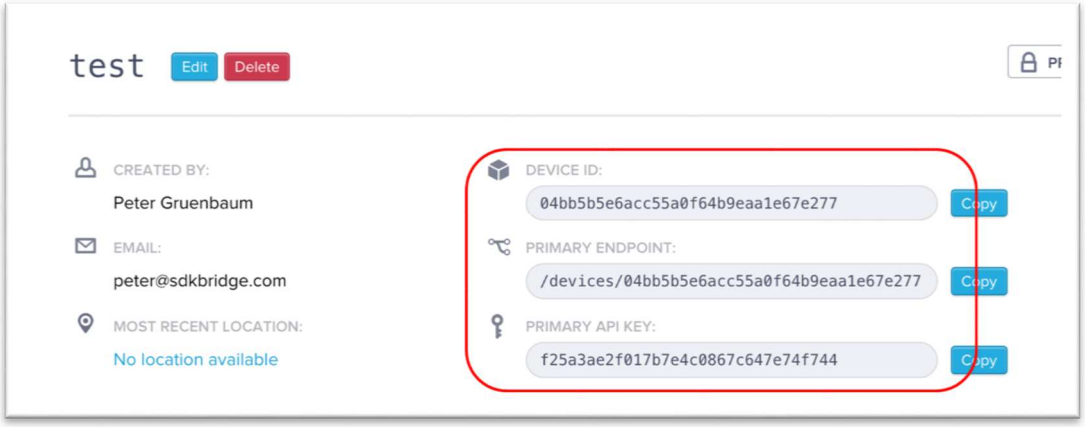
4. Click on **Create Device**.



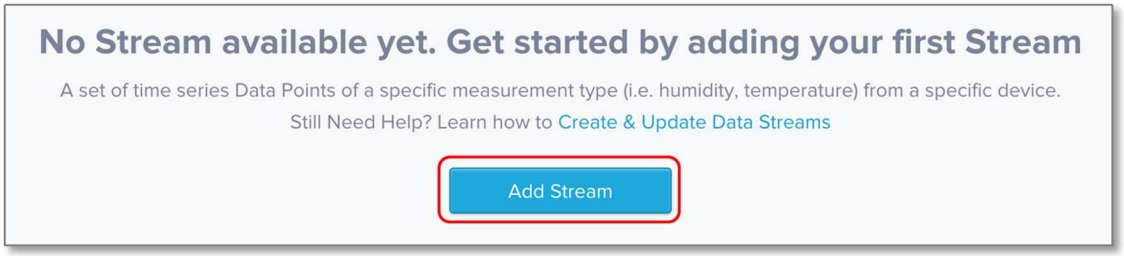
5. For **Device Name**, type "test". Leave everything else the same. Click **Create**.



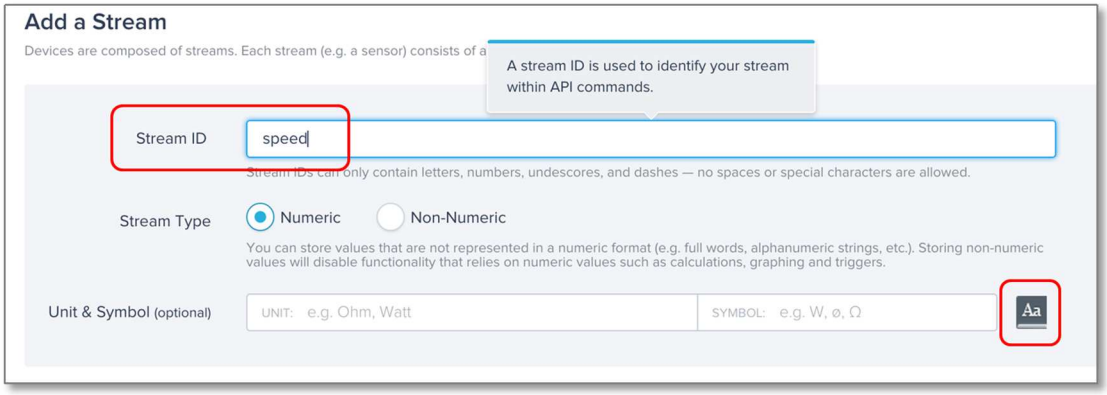
6. Your Device will have a Device ID and API Key associated with it. You will use these later.



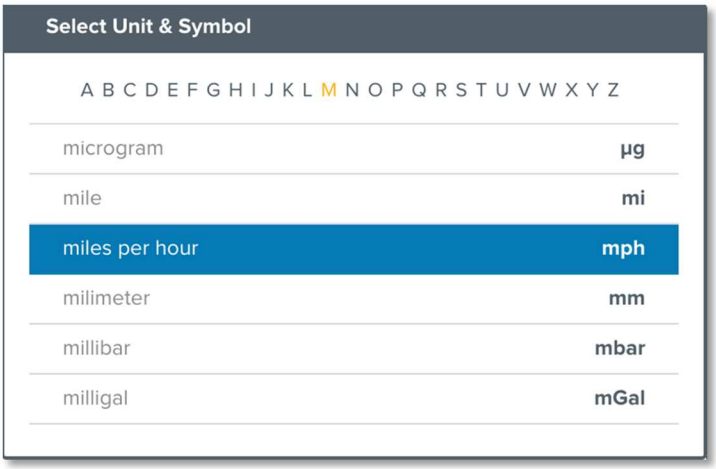
7. Let's say that you were collecting information on a fleet of vehicles, and you wanted to keep track of the speed of each vehicle as a function of time. Let's add a data stream for device. Scroll down and click on the **Add Stream** button.



8. Type in "speed" for the **Stream ID**, and then click on the **Aa** button.



9. Click on the **M**, then scroll down and select "miles per hour".



10. Click on **Save** to create the stream.

You now have a Device and a stream. Next, you'll send some data to that Device. To do this, you will send a PUT request with the following JSON in the PUT body, which contains a value of zero:

```
{"value": 0}
```

Note: You can add an optional timestamp. If the timestamp is not included, as in the example above, then it uses the date and time that the request is received by the server. See Update

Data Stream Value in the documentation. Note that if you include the timestamp, the you may need to add the following to your **.bashrc** file to have the timestamp be in the correct format:

```
alias utc='date -u +"%Y-%m-%dT%H:%M:%SZ"'
```

Note: It is also possible to send multiple values with one request where each value can have a time-stamp. Read the previous note for information on timestamp format. See Post Data Stream Values in the documentation for more information.

```
{
  "values": [
    { "timestamp": "2015-04-07T19:15:00.624Z", "value": 32 },
    { "timestamp": "2015-04-07T20:15:00.522Z", "value": 30 }
  ]
}
```

Follow these steps to send a value of zero using curl:

1. Paste the following into a text editor. (You may have to put in the line breaks after pasting.)

```
curl --request PUT \
--header "Content-Type: application/json" \
--header "X-M2X-KEY: {API-key}" \
--data '{"value": 0}' \
http://api-m2x.att.com/v2/devices/{device-ID}/streams/speed/value?pretty=true
```

2. Replace {API-key} with your Device's API key and {device-ID} with your Device ID, which you can find on the Devices webpage. You can use the **Copy** buttons to copy the values to the clipboard.

Note: Remove the curly brackets when replacing {API-key} and {device-ID}. For example, your final version should look something like this:

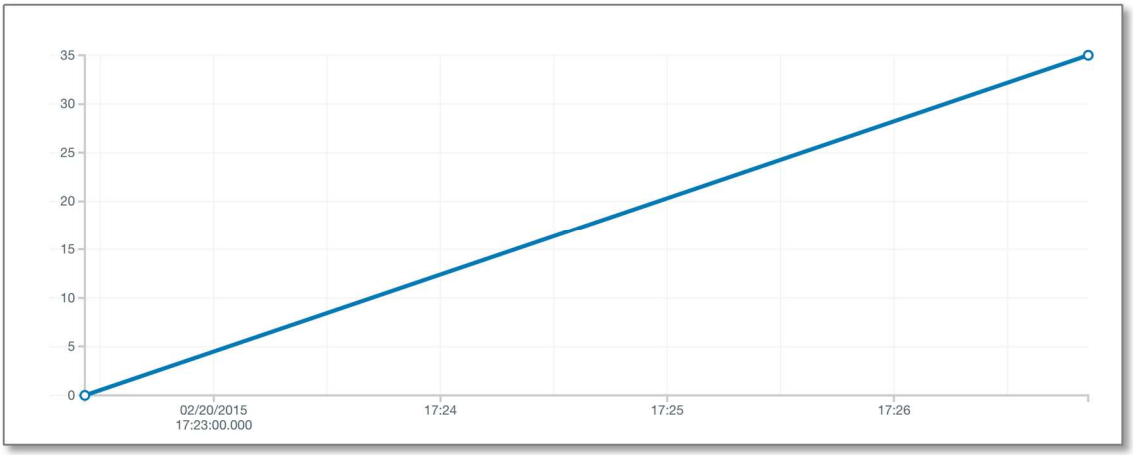
```
curl --request PUT \
--header "Content-Type: application/json" \
--header "X-M2X-KEY: 8a31911e8b2c361ad535fcb2da6a7064" \
--data '{"value": 0}' \
http://api-m2x.att.com/v2/devices/84a7221ec4a9a3b19c30/streams/speed/value?pretty=true
```

3. Open up a terminal (command prompt in Windows) that has curl.
4. Paste the edited curl command into the terminal and hit Enter. If successful, then the response will look something like this.

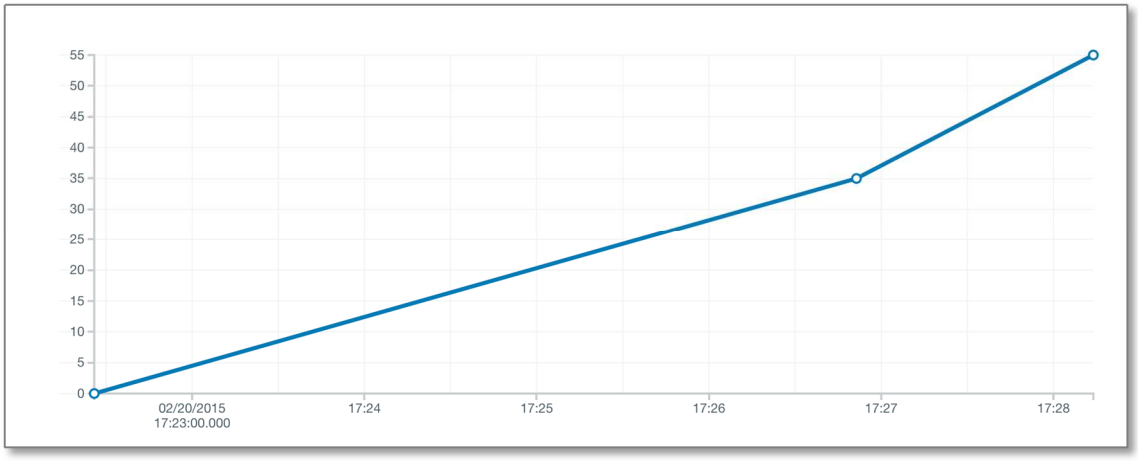
```
{
  "status":"accepted"
}
```

5. Go back to your Device page and scroll down until you see the streams. There will be one stream for speed. Scroll down to see a graph. Note that the graph has one data point, which is at zero.

6. In your text editor, change the value from zero to 35. Copy and paste the modified curl command into the terminal. Your graph will immediately show a slope from 0 to 35, with a time scale on the x-axis.



7. Try it again, this time putting in a value of 55. You can see that each time it updates the graph.



10. Now, let's use an HTTP request to retrieve those values. Change the PUT request to GET, change `value` to `values` and remove the `--data`. This will return all values so far. The curl command should look like this, with the ID and key in curly brackets replaced by your actual ID and key:

```
curl --request GET --header "Content-Type: application/json" \
--header "X-M2X-KEY: {API-key}" \
http://api-m2x.att.com/v2/devices/{device-ID}/streams/speed/values?pretty=true
```

11. Your output should look something like this, with the most recent values displayed first:

```
{
  "limit":100,
  "end":"2015-02-21T23:27:49.673Z",
  "values":[
    {
      "timestamp":"2015-02-21T23:11:24.780Z",
      "value":0.0
    },
    {
      "timestamp":"2015-02-20T17:28:13.880Z",
      "value":55.0
    },
    {
      "timestamp":"2015-02-20T17:26:51.375Z",
      "value":35.0
    }
  ]
}
```

Summary

You've learned how to create an M2X Device with a stream using the AT&T M2X Data Service website. You've also learned how to send data to that stream and retrieve it.

Lab #2: TI LaunchPad and M2X

Pre-requisites: browser, TI MSP432 LaunchPad, CC3100MOD BoosterPack, and micro-USB cord (micro to regular).
Length: 30 minutes

In this exercise:

You can use the AT&T M2X Data Service with any device that can make HTTP requests. In this exercise, you will be using the TI MSP432 LaunchPad with the CC3100MOD BoosterPack that can connect to WiFi. We will provide you with baseline code that contains classes making it easy to communicate with both the device and the M2X service. You will add code that reads data from the device and makes the calls to the M2X service.

In this exercise, you will read pushbutton data from the TI LaunchPad and send it to an M2X stream.

Note: If you have trouble getting the device to work properly, see the Troubleshooting section.

Setting Up the Energia IDE and Installing the LaunchPad Drivers

The Energia IDE allows you to communicate with the TI device. Follow these steps to download it and set it up.

For Mac OS

1. From your USB drive, open the file labeled **energia-0101E0017-macosx-signed.dmg**. Alternatively, you can download it from here: http://energia.nu/download/
2. Open the dmg file and drag **Energia** into the **Applications** folder.
3. Install the drivers by double-clicking the driver files that end in **.pkg**
4. Open your **Applications** folder and double-click on **Energia.**

Note: You may need to install a legacy Java version. Follow the instructions to do so. If you have trouble downloading the Java installer, a file called **javaforosx.dmg** is on your USB drive.

For Windows

1. Install the MSP432 drivers. From the folder on your USB drive called **xds110_drivers**. (Alternatvely, download the zip file from this link: http://energia.nu/files/xds110_drivers.zip)

2. If you have 64-bit system, double-click on **DPInst64.exe**. If you have a 32-bit system, double-click on **DPInst.exe**. Follow the instructions.
3. From your USB drive, copy the folder labeled **energia-0101E0017** to your hard drive. Alternatively, you can download it from here and extract it: http://energia.nu/download/

Note: Make sure the file path does not include spaces. This will lead to compile errors with the MSP432 LaunchPad. To be safe, you can install under a new folder such as C:\ti\energia.

