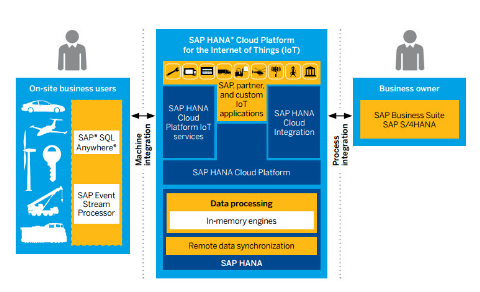
IoT1C02 – Simulate a Device with Node.js - HCP

|  |  |
| --- | --- |
| **Product**  HANA Cloud Platform IoT Services  **Level**  Undergraduate/Graduate  Beginner  **Focus**  HANA Cloud Platform  **Author** Ross Hightower | MOTIVATION  In this case you will simulate a device using Node.js to test the HCP IoT services  **PREREQUISITES**  IoT1C01 – Setup IoT Service - HCP |



# Simulate a Device with Node.js

Node.js is a platform for creating network applications using JavaScript. Node makes it easy to create applications that can create and consume RESTful services so it is ideal to interacting with HCP IoT services. In this case, you will create a node.js application that simulates a device. A later case uses node.js to interface between an actual device and IoT services.

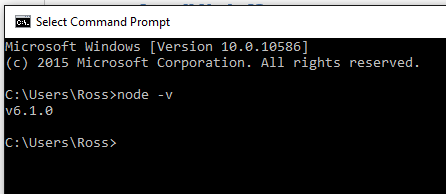
## Install Node.js

To install Node.js navigate to the URL below:

[http:// nodejs.org/](http://nodejs.org/)

Click **Install** and follow the installation instructions**.** To test the installation open a command prompt (on Windows) or a terminal (on a Mac) and enter the command node –v.

|  |
| --- |
| On Windows: Find the folder in Windows Explorer then hold the shift key while you right-click the folder. Select Open Command Window Here  On a Mac: Open System Preferences and select Keyboard > Shortcuts > Services. Find "New Terminal at Folder" in the settings and click the box. Now, when you're in Finder, just right-click a folder and you're shown the option to open Terminal. When you do, it'll start right in the folder you're in. |



## Create the Node.js App

Node applications are text files so, to create the application, you can use any text editor including Notepad. If you want a more advanced, free editor you can use and editor like the open source Atom (<https://atom.io/>).

|  |
| --- |
| This example was adapted from the SAP HANA Academy IoT series of videos. The code in this case was adapted from the code at the url below. Navigate to the nodejs folder then the climateHTTP.js file.  https://github.com/saphanaacademy/IoTServices |

Create a folder that will contain your application file and create a file called index.js. Copy the code shown below into the file and save it.

|  |
| --- |
| //http is the native node module for sending and receiving http messages  //This statement imports the module  var http = require('https');  //Update this code with your parameters  //This is described in the case  var host = 'iotmms<userid>trial.hanatrial.ondemand.com';  var device = '<deviceid>';  var oAuthToken = '<oauthtoken>';  var messageType = '<messagetypeid>';  //The base url for MMS services on HCP  var path = '/com.sap.iotservices.mms/v1/api/http/data/';  //Connection details for sending and receiving messages  var options = {  host: host,  port: 443,  path: path + device,  agent: false,  headers: {  'Authorization': 'Bearer ' + oAuthToken,  'Content-Type': 'application/json;charset=utf-8'  },  method: 'POST'  };  //Create an agent that implements the http module and pass in the options  options.agent = new http.Agent(options);  //The callback function is called when http messages arrive  //It will either write the message to the console or report any errors  callback = function(response) {  var body = '';  response.on('data', function (data) {  body += data;  console.log(body);  });  response.on('end', function () {  console.log("From MMS:", response.statusCode, body);  });  response.on('error', function(e) {  console.error(e);  });  }  //The request is the message sent from the application to the server  //This statement creates a request options and passes it the options and the callback  var req = http.request(options, callback);  req.on('error', function(e) {  console.error(e);  });  req.shouldKeepAlive = true;  //This is the message sent to the MMS service. The same format as the previous case  var jsonData = {  "mode": "sync",  "messageType": messageType,  "messages": [{  "temperature": 12.34,  "humidity": 56.78,  "timestamp":1413191650  }]  }  //Format the message as a JSON string  var strData = JSON.stringify(jsonData);  //Send the message and wait for a response  req.write(strData);  //Stop  req.end(); |