4. Open the copied folder and double-click on **energia.exe**. (If your computer is set up not to show extensions, this might appear as **energia**.)

Note: In some rare cases your driver installation application may not work, but you can still manually install the drivers under your Windows Device Manager. Right click the unrecognized devices and click **Install the Drivers**. Then navigate to the folder containing the drivers.

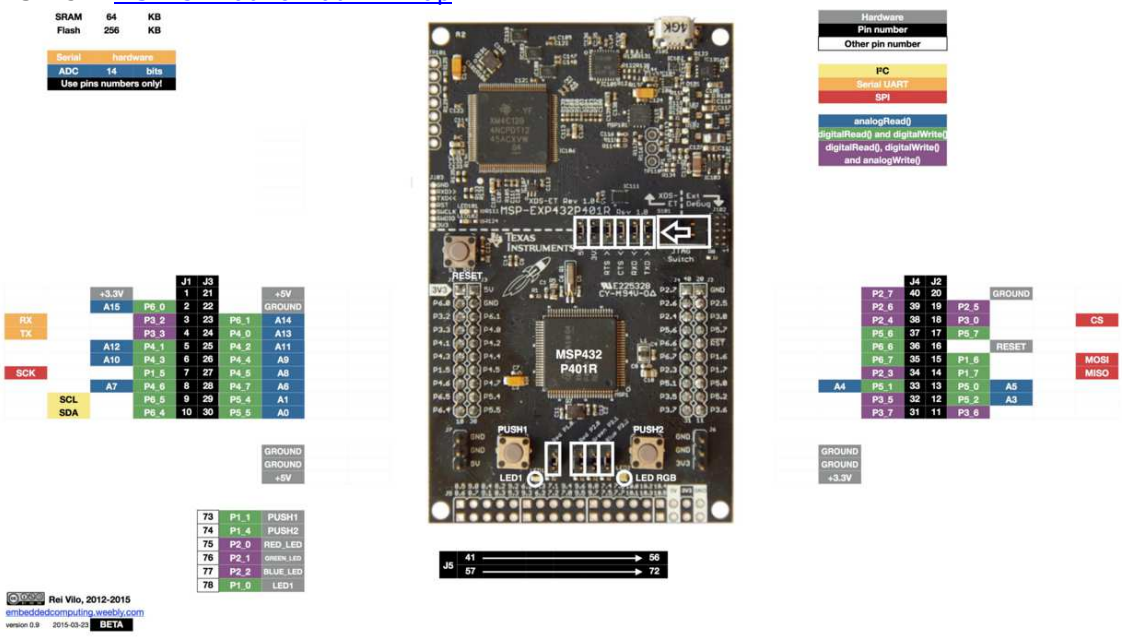
Note: For Windows 8 and 10, you may need to disable your signed driver feature. If you receive an error, follow this guide: https://learn.sparkfun.com/tutorials/disabling-driver-signature-on-windows-8

Setting Up the TI MSP432 LaunchPad and CC3100MOD BoosterPack

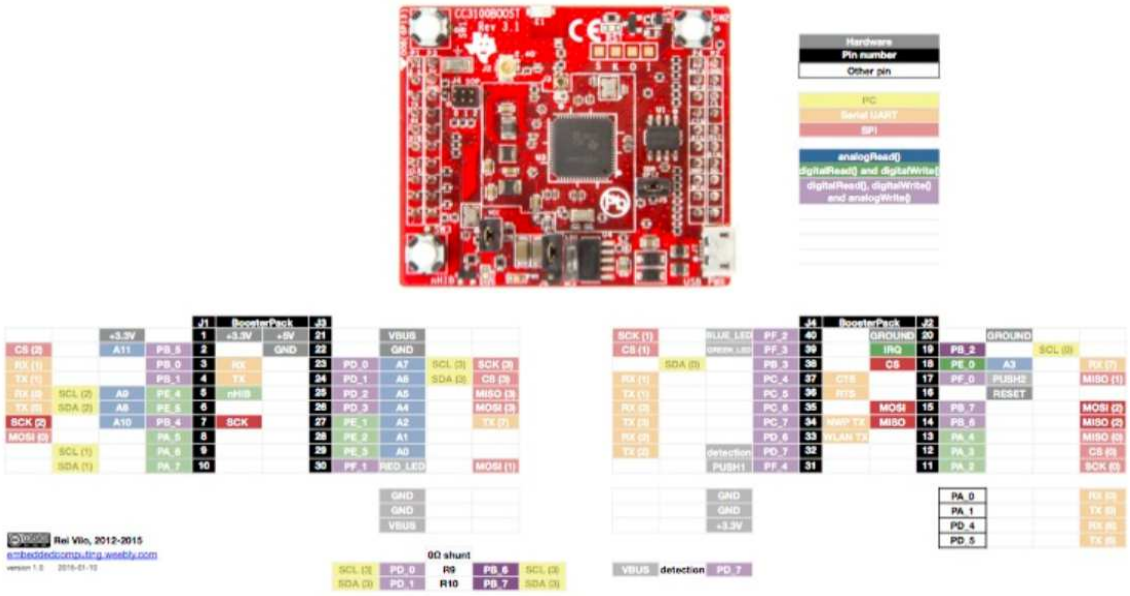
The MSP LaunchPad is a low-power microcontroller. To connect it to the internet, you need to add the CC3100MOD BoosterPack.

For your convenience you can find the PIN Maps for each board in our DevLabs at the following locations:

- MSP432: MSP432 LaunchPad Pin Map

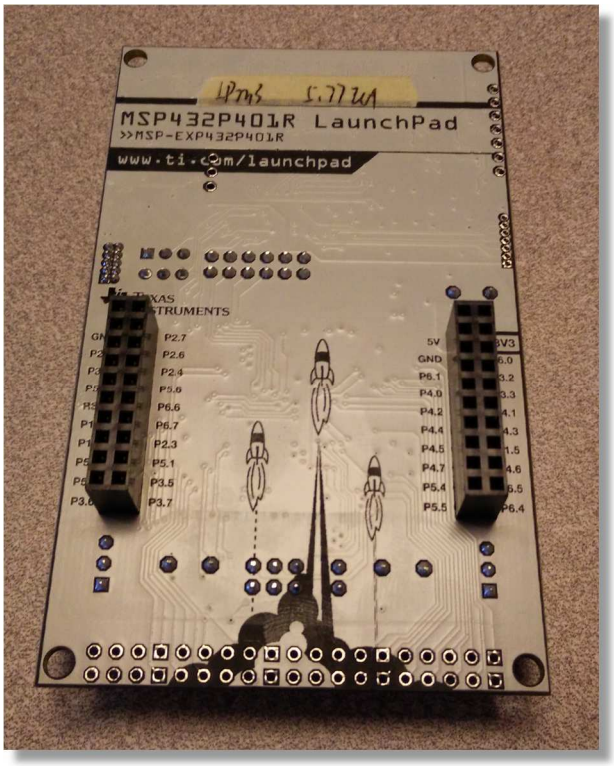


- CC3100: CC3100 Wifi BoosterPack Pin Map



Follow these steps to connect them:

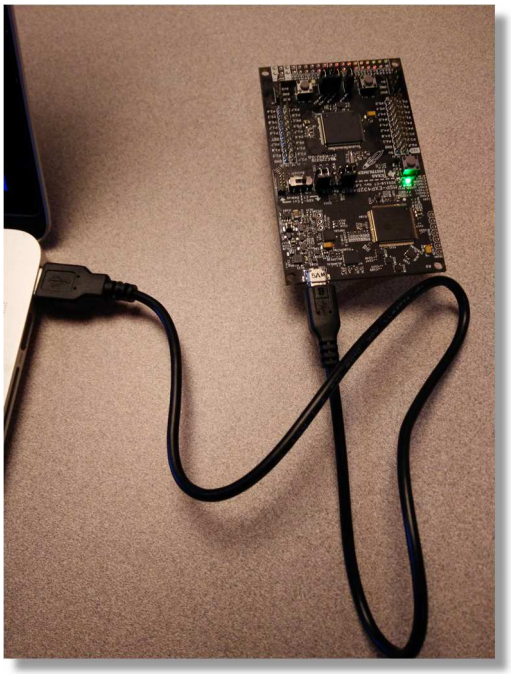
1. Place the MSP432 face-down so that the writing is right-side-up.



2. Add the BoosterPack , carefully sliding the pins into the slots. The writing should also be right-side-up. (Note that you could put the BoosterPack on the front side, but it will be hard to reach the MSP pushbuttons, which will be used in the lab.)

3. Attach the micro-USB cord to the micro-USB port at the end of the MSP432. Attach the other end of the cord to your computer. (Note: the BoosterPack also has a micro-USB port, but you will not be using it.)

4. Turn the devices over so that the MSP432 is on top.



Creating Code to Read Pushbutton Data

Let's consider a scenario where a door to a controlled environment (like a refridgerator) should only be open for a certain amount of time. You want to collect data on how long it stays open. We can simulate that by using the pushbutton on the LaunchPad. Follow these steps to add code in Energia that measures that time the pushbutton is pressed and sends it to M2X.

1. In Energia, from the **File** menu, choose **Open.** Navigate to the folder **Lab2** on your USB drive. Open the folder **lab2** and then open file **lab2.ino**. This contains the basic code to connect to the WiFi and the device.
2. In lines **12** and **13**, change **<ssid>** to your wireless SSID, and **<password>** to your wireless password.
3. In lines **18**, **19**, and **20**, change **<device ID>** to your device ID, **<API key>** to your Primary API key and, and **<stream name>** to **presstime**, which is the name of the stream that will collect the data. You can get these values from your device page in the M2X website.
4. At around line **24** (before the **setup** function), add code to declare the following variables:

```
int btn1 = 1;  // The button state
boolean pushed = false; // True when the button is pushed
int whenPushed = 0;  // The time in milliseconds from when the button is pushed.
```

   Make sure the variables are initialized to proper values: **btn1** should be **1**, **pushed** should be **false** and **whenPushed** should be **0**.

5. At around line **58**, in the **loop** method, add the following code. This will keep track of the button state. When it has been pushed and then released, it calculates the time that's elapsed and then sends it to M2X. The **millis** function returns the number of

milliseconds from when the code started. The **Serial.print** and **Serial.printf** lines will send debugging information to the serial monitor.

```
// Read the pushbutton state. 1 for not pushed, 0 for pushed.
btn1 = digitalRead(PUSH1);

if (btn1 == 0 && !pushed) {
  // If pushed, then store the time
  pushed = true;
  whenPushed = millis();
} else if (btn1 == 1 && pushed) {
  // Reset pushed
  pushed = false;

  // Released. Calculate the time elapsed
  double seconds = (millis() - whenPushed) / 1000.0;
  Serial.print("Seconds pushed: ");
  Serial.println(seconds);

  // Send to M2X
  int response = m2xClient.updateStreamValue(deviceId, streamName, seconds);
  Serial.print("M2X client response code: ");
  Serial.println(response);

  // If the response is an error, then send into infinite loop to stop loop
  if (response == -1) {
    Serial.println("Error occurred");
    while (1);
  }
}
```

Note: Sometimes copying and pasting text from a PDF results in strange line breaks that can cause problems with the code. If you are seeing this issue, copy and paste the code from the file **lab2code.txt** in the **Lab2** folder on your USB drive.

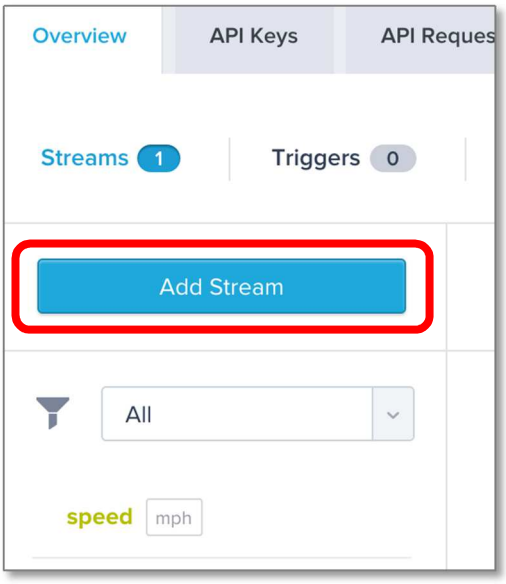
Alternatively you can open the file **lab2_with_Code.ino** from **lab2_with_code** folder. All the changes are already incorporated in this file.

6. From the **File** menu, click **Save**.

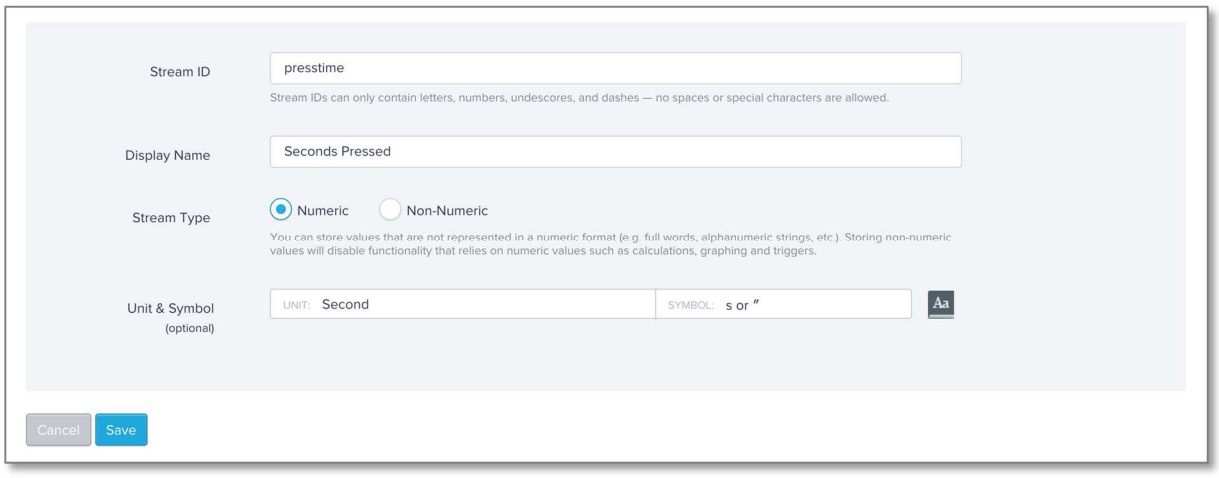
Creating a Stream for the Pushbutton Data

To run the code, first we need to set up a stream called "presstime" to collect the time when the button is pressed. Follow these steps:

1. Return to your Device page on the M2X. Click the **Add Stream** button.



2. Type in "**presstime**" for the Stream ID. For the display name, type "**Seconds Pressed**." Then click the **Aa** button on the **Units & Symbols** line. Click the letter **S**, and then choose **Second**.