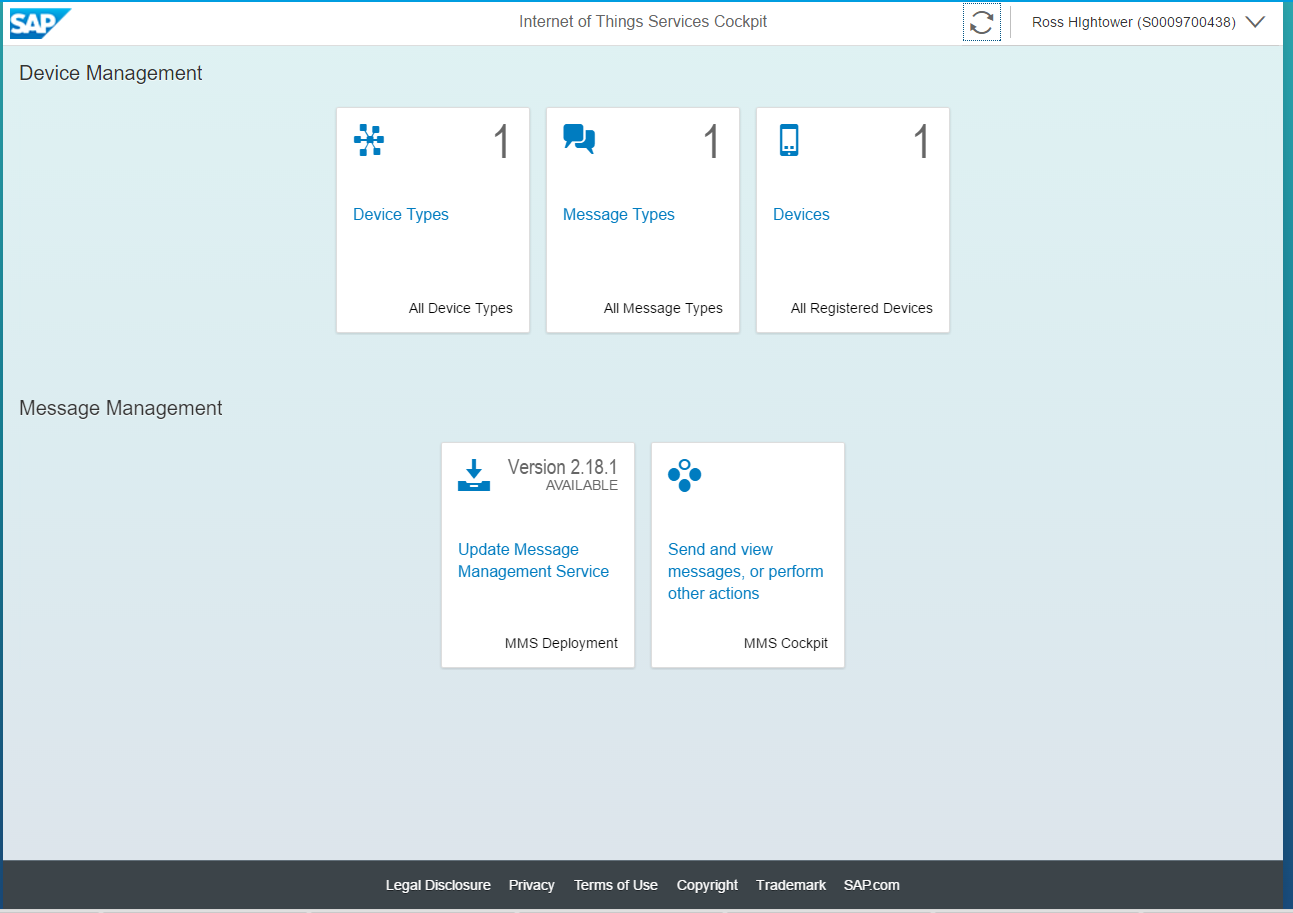
Listing

## Configure MMS

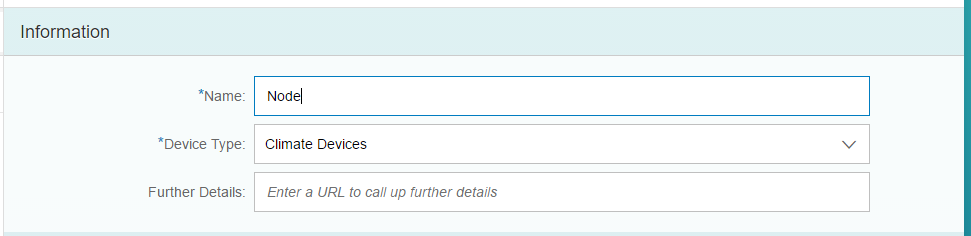
The next step is to update this portion of the code with your parameters.

|  |
| --- |
| //Update this code with your parameters  //This is described in the case  var host = 'iotmms<userid>trial.hanatrial.ondemand.com';  var device = '<deviceid>';  var oAuthToken = '<oauthtoken>';  var messageType = '<messagetypeid>'; |

Logon to HCP and open the IoT Services Cockpit. If you bookmarked it in the first case you can navigate directly to it.



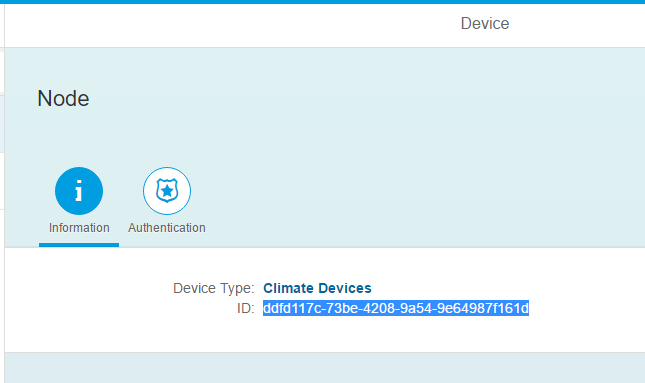
First, click on the Devices tile and create a new device called Node with Device Type of Climate Devices.



Copy the OAuth token into your code as shown below:

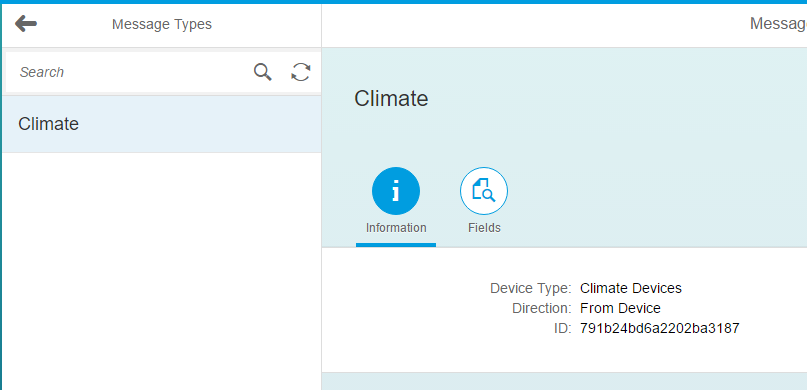
|  |
| --- |
| //Update this code with your parameters  //This is described in the case  var host = 'iotmms<userid>trial.hanatrial.ondemand.com';  var device = '<deviceid>';  var oAuthToken = '19db22ae2f75c9a6dd079d4be77cfa';  var messageType = '<messagetypeid>'; |