3. Click the **Save** button.

Now return to Energia to run the code on your device.

Running the Code on the Device

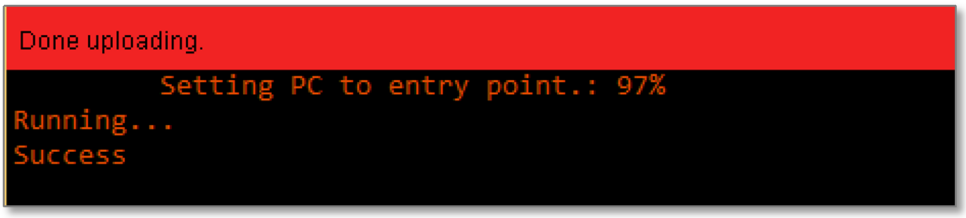
Use these steps to upload the code to the device and run it. Make sure the device is plugged into your USB drive.

1. From the **Tools** menu, select **Board**, and then select **LaunchPad w/ msp432 EMT (48 MHz).**

2. From the **Tools** menu, select **Serial Port**, and then select the appropriate item in the list. (Typically the first port that starts with **/dev/tty.usbmodem** on a Mac and the first port that starts with **COM** on a Windows computer.)

Note: You can verify the correct serial port under Windows Device Manager (Windows) or Mac USB Device Tree (Mac).

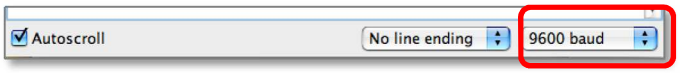
3. From the **File** menu, select **Upload**. (Or click the Upload button. ) This will upload the code to the device and run it. This will take a couple of minutes. When it is done, you will see text at the bottom of the window that says "Done uploading" and "Success".



Note: If the code does not compile successfully there is a chance something was not copied and pasted properly from the exercise book PDF. You can either try and troubleshoot yourself or just go ahead and open **lab2_with_Code.ino** from **lab2_with_code** folder. You still have to change ssid, pass, deviceId, m2xKey and streamName values manually!

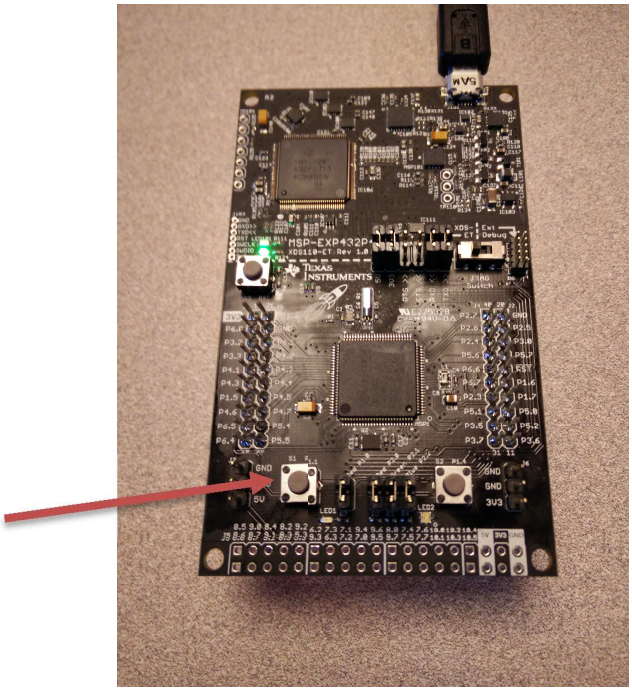
4. From the **Tools** menu, select **Serial Monitor**. This will open a window that will display debugging information.

   Note: If no debugging information is visible, then try changing the serial port from the **Tools** menu. If the debugging information is not readable, change the baud rate at the bottom corner to 9600 baud.



5. In the serial monitor window, you'll that it connected to WiFi. (It prints dots until it connects.)

6. Position the board so that the writing is right-side-up and the micro-USB cord is away from you. You'll see two pushbuttons on the side closest to you. Quickly press the one on the left.



7. Return to the M2X website. The first point should have appeared on your graph, indicating how long the button press lasted. Try it again, holding the button down longer and watch as the data appears. It takes 5 to 10 seconds for the new data to appear in the graph.

Troubleshooting

You may be following the steps in this exercise and find that data is not being sent to your M2X stream. If you are receiving anything other than a 202 response, this means that your connection to the WiFi or M2X platform is not working. Verify that the WiFi and M2X Device information is all correct.

Summary

You've learned how to import code into the Energia IDE and to modify it so that it reads pushbutton data and sends data to an M2X stream.

Lab #3: Triggers

Pre-requisites: configured M2X Device as in Lab #2, configured LaunchPad sending data to M2X as in Lab #2, curl or REST client
Length: 20 minutes

In this exercise:

The AT&T M2X Data Service allows you to create triggers so that when certain conditions are fulfilled, the M2X service will make HTTP requests to a URL that you have specified, with. This allows you to be notified when a value becomes too high or too low, or hits a certain value. Triggers can either be created through the web interface or through the API.

In this exercise, we will lead you through creating a trigger and then sending data to the stream that triggers it.

API Details

To create a new trigger through the API , make an HTTP POST request to this URL:

```
http://api-m2x.att.com/v2/devices/{device-ID}/triggers
```

where {device-ID} is the ID of the device. The POST contains the following headers:

Header Name	Description
Content-Type	application/json
X-M2X-KEY	Your Data Source's API Key

The POST body contains JSON with the following key/value pairs:

Key	Required	Value
name	Required	The name of the trigger
stream	Required	The name of the stream
condition	Required	The condition, which can be <, <=, =, > or >=
value	Required	The value to use with the condition
callback_url	Required	The URL that will be called if data matches the condition
status	Required	Either "enabled" or "disabled"
send_location	Required	If true, the trigger payload will include the last set of long/lat/altitude values stored for the device. If false, not included.

If successful, JSON is returned with the following key/value pairs:

Key	Value
id	The ID of the trigger
name	The name of the trigger
stream	The name of the stream
condition	The condition, which can be <, <=, =, > or >=
value	The value to use with the condition
callback_url	The URL that will be called if data matches the condition
url	The URL of the trigger
status	Either "enabled" or "disabled"
send_location	If true, the trigger payload will include the last set of long/lat/altitude values stored for the device. If false, not included.
created	Timestamp for when the trigger was created
updated	Timestamp for when the trigger was last updated

See the following documentation pages for more on triggers:

- List triggers: https://m2x.att.com/developer/documentation/v2/device#List-Triggers
- Create trigger: https://m2x.att.com/developer/documentation/v2/device#Create-Trigger
- View Trigger: https://m2x.att.com/developer/documentation/v2/device#View-Trigger
- Update Trigger: https://m2x.att.com/developer/documentation/v2/device#Update-Trigger
- Test Trigger: https://m2x.att.com/developer/documentation/v2/device#Test-Trigger
- Delete Trigger: https://m2x.att.com/developer/documentation/v2/device#Delete-Trigger

Setting Up a Server to Handle a Trigger

Let's go back to the example of a door to a controlled environment that should stay open only for a certain amount of time. Let's say that, in addition to collecting data every time the door is open, you want to be notified every time the door is opened for more than 5 seconds, since this requires taking immediate action. We can set up a trigger to do this, which will make a POST request to a URL. Ultimately, we want to do something like have the POST request result in sending a text message so that we are informed wherever we are.

First, we need to set up a server URL to receive the notification. We can do this by using a site called RequestBin, which allows you to set up URLs for testing purposes. Follow these steps:

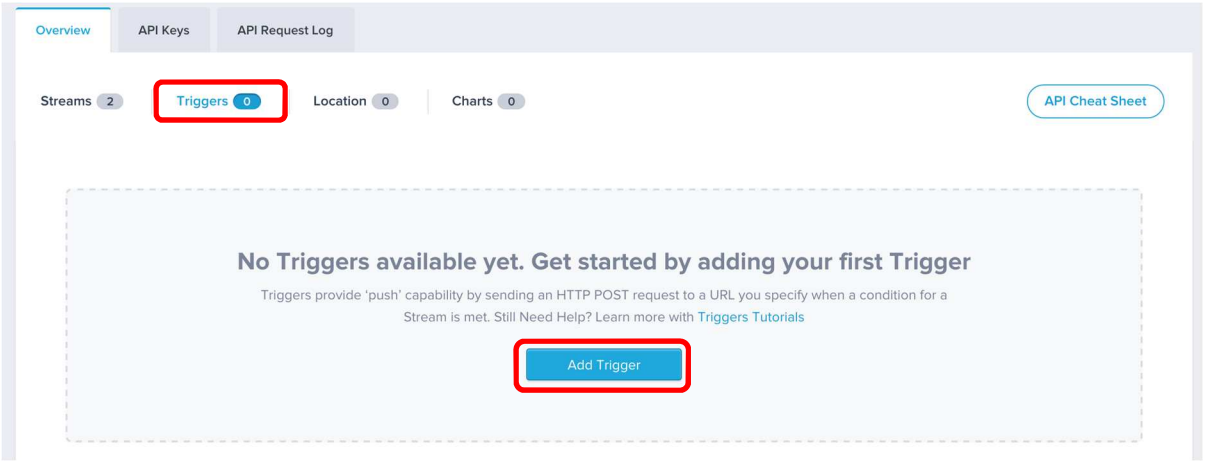
1. In a new browser tab or window, navigate to http://requestb.in/
2. Click on **Create a RequestBin.**

3. This will create a URL that we can use when creating a trigger. Save the page so that you can copy and paste the URL.

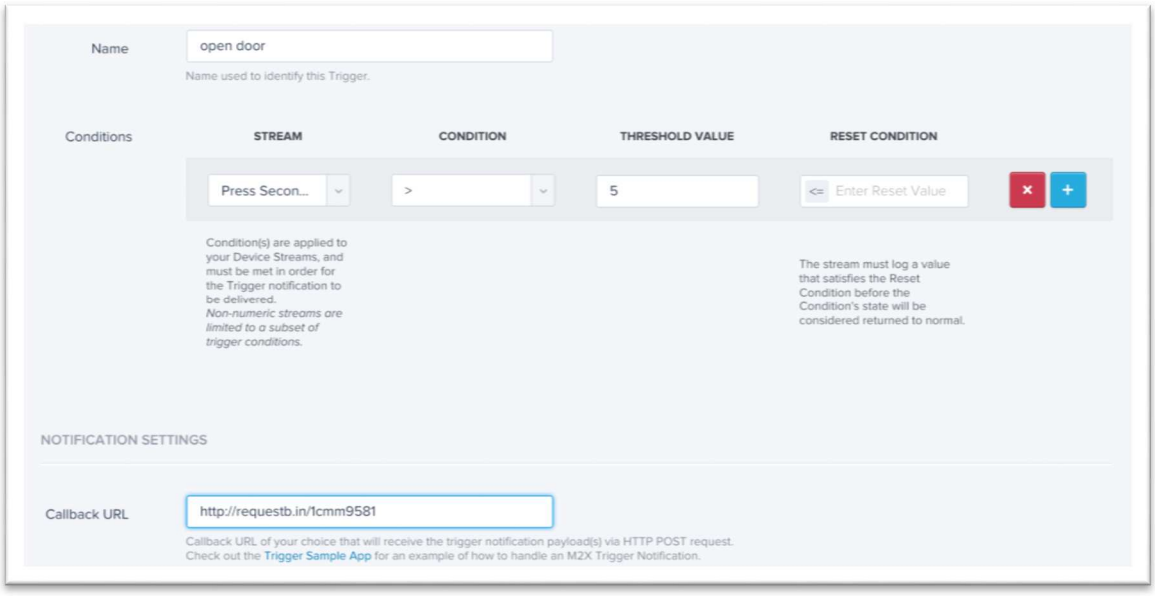
Setting Up a Trigger

Now that we have a place to receive the trigger POST, let's set up the trigger. Use the following steps to create a trigger for the condition where the button is pressed for more than 5 seconds.

1. In your browser, open your Device page in another tab or browser and click the **Trigger** tab. Then click **Add Trigger**.

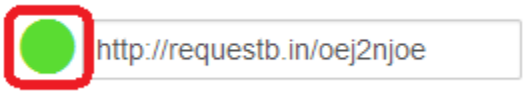


2. For the **Trigger Name**, type "open door". For the **Stream**, choose **Seconds Pressed**. For the **Condition**, choose **> is greater than**. For the threshold value, type "5". For the **Callback URL**, copy and paste the URL from the RequestBin page. Click **Save**.



Using the Trigger

1. Press the button for more than 5 seconds. Check your RequestBin website and refresh it by clicking on the circle next to your URL, in the top right corner of the page.



2. You will see the data that was posted to the URL, which will take the following form. As you can see, it contains the device ID, the stream name, the trigger name, the condition and threshold, as well as the value that crossed the threshold and its timestamp. If this were a real server that it was posting data to, you could parse the incoming data and take action.

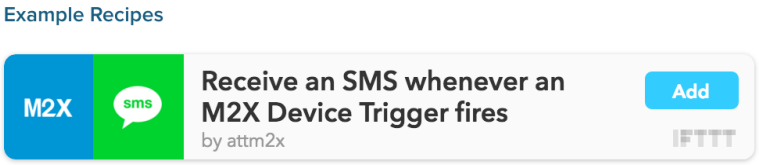

```
{"trigger":"open door","timestamp":"2016-02-
15T21:05:31.454Z","event":"fired","device":{"id":"c656bd781d367d3517e4f
0aa2d64b8c8","name":"MPS432
Launchpad","serial":null},"conditions":{"presstime":{"gt":5.0}},"values
":{"presstime":{"value":6.15,"timestamp":"2016-02-
15T21:05:31.454Z","unit":"s or ″"}},"timeframe":0,"custom_data":null}
```

Setting up IFTTT to handle the M2X Trigger

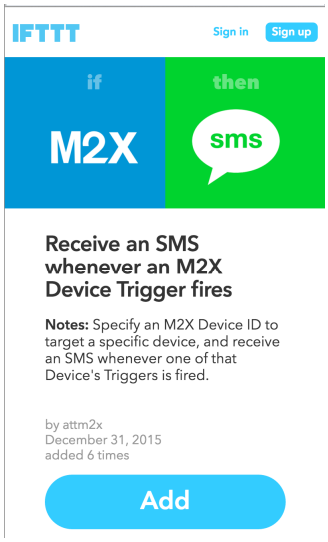
IFTTT (an abbreviation of "If This Then That") is a web-based service that allows users to create chains of simple conditional statements, called "recipes", which are triggered based on changes to other services. There are hundreds of channels available on IFTTT which can be used to trigger actions.