Next copy the device ID and paste it into the code:



|  |
| --- |
| //Update this code with your parameters  //This is described in the case  var host = 'iotmms<userid>trial.hanatrial.ondemand.com';  var device = 'ddfd117c-73be-4208-9a54-9e64987f161d';  var oAuthToken = '19db22ae2f75c9a6dd079d4be77cfa';  var messageType = '<messagetypeid>'; |

The application is designed to send the same message as in the previous case so you can use the same Message Type ID. Click on the All Message Types tile. Select the Climate message and copy the ID.



Paste it into the code as shown below.

|  |
| --- |
| //Update this code with your parameters  //This is described in the case  var host = 'iotmms<userid>trial.hanatrial.ondemand.com';  var device = 'ddfd117c-73be-4208-9a54-9e64987f161d';  var oAuthToken = '19db22ae2f75c9a6dd079d4be77cfa';  var messageType = '791b24bd6a2202ba3187'; |

Finally, replace <userid> with your user id.

## Send a Message

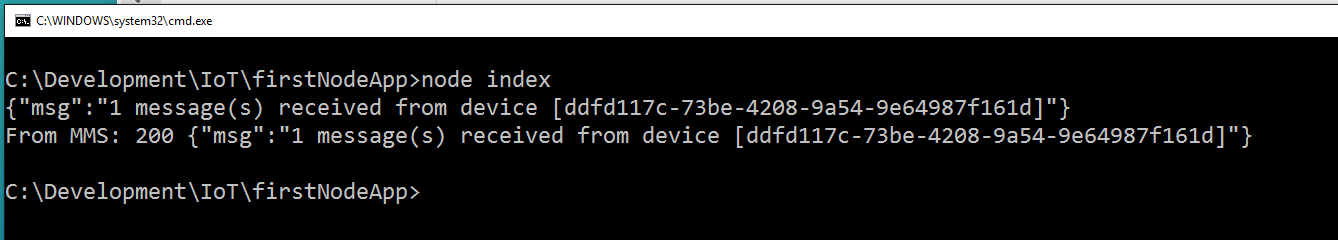
The message is the same format as in the previous case. You will find the message near the bottom of the application code.

|  |
| --- |
| var jsonData = {  "mode": "sync",  "messageType": messageType,  "messages": [{  "temperature": 12.34,  "humidity": 56.78,  "timestamp":1413191650  }]  } |

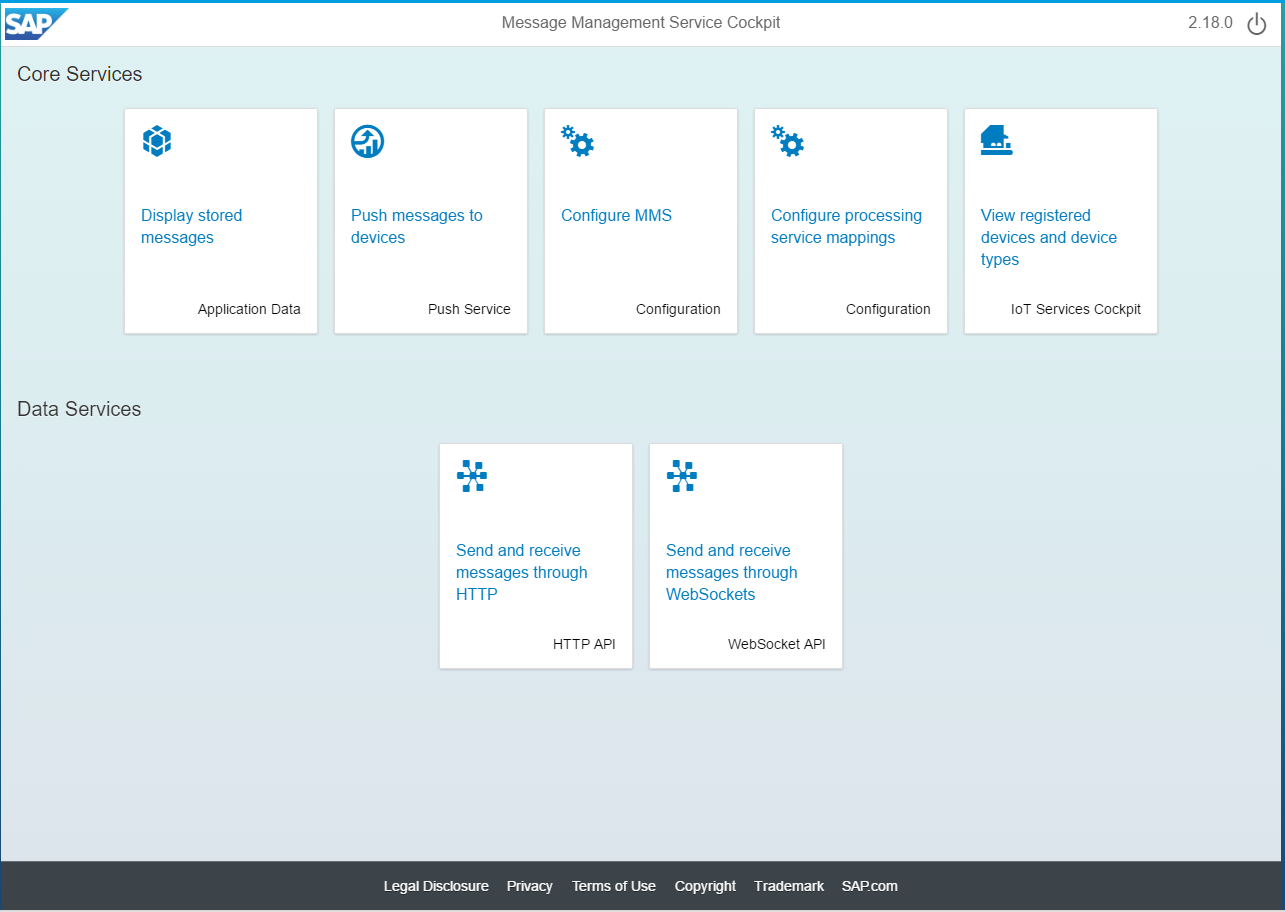
You can modify this message if you wish with different values.

To send the message, open a command prompt in the folder where you saved the application file. Enter the following command:

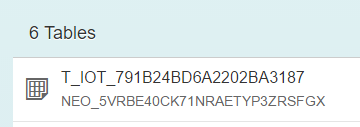
node index



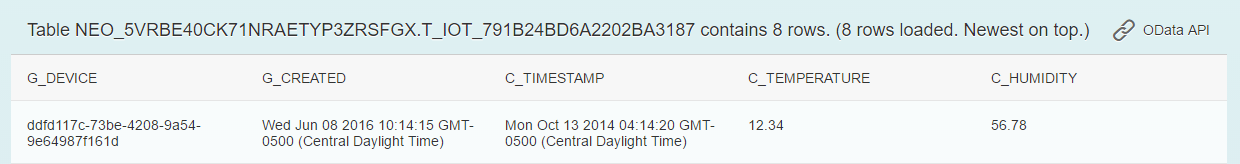
The response shows what would be expected for a message sent synchronously. Navigate to the MMS Cockpit and click on the Application Data tile.



Click on the table for the message type you created in the previous case.



The message should be in the top row.