In this exercise, we will set IFTTT to send an SMS message to your mobile phone once a M2X Device Trigger fires.

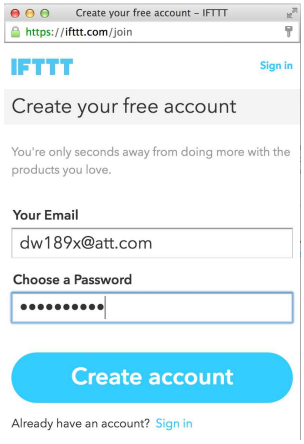
1. In a new browser tab or window, navigate to https://m2x.att.com/developer/tutorials/ifttt
2. Click the **Add** button for "Receive an SMS whenever an M2X Device Trigger fires" recipe.



3. Click on the **Add** button

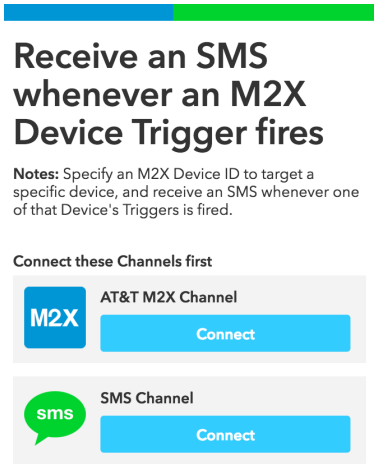


4. **Create an account**, or sign in if you already have one.



Integrating M2X and IFTTT

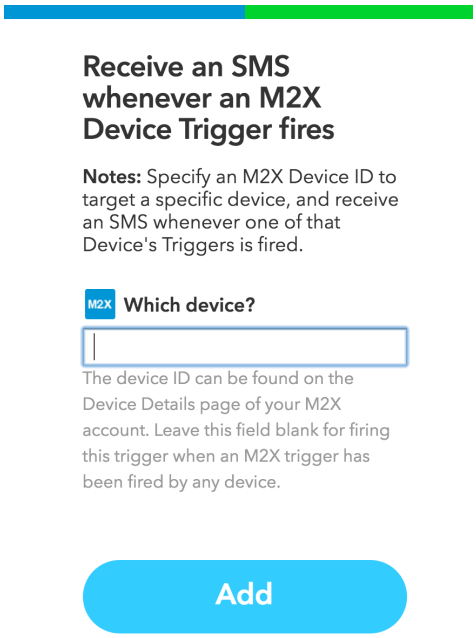
In order to complete the M2X and IFTTT integration, we need to connect to the M2X Channel and SMS Channel. Follow the Instructions on the IFTTT web pages to complete integration.



5. Click the **Connect** button under AT&T M2X Channel**,** IFTTT authenticates with your M2X account in the background. Click **Done** button once AT&T M2X has been connected to IFTTT.

6. Click the **Connect** button under SMS Channel, and follow the instructions to allow IFTTT to send a text message to your phone whenever an event is triggered in M2X. Click **Done** button once your phone number has been added to IFTTT.

7. Copy the **Device ID** for the LaunchPad device in M2X, and paste it into the **Which Device?** field in the IFTTT window, then click the **Add** button.



8. Click **Done** once the recipe is created.
    - Grab the Launchpad device, make sure the Energia code is loaded, then hold down the pushbutton for more than 5 seconds.
    - After a few seconds, you will receive a text message alerting you JSON object appear in the Debug window.

Next, you'll create a trigger by using the REST API.

Creating a Trigger through the API

Exercise Variant: REST Client (Postman Chrome App)

Sometimes a threshold is constant, and you will want to use the web interface to create a trigger, as you just did. However, sometimes a trigger will need to be dynamically generated or removed under certain circumstances. In this case, we can use the REST API to accomplish this. Follow these steps to add a new trigger for an even higher speed using the Postman Chrome App:

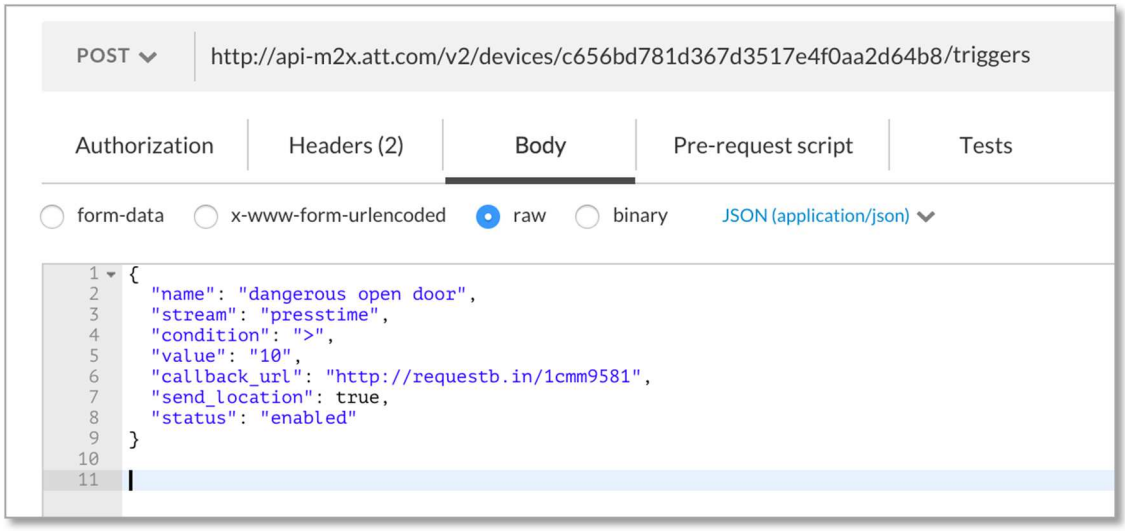
1. Open the Postman add-on for Chrome.
2. In the **Enter request URL here** box type **http://api-m2x.att.com/v2/devices/{device-ID}/triggers**
3. Replace the {device-ID} with the device ID. You can use the **Copy** button to copy the ID to the clipboard.
4. Set the method dropdown to 

5. Click on the **Body** tab and choose the **raw** button.

6. Use the same headers as before:
    a. Content-Type: application/json
    b. X-M2X-KEY: <your API key>
7. Add the following into the body box. This will create a new trigger with the name "dangerous open door" that will be triggered when the press time goes above 10 seconds. Note that the send location is set to true, so it will include location information in the POST request.

```
{
  "name": "dangerous open door",
  "stream": "presstime",
  "condition": ">",
  "value": "10",
  "callback_url": "<url>",
  "send_location": true,
  "status": "enabled"
}
```

8. Replace the <url> above with your RequestBin URL.
9. Your page should look like this:



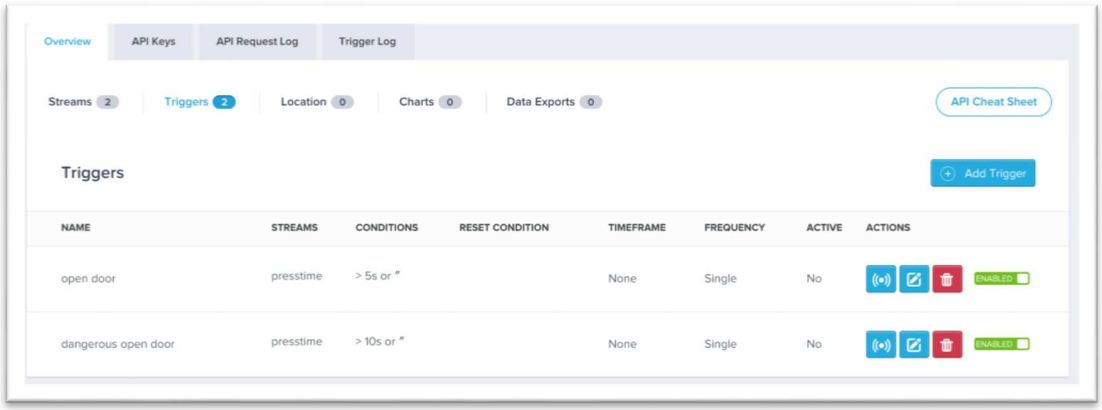
10. Click **Send**. You should see a response like this:

```
{
  "id": "AVLmxFnVnRrsYkigWPub",
  "name": "dangerous open door",
  "conditions": {
    "presstime": {
      "gt": 10
    }
  },
  "frequency": "single",
  "timeframe": 0,
  "callback_url": "http://requestb.in/165bf6v1",
```

```
  "url": "http://api-
m2x.att.com/v2/devices/c656bd781d3e4f0aa2d64b8c8/triggers/AVLmxFnVnRrsY
kigWPub",
  "status": "enabled",
  "activated": false,
  "send_location": true,
  "notify_on_reset": false,
  "custom_data": null,
  "payload_version": 2,
  "created": "2016-02-15T21:09:33.780Z",
  "updated": "2016-02-15T21:09:33.780Z"
}
```


Note that the **id** returned is the ID for the trigger, which you can then use to update the trigger by using the PUT method and remove it by using the DELETE method. See API Details for more information.

11. Return to the M2X website with your feed information. Refresh the page, then click the **Triggers** tab. You should now see your new trigger listed.



Exercise Variant: Command Line Tool (curl)

Note: The curl exercises are designed for Unix-based terminals, such as Unix, MacOS, and Cygwin. They use the backslash (\) in order to continue onto more than one line. To run them on a Windows command prompt, substitute each backslash for a caret (^), or remove the backslashes and put it all on one line.

Sometimes a threshold is constant, and you will want to use the web interface to create a trigger, as you just did. However, sometimes a trigger will need to be dynamically generated or removed under certain circumstances. In this case, we can use the REST API to accomplish this. Follow these steps to add a new trigger for an even higher speed using the curl:

1. Paste the following into a text editor. This will create a trigger for when the press time goes over 10. (You may have to put in the line breaks after pasting.) The `?pretty=true` parameter will return formatted JSON.

Note: If you have trouble with line breaks when copying-and-pasting from the PDF file, then copy-and-paste from the code that it's the **codeForM2X.txt** file in the **Lab1** folder.

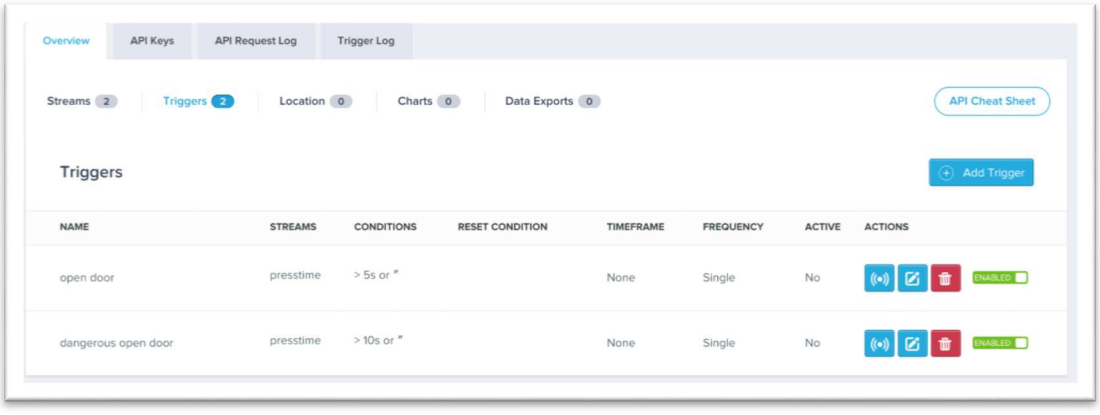
```
curl --request POST \
--header "Content-Type: application/json" \
--header "X-M2X-KEY: {API-key}" \
--data '{ "name": "dangerous open door", "stream": "presstime",
"condition": ">", "value": "10", "callback_url": "{url}", "status":
"enabled" }' \
http://api-m2x.att.com/v2/devices/{device-ID}/triggers?pretty=true
```

2. Replace {API-key} with your API key. (There should not be curly brackets around it.)
3. Replace {device-ID} with your device ID. (There should not be curly brackets around it.)
4. Replace {url} with the URL from ResponseBin. (There should not be curly brackets around it.)
5. Open up a terminal (command prompt in Windows) that has curl.
6. Paste the edited curl command into the terminal and hit Enter. You should see a return value like this:

```
{
  "id": "AVLmxFnVnRrsYkigWPub",
  "name": "dangerous open door",
  "conditions": {
    "presstime": {
      "gt": 10
    }
  },
  "frequency": "single",
  "timeframe": 0,
  "callback_url": "http://requestb.in/165bf6v1",
  "url": "http://api-
m2x.att.com/v2/devices/c656bd781d3e4f0aa2d64b8c8/triggers/AVLmxFnVnRrsY
kigWPub",
  "status": "enabled",
  "activated": false,
  "send_location": true,
  "notify_on_reset": false,
  "custom_data": null,
  "payload_version": 2,
  "created": "2016-02-15T21:09:33.780Z",
  "updated": "2016-02-15T21:09:33.780Z"
}
```


Note that the **id** returned is the ID for the trigger, which you can then use to update the trigger by using the PUT method and remove it by using the DELETE method. See API Details for more information.

7. Return to the M2X website with your feed information. Refresh the page, then click the **Triggers** tab. You should now see your new trigger listed.



Summary

You've learned how to add and use a trigger, both using the M2X website and using the M2X API.

Lab #4: My First project in Flow Designer

In this lab you will

- Learn to create your first project
- Write a simple flow
- Deploy your project to AT&T Platform as a Service prebuilt sandbox environment
- Test and Debug your project
- Make changes, redeploy and retest it
- Import, deploy and test a flow snippet
- Fork someone else's project, 'make it your own' and 'continue where they left off'

AT&T Flow Designer is a new cloud-based visual development tool aimed at speeding the development time to build new IoT applications. Flow Designer allows you to easily build data flows and policies for machine data using visual tools with JavaScript code snippets.

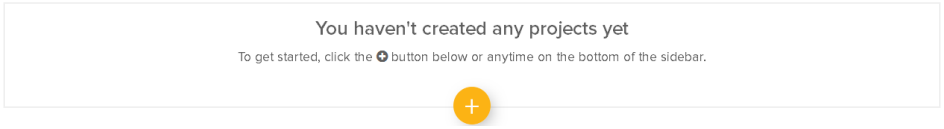
For more information, see AT&T Flow Designer.

Create your first project

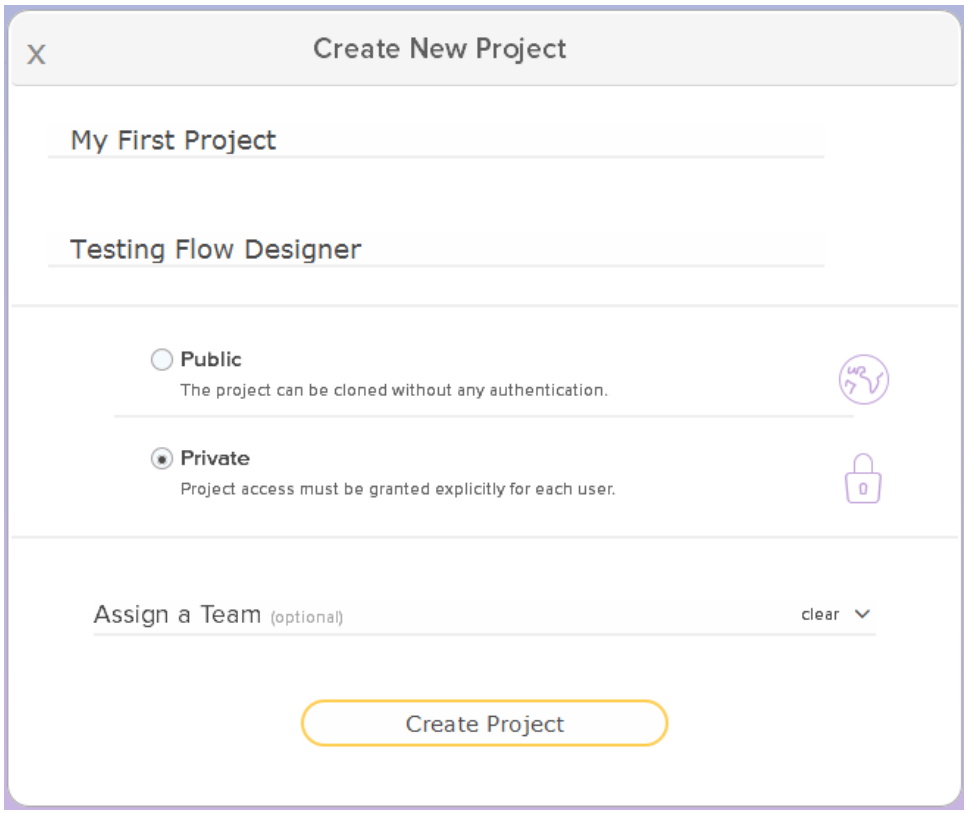
Projects contain the flows that together make your application. Let's start by creating your first project:

- Navigate to https://flow.att.com/
- Click **Login**. As long as you are signed in with M2X, you will be signed into Flow Designer. (If not, use your M2X username and password.) If this is the first time you've logged into Flow Designer, a new Flow Designer account will be created for you.

- Click the **+** button to create a new project.



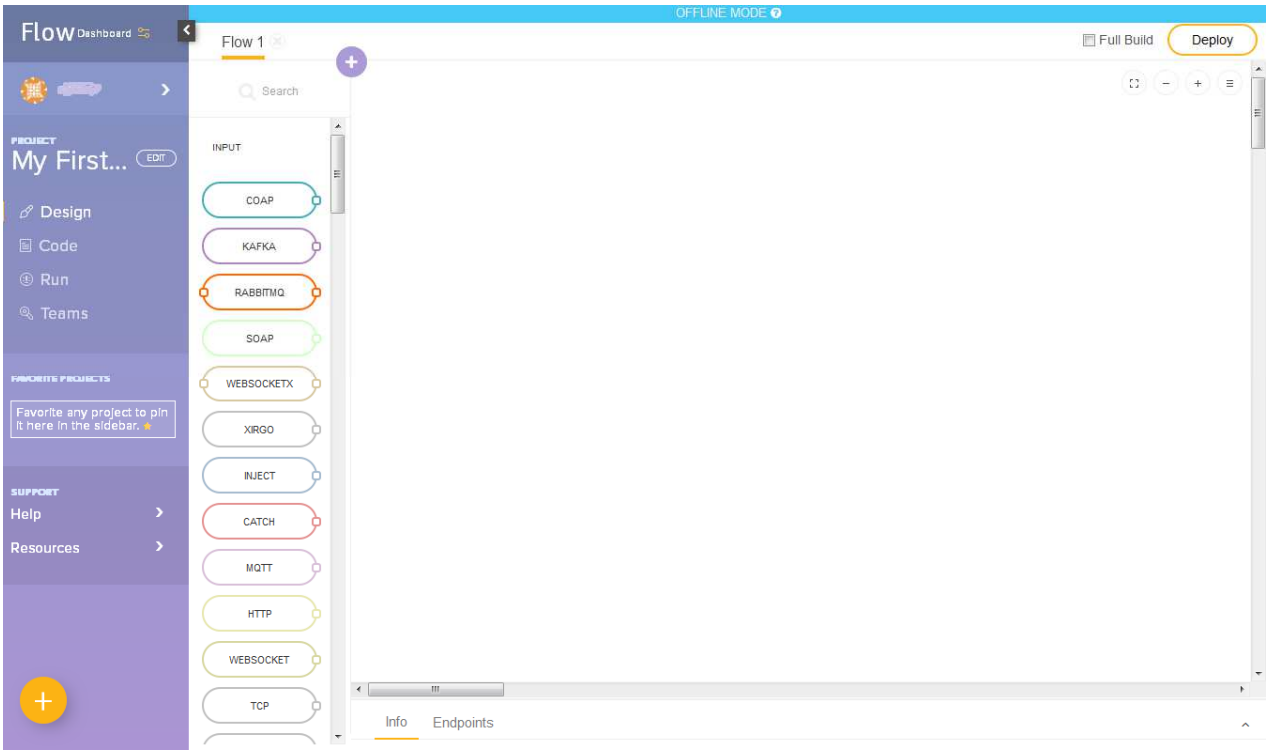
- Give your project a name, something like 'My first project'. You can also add a description if you like, 'Testing Flow Designer' for example.



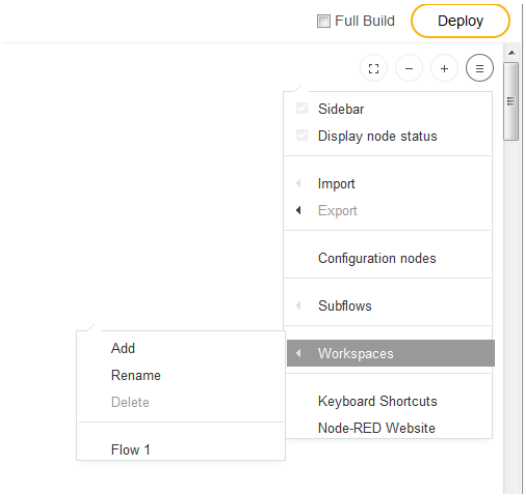
- Keep 'visibility' to private (you can always change it later). 'Internal' or 'Public' visibilities allow others to see your project, clone it, etc. and is great for collaborative work. For now, we'll keep this first project Private.
- Click 'Create' 
- The system will import the required template files and will set the environment for you. Once done, your new project is in ‘Design’ mode, ready for you to write your first flow.

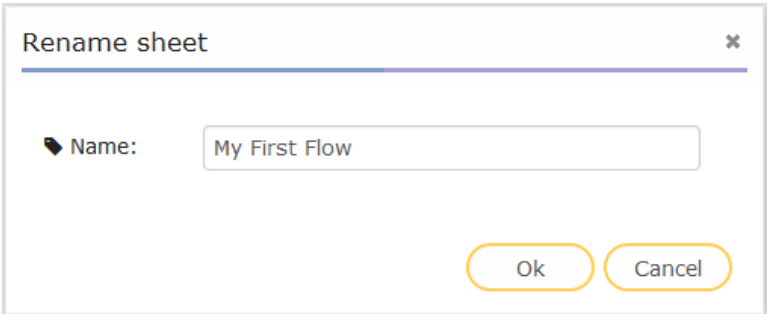
Create your first flow

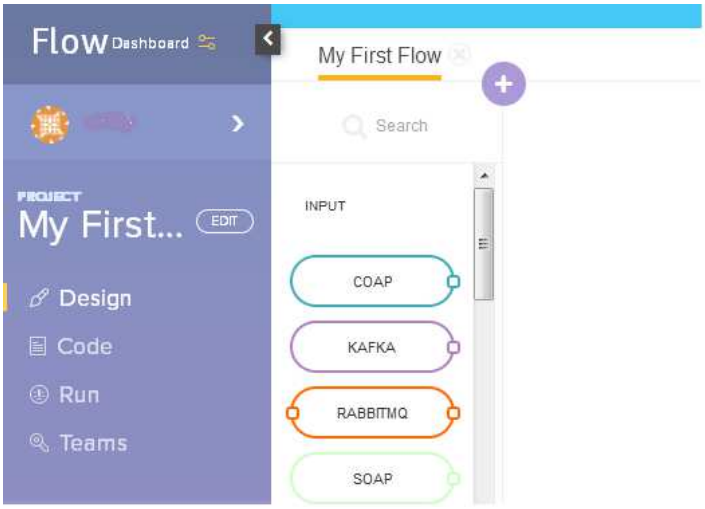
Your project was created with a default first flow called Flow 1'.



Rename that flow to 'My first flow' by double-clicking ‘Sheet 1’ or by clicking the Workspaces->Rename option from the dropdown menu on the top right of the canvas, renaming and hitting OK.



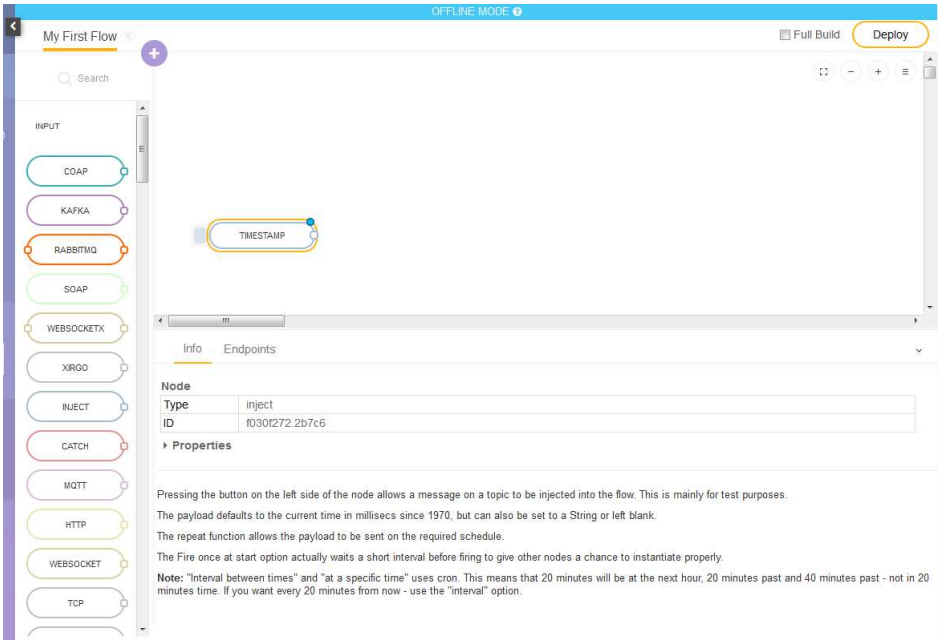




Now, let's write some code. Flow Designer uses 'nodes' as the main building blocks and wires between these nodes to create the flow of the application. We'll start with a few basic nodes.

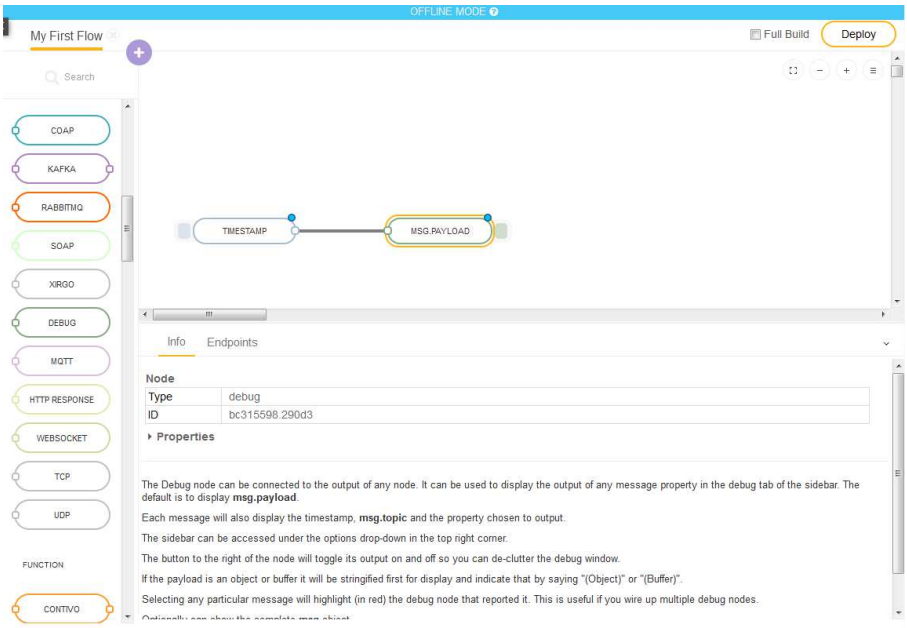
- Add an Inject node. The Inject node allows you to inject messages into a flow, either by clicking the button on the node, or setting a time interval between injects.
    - Drag an Inject node from the left palette onto the central workspace area.

    - Select the newly added Inject node. You will see information about its properties and a description of what it does on the 'Info' pane below. You may need to expand (drag up) the lower section to see the information pane shown below.



- Add a Debug node and place it to the right of the Inject node. The Debug node causes any message to be displayed in the Debug sidebar. By default, it just displays the payload of the message, but it is possible to display the entire message object.

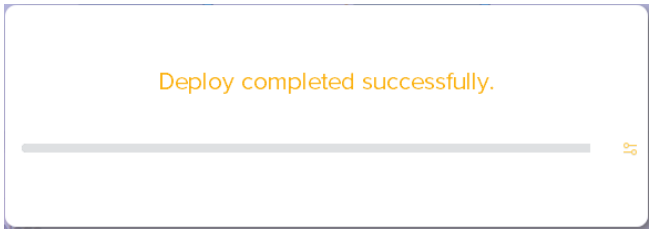
- Wire the two together. Connect the Inject and Debug nodes together by dragging between the output port of one to the input port of the other.