## Simulate a Continuous Feed

In most realistic cases, a device would be sending updates to the service continuously. In this section we will simulate that situation. Create a new application file called deviceSim.js and paste the following code into it.

|  |
| --- |
| //http is the native node module for sending and receiving http messages  //This statement imports the module  var http = require('https');  //Update this code with your parameters  //This is described in the case  var host = 'iotmms<userid>trial.hanatrial.ondemand.com';  var device = '<deviceid>';  var oAuthToken = '<oauthtoken>';  var messageType = '<messagetypeid>';  //The base url for MMS services on HCP  var path = '/com.sap.iotservices.mms/v1/api/http/data/';  //Connection details for sending and receiving messages  var options = {  host: host,  port: 443,  path: path + device,  agent: false,  headers: {  'Authorization': 'Bearer ' + oAuthToken,  'Content-Type': 'application/json;charset=utf-8'  },  method: 'POST'  };  //Create an agent that implements the http module and pass in the options  options.agent = new http.Agent(options);  //The callback function is called when http messages arrive  //It will either write the message to the console or report any errors  callback = function(response) {  var body = '';  response.on('data', function (data) {  body += data;  console.log(body);  });  response.on('end', function () {  console.log("From MMS:", response.statusCode, body);  });  response.on('error', function(e) {  console.error(e);  });  }  var currentTemp = 70;  var currentHum = 30;var tempTrend = .8;  var humTrend = 2;  var sequence = 0;  //Function to send a message  sendMessage = function(){  //Compute a timestamp in seconds  var timestamp = Math.floor(Date.now() / 1000);  //Compute temperatue and humidity values  currentTemp += tempTrend + Math.floor(Math.random()\*2-2)/10;  currentHum += humTrend + Math.floor(Math.random()\*2-2)/10;  var jsonData = {  "mode": "async",  "sequence": sequence++,  "messageType": messageType,  "messages": [{  "temperature": currentTemp,  "humidity": currentHum,  "timestamp": timestamp  }]  }  var strData = JSON.stringify(jsonData);  //The request is the message sent from the application to the server  //This statement creates a request options and passes it the options and the callback  var req = http.request(options, callback);  req.on('error', function(e) {  console.error(e);  });  req.shouldKeepAlive = true;  req.write(strData);  }  sendMessage();  //Send a message every 1 minute  setInterval(sendMessage, 60000); |

Listing

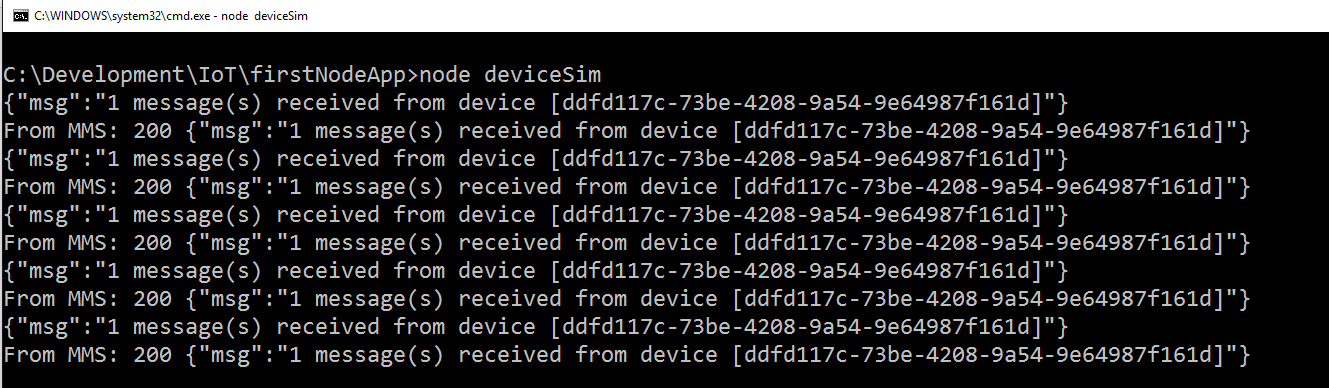
As before, update the highlighted portions of the code with the appropriate values.