Test your flow

- Test your flow in the sandbox. At this point, the nodes only exist in the editor and must be deployed to the sandbox environment.
    - Click the ‘Deploy’ button. You'll see a progress report updated you on the status of the build.

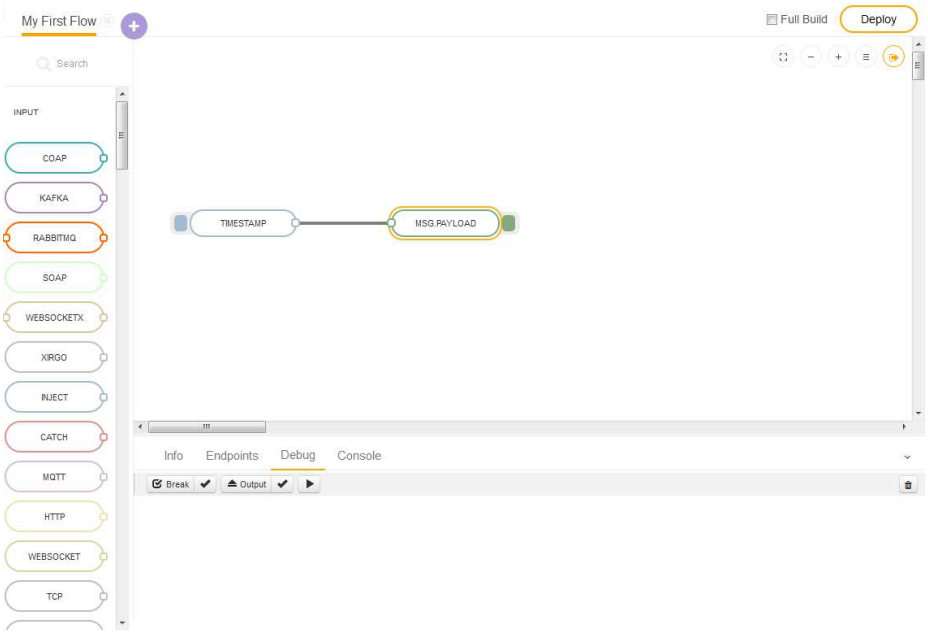
Once successfully completed, your flow has been built and deployed to the AT&T IoT Platform as a Service (PaaS), to a special default environment created for each project called 'sandbox'.



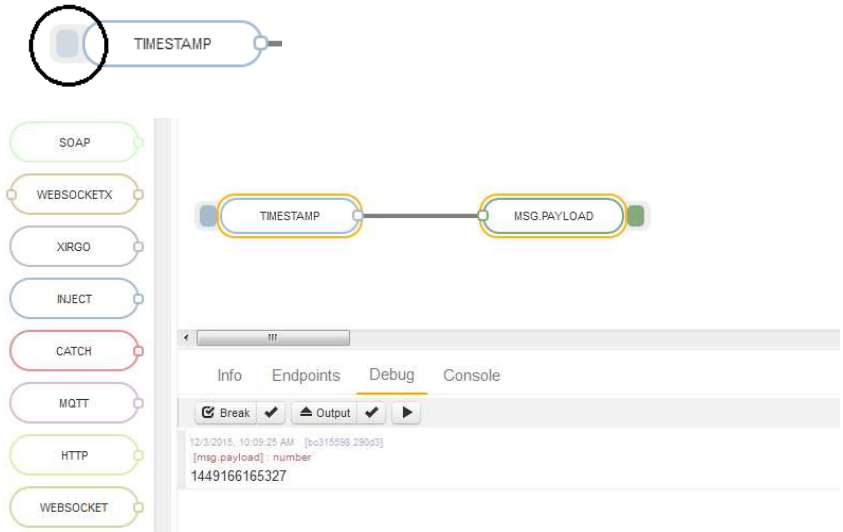
Once finished, you will need to click on the ‘Enter Online Mode’ Button in the upper right corner  to move to ‘Online Mode’. Online mode means you are actually running your code on a designated ‘Docker’ (lightweight portable code execution container, equivalent to virtual machine, but much more efficient), as if you were running it on your own machine. Offline mode in contrast, means as if you are

looking at an architectural drawing of your house (the flow), but unable to actually walk up the stairs to 'test' or debug their pitch. (Note that the online button icon indicates the state that you will be in once you've clicked it, not the state that you are currently in. So you will see a "connected" icon when you are offline and a "disconnected" icon when you are online.)

- Once you are in Online Mode, the top blue 'Offline mode' banner will disappear and two new tabs, named 'Debug' and 'Console' will be created next to 'Info' and 'Endpoints' tabs on the bottom panel.
- Expand the lower section if not already visible and select the 'Debug' tab on the bottom of the page.



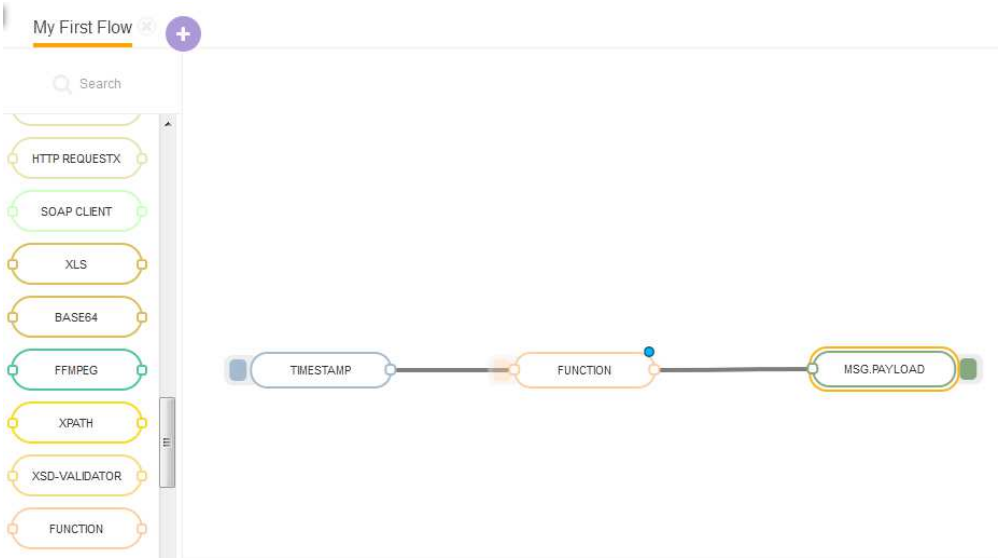
- Now click the small 'button' to the left of the Inject node. You should see numbers appear in the sidebar.



- By default, the Inject node uses the number of milliseconds since January 1st, 1970 as its payload. This is the number you now see in the debug pane. In the next step, we will do something more useful with that.

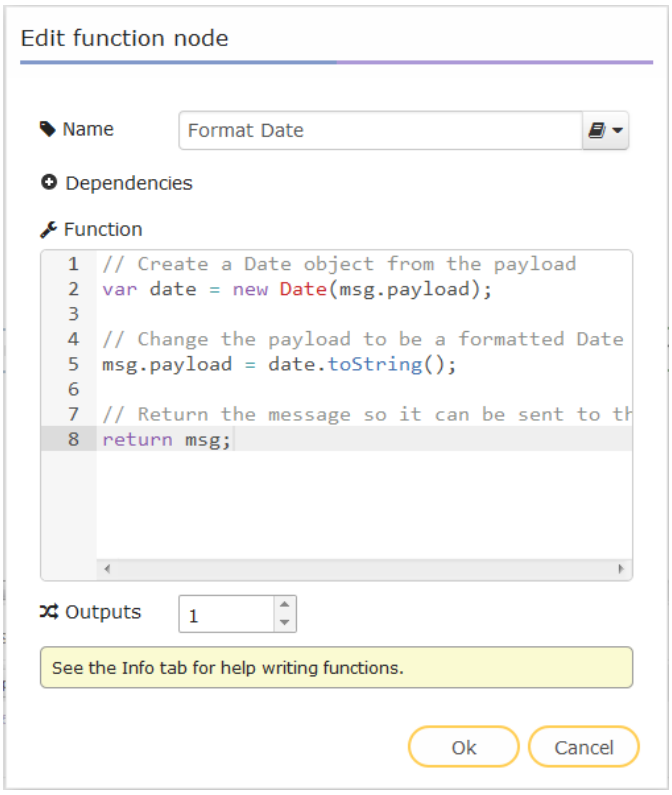
Modify and retest your flow

- Add a Function node. The Function node allows you to pass each message though a JavaScript function. Wire the Function node in between the Inject and Debug nodes. You'll need to delete the existing wire (select it and hit delete on the keyboard).

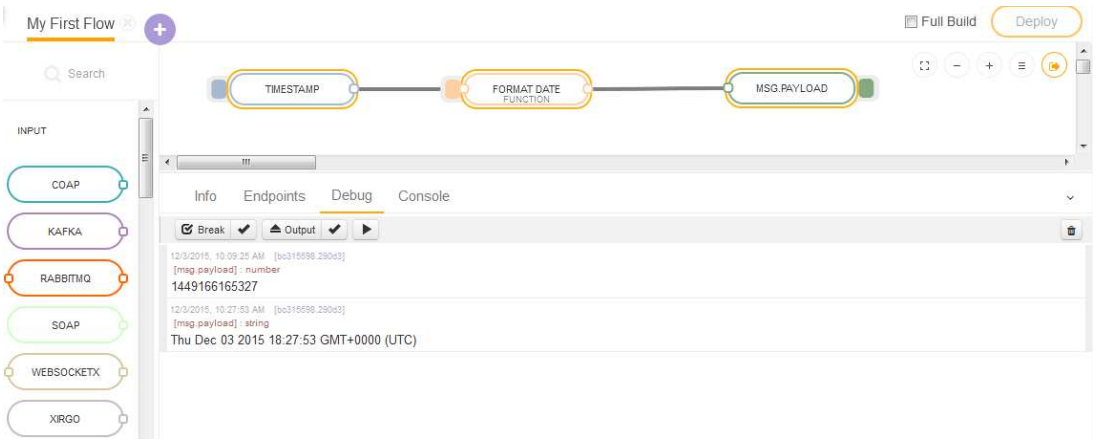


- Double-click on the Function node to bring up the edit dialog (Note: if you are using Chrome browser and nothing happens, try double-right-click instead). Name the node something like 'Format Date' and copy-paste the following code into the function text-area:

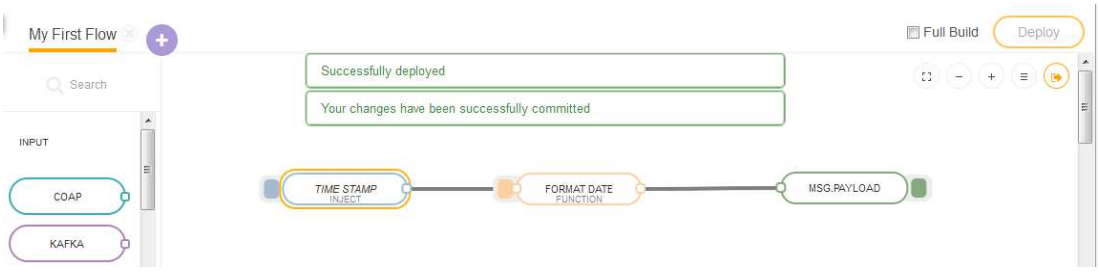
```
// Create a Date object from the payload
var date = new Date(msg.payload);

// Change the payload to be a formatted Date string
msg.payload = date.toString();

// Return the message so it can be sent to the next node
return msg;
```



- Deploy and Retest your flow. You should be still in online mode, as can be verified by the lack of the blue 'Offline mode' top bar



- Click the 'Deploy' to rebuild and deploy the changes into Flow Runtime sandbox environment.

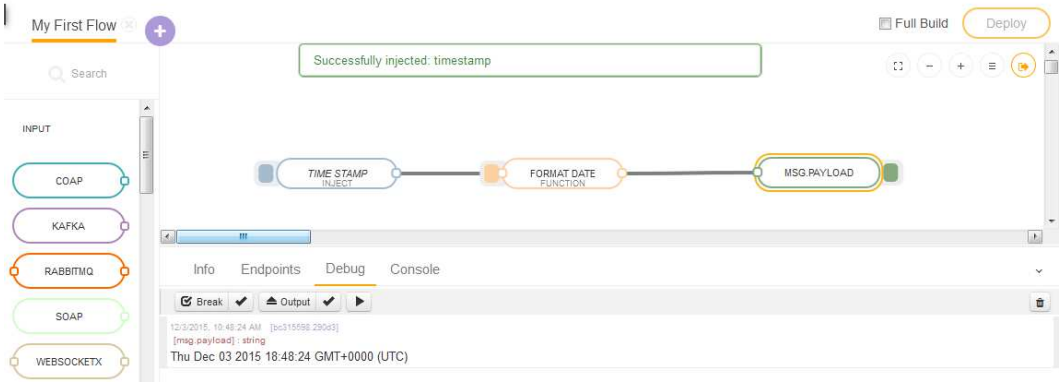


You will notice this build was much faster, with two small green toast messages popping from the top, as can be seen above.

The reason for it is that in Connected mode, like the name implies, you have actually 'jumped' into the Dockerized machine in AT&T Flow Runtime and everything here works 'locally', much closer to running a local NodeRED flow on your local machine, without the hassle and with all the versioning, multi tenancy and robustness power provided by AT&T's cloud.

Online mode is actually a good place to try and test small changed to your flow. The system is smart enough to decide which changes cannot be done in connected mode and requires a 'full build'. You, the developer, can always force a full build as well.

- Click on the Inject node again to see the changes you have made in the debug pane



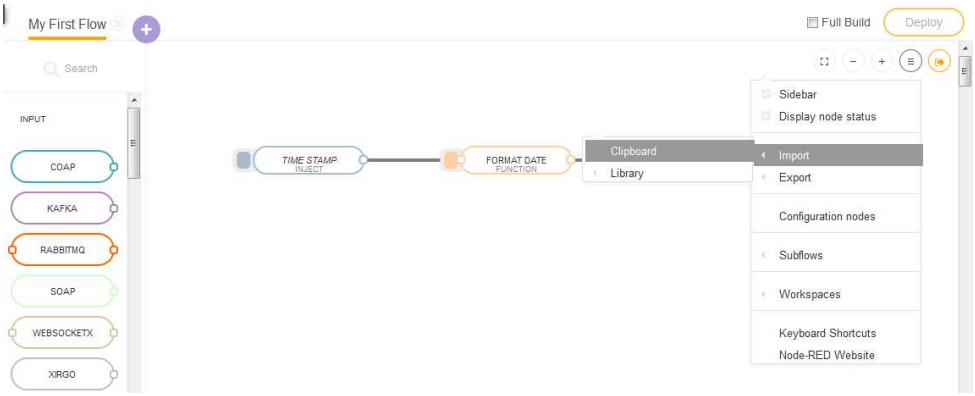
Import a flow

It is sometimes useful to reuse a piece of code you or someone else wrote. There are two ways of doing that in Flow Designer: Import and Fork a project. We will learn about forking shortly, but for now, let's focus on import.

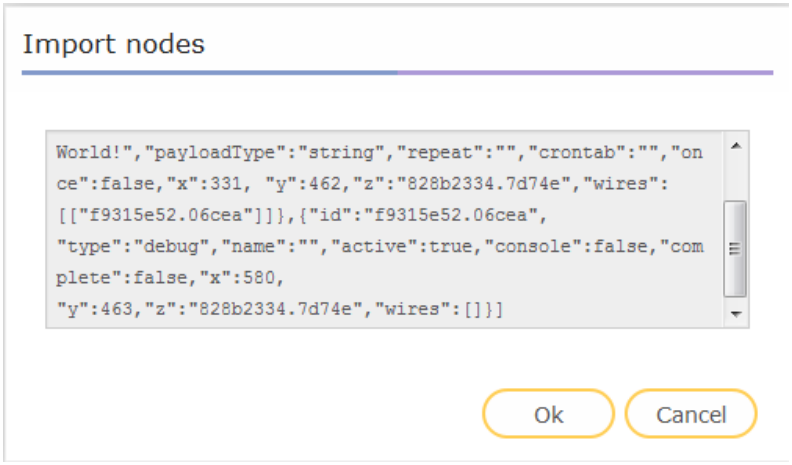
- The JSON annotation below represents a simple 'hello world!' flow, using only inject and debug nodes, much like your first flow. Select and copy it to your clipboard (Crtl+C for Windows, Cmd+C for Mac, etc.)

```
[{"id":"60b43c93.9f4bc4","type":"inject","name":"","topic":"","payload":"Hello
World!","payloadType":"string","repeat":"","crontab":"","once":false,"x":331,
"y":462,"z":"828b2334.7d74e","wires":[["f9315e52.06cea"]]},{"id":"f9315e52.06cea",
"type":"debug","name":"","active":true,"console":false,"complete":false,"x":580,
"y":463,"z":"828b2334.7d74e","wires":[]}]
```

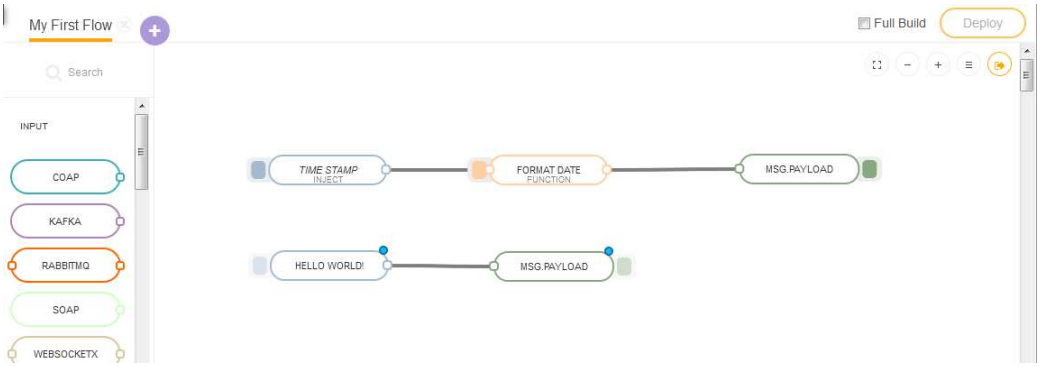
- In your Flow Designer project, click the menu -> Import... -> Clipboard...



  - Paste the JSON annotation from your clipboard (Ctrl+V or Cmd+V) and click OK

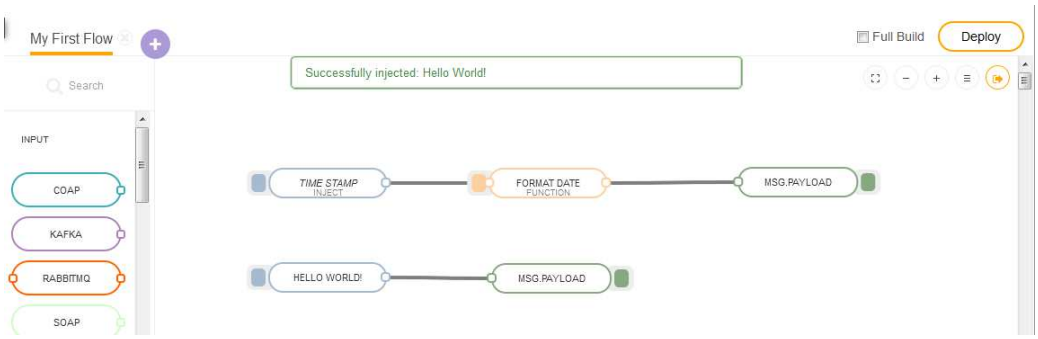


  - The flow will appear and will allow you to place it on your canvas. Make sure you place it on a clean spot, possibly below your existing flow.

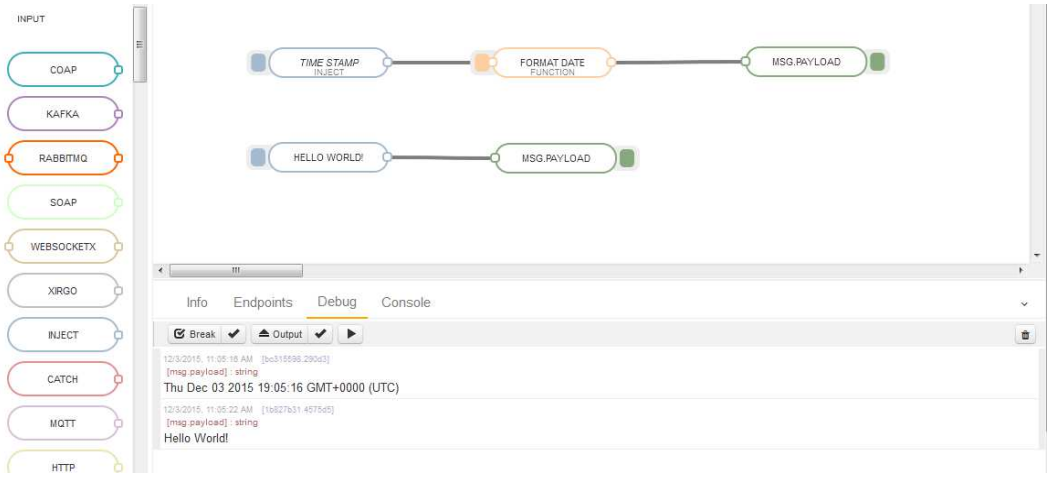


Notice the blue dot on the newly added flow. This is the systems way to tell you there are changes to existing nodes and/or new ones that were not yet deployed.

- Click Deploy button again to build and deploy the entire flow, including this new one. System will redeploy the code to the sandbox environment and changes will be committed to the underlying Git based version control system that is built into Flow Designer.



- Once done, test the new element by selecting the button next to the Hello World inject node. You should see 'Hello World!' in the debug window

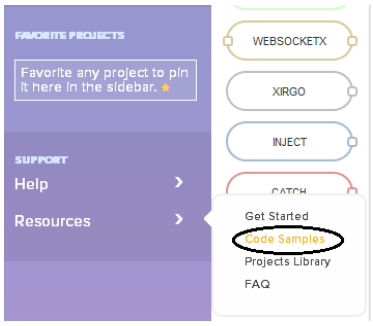


Feel free to double click the Hello World inject button to explore what is different it its configuration/code vs. your original inject node above it.

Fork public projects

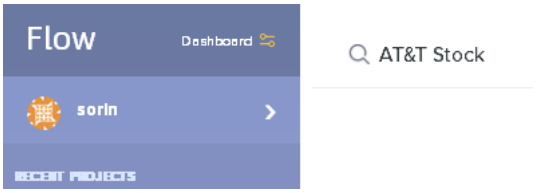
Now, let's try forking a project. Unlike import, forking takes the entire project, clone it and 'make it your own' so you can start working where the cloned project left off. Let's give it a try.

- Assuming you are still logged in, select the Resources->Code Samples option.

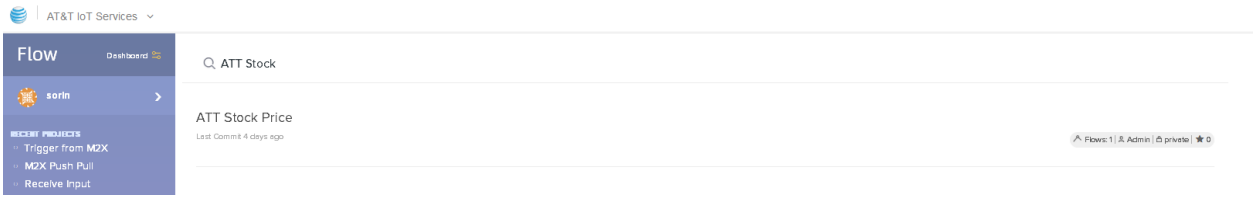


You could have chosen ‘Project Library’ as well from that menu. Code Samples includes the more official examples, while Project Library is the community section, with every project made visible by developers like you.

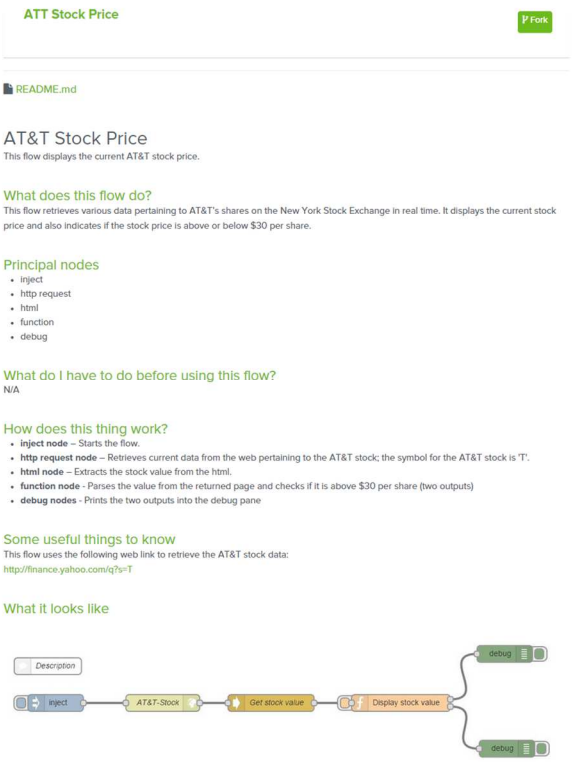
- Use the search-as-you-type box on the top right and type ‘ATT Stock’



You should be getting something like this:



- Click the ATT Stock Price project (owner should be ‘sample’). You should see the Readme.md project description with an option to ‘fork’ the project.
  Note: it is always a good idea to update the default readme.md file as this is how other will see your project.



- Click the 'Fork' button

- The system will clone the project to your space. Once finished, the canvas will show up.

- You can now explore the project nodes and code, test it, make changes, etc. This specific project checks the AT&T stock price on the Yahoo Finance web site and returns the price and 'true' if it is above $30/share
- Try to click the Deploy button to test the project functionality. You can then modify it to your needs and test again.

Lab #5: Responding to a trigger from M2X

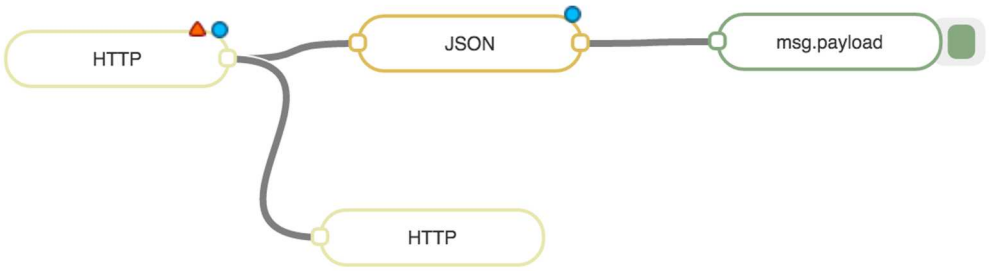
This lab assumes you have completed lab #4 prior to starting this one.

In this lab you will

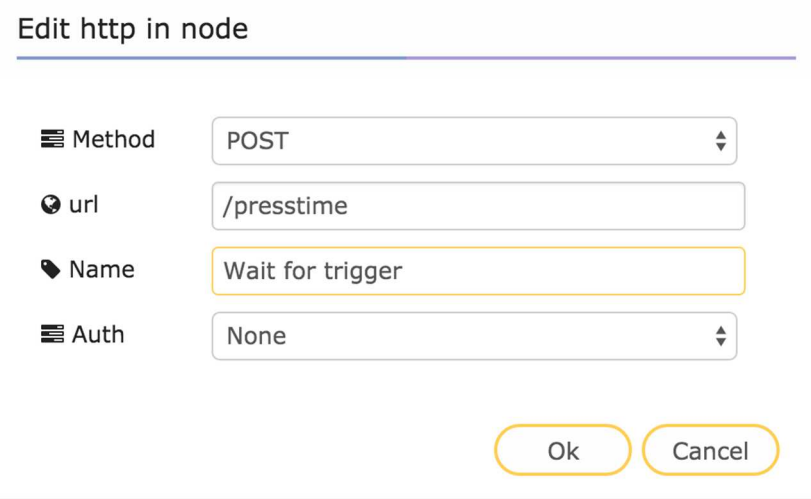
- Define a trigger in M2X (already accomplished in Lab #3
- Listen to that trigger in Flow Designer and
- Act on that trigger, once arrives

Create the project and flow

- Create a new project and name it something like 'Trigger from M2X'
- Drag in an HTTP node from the input section, and HTTP node from the output section, a JSON node, and a Debug nodes. Write them up as shown below, where the HTTP input node is at the left



- Double click the HTTP input node. Change the method to POST, the url to **/presstime** (this is the stream we created in Lab #2 and #3), give it a meaningful name like "Wait for trigger" and click OK.