The setInterval() function at the bottom of the code will call the sendMessage function once a minute. The interval is in milliseconds.

In the sendMessage() function, a time interval, current temperature and current humidity are calculated and the message is sent. This will provide some data for the next case in which we create an application to view the data.

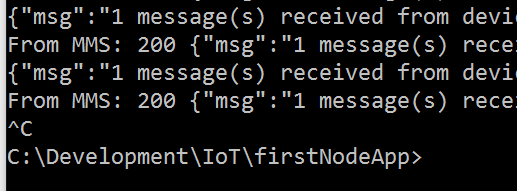
Execute the application using the following command at the command prompt:

node deviceSim

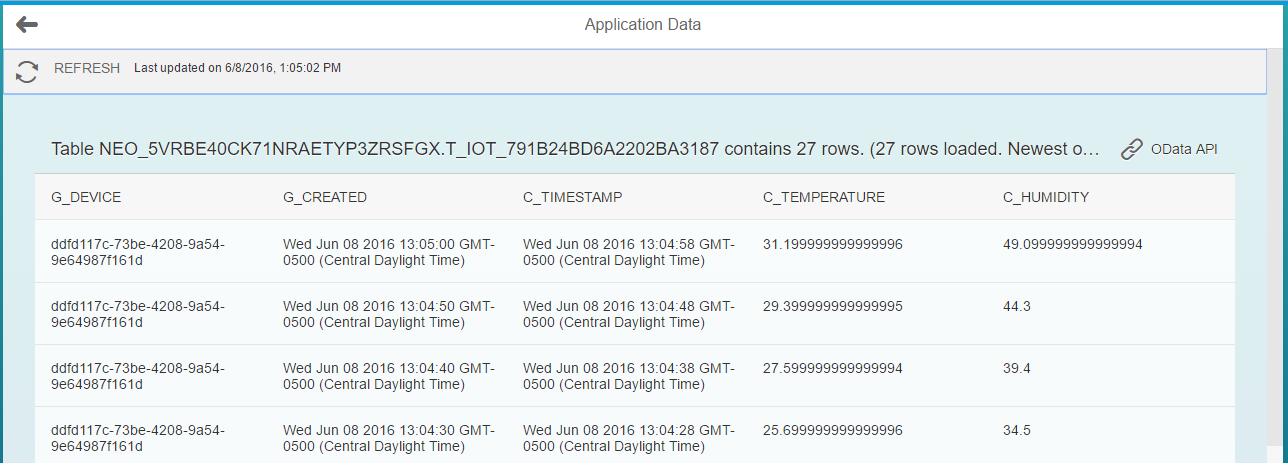


The program sends a message to MMS every minute. Let the program run so that you have 10 – 20 messages.

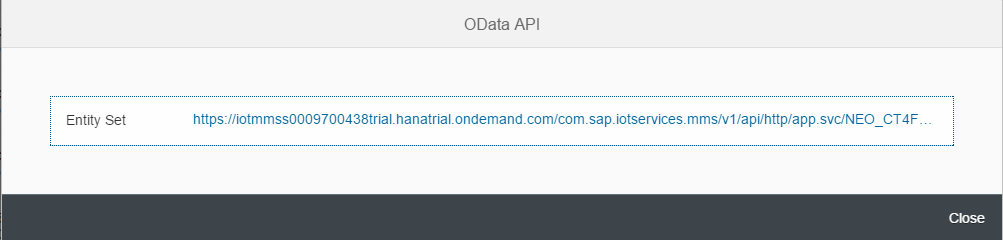
To end the application use control-C (on Windows) or command-C (on a Mac).



Clicking refresh in MMS shows the messages in the table.



Click the  icon and then click the URI.



This retrieves the data in OData XML format.