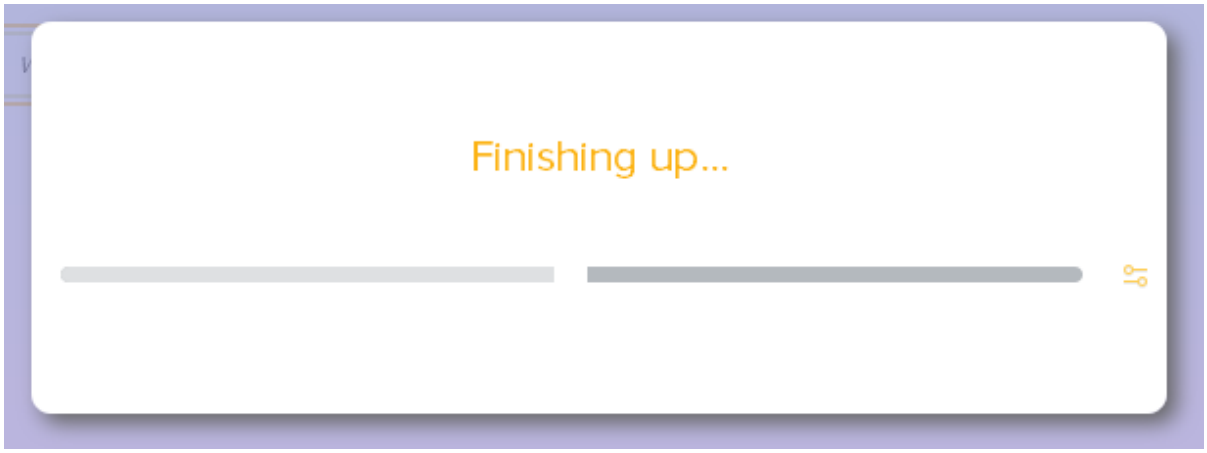


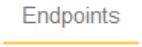
- There is nothing to do on the JSON node, other than give it a 'nicer' name. This node converts javascript object to JSON notation and vice versa (depending on the input).

Deploy the project and obtain the callback URL

- Click on the Deploy button 

in the upper right corner. Wait for the deployment to finish. It can take a few minutes.



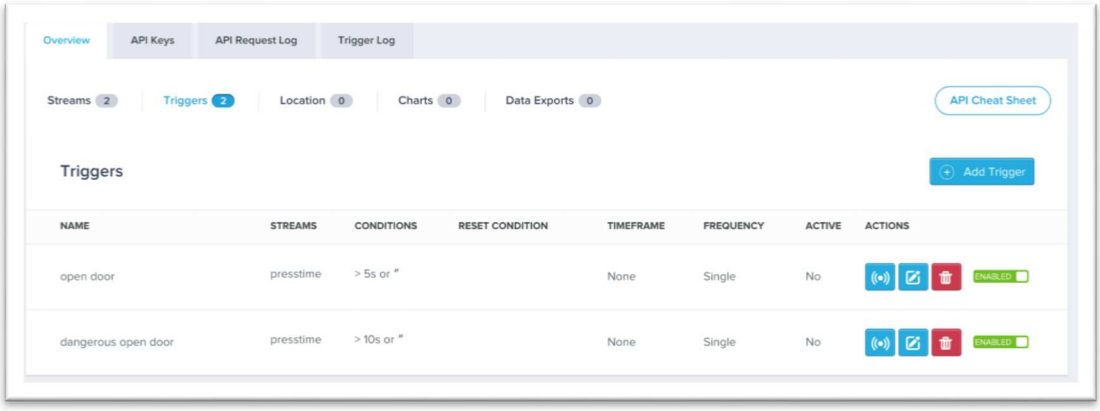
- Click the mode button in the upper right corner  to put the flow in online mode
- Click on the Endpoints tab 

at the bottom right under the Canvas of the flow you created.

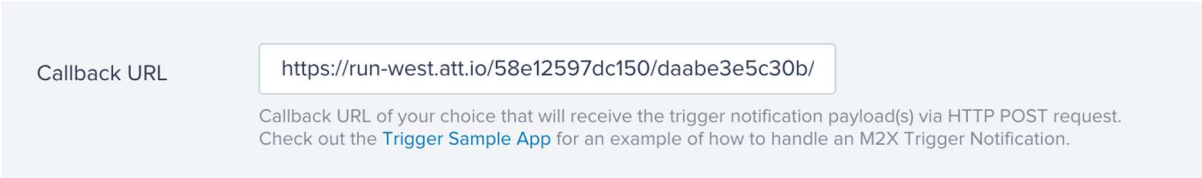
Look for the HTTPS endpoint field and click on the Copy botton on the right side.

Edit the M2X trigger

- Go back to your device page in M2X and click on the **Triggers** tab.



- Edit the “open door” trigger by clicking on the edit button
- Go into the **Callback URL** field and paste in the URL from the clipboard

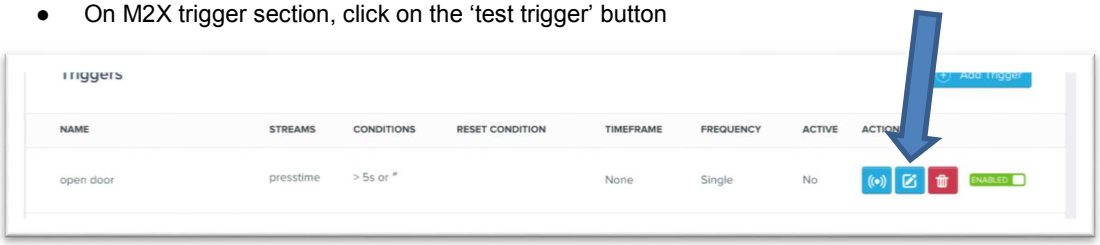


- Click ‘Save’

Test the trigger

Let’s first test the trigger through the M2X portal.

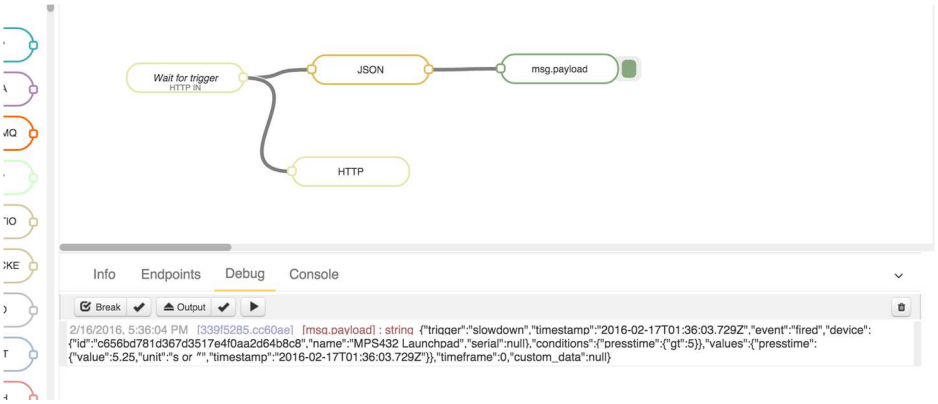
- On M2X trigger section, click on the ‘test trigger’ button



- Switch to Flow Designer and click on the Debug tab 

at the bottom of the Canvas.

The debug pane should show something like this (you may need to expand it):



As you can see, a lot of information is sent from M2X, allowing us to create a lot more logic around it, like a single 'listener' for many triggers, etc.

- Now try it with the actual device. Make sure the Energia code is loaded and then hold down the pushbutton for more than 5 seconds.
- After a few seconds, you will see another JSON object appear in the Debug window.

The flow we built is obviously not very useful, but FlowDesigner makes it easy to process the data and then make other events happen, such as sending someone an SMS so that they are instantly notified when an M2X trigger is fired.

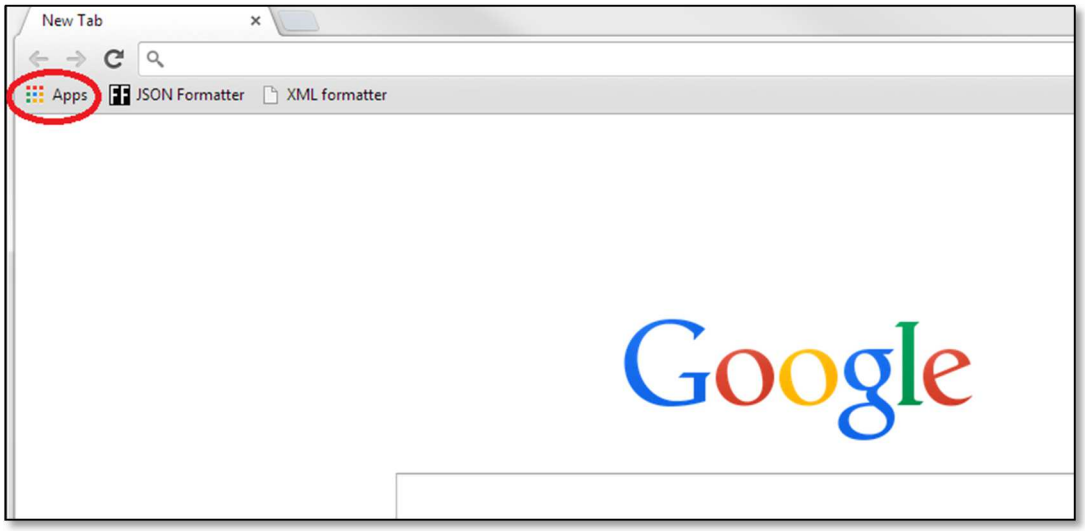
Appendix A: Installing Postman for Chrome

Pre-requisites: Google Chrome

Postman is a Chrome app that lets you make REST calls from a graphical user interface.

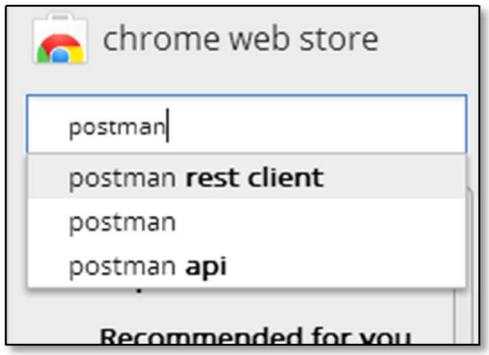
Use the following steps to install Postman:

1. Open a new tab in Chrome.
2. In the bookmark bar, select Apps



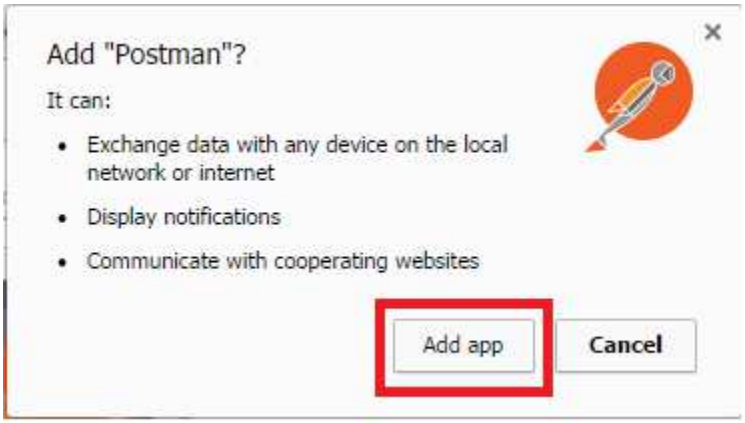
3. In the very bottom right-hand corner, select **Web Store**.

4. In the search box, type in **postman**, and then choose **Postman (offered by ww.getpostman.com)**.



5. Next to Postman (offered by www.getpostman.com), click on **+ ADD TO CHROME**

6. Click **Add app** when asked to add the app.



7. Postman is now installed. Whenever you click on the **Apps** icon in the bookmarks bar, you can select Postman to run the app.
   When you start Postman for the first time it will ask you to sign up, this is not needed, just select **Skip this, go straight to the app**

Appendix B: Installing curl for Windows

Pre-requisites: Windows 7 or later

curl (or cURL) is a powerful command-line tool for making HTTP calls. It is already installed on Mac computers.

For Windows, you can copy **curl.exe** from the **Lab1** folder of your USB drive. Copy it to somewhere on your hard drive that's easy to access from the command prompt.

Downloading curl

Use the following steps to install curl:

1. Open http://curl.haxx.se/dlwiz?type=bin in a browser.
2. Select your operating system in the dropdown box: either Windows /Win32 or Win 64. Click **Select**!
3. For Win 32, choose whether you will use curl in a Windows Command Prompt (**Generic**) or in a Cygwin terminal (**cygwin**). For Win 64, choose whether you will use curl in a Windows Command Prompt (**Generic**) or MinGW (**MinGW64)**. Click **Select!**
4. If required, choose your Windows operating system. **Finish.**
5. Click **Download** for the version which has SSL enabled.
6. Choose a version with support for SSL.
7. Open the downloaded zip file. Extract the files to an easy-to-find place, such as C:\Program Files.

Testing curl

1. Open up the Windows Command Prompt terminal. (From the Start menu, click Run, then type **cmd**.)
2. Set the path to include the directory where you put **curl.exe**. For example, if you put it in **C:\Program Files\curl**, then you would type the following command:

```
set path=%path%;"c:\Program Files\curl"
```

3. Type `curl`.

You should see the following message:

```
curl: try 'curl –help' or 'curl –message' fo r more information
```

This means that curl is installed and the path is correct.

4. Type:

```
curl --insecure https://api.att.com
```

You should see JSON returned:

```
{
```

```
"error": "invalid API request"
}
```

Troubleshooting

SSL certification error

If you see an SSL certification error, add a -k flag into your curl command.

https not supported error

If you get an error that says, “Protocol http not supported or disabled in libcurl”, then you need a different version of curl. Make sure you have downloaded one that says SSL enabled. If you have downloaded the Win64 version, try the Win32 version instead, even if you are on a 64 bit machine.

Appendix C: Software Clients

Several software clients are available to make it easier to integrate the AT&T M2X Data Service API into your apps. Rather than requiring you to make low-level HTTP calls, these SDKs and tools provide classes and methods that reduce the amount of code to call the underlying API requests. The following libraries are available from the M2X website at Client Libraries:

- .NET (Microsoft)
- Android (Google)
- Arduino
- BeagleBone
- C/C++
- Cypress PSOC
- Electric Imp
- iOS (Apple)
- Java
- JavaScript
- LaunchPad Energia (Texas Instruments)
- mbed
- Node.js
- PHP
- Python
- Raspberry Pi
- Ruby/mRuby
- Elixir
- Erlang
- Groovy