# Heating Monitor

## Block Diagram

STREAM ANALYTICS [HeatingMon/HeatingMonIoT]

IOT HUB [HeatingMonIoT] or EVENT HUB [heatingmon]

SQL DB [HeatingMonDb]

Web client to monitor (and control) measurements [HeatMonitor.HTML]

Internet

HTTP/REST

Sensors

WiFi/Internet

Measurement system, reads sensor data (temperature, humidity, etc) and send them to the cloud

HTTP

AZURE

HTTP

Web service [HeatMonServer.js]

Contents

[Heating Monitor 1](#_Toc453271359)

[Block Diagram 1](#_Toc453271360)

[Change history 3](#_Toc453271361)

[Setting up Azure Services 4](#_Toc453271362)

[Azure Active Directory (AD) 4](#_Toc453271363)

[Service Bus / Event Hub 4](#_Toc453271364)

[IoT Hub 5](#_Toc453271365)

[SQL Database (and Server) 6](#_Toc453271366)

[Stream Analytics 7](#_Toc453271367)

[Web Application (and Webjob) 8](#_Toc453271368)

[SQL Database (HeatingMonDB) 8](#_Toc453271369)

[CUSTOMER\_DATA 8](#_Toc453271370)

[DEVICE\_DETAILS 10](#_Toc453271371)

[MEASUREMENTS 11](#_Toc453271372)

[Related Software 14](#_Toc453271373)

[Device Software 14](#_Toc453271374)

[Web Server Software 14](#_Toc453271375)

[Client Software 15](#_Toc453271376)

[Using IoT Hub 18](#_Toc453271377)

[Device Explorer 18](#_Toc453271378)

[Posting messages to Azure IoT hub 21](#_Toc453271379)

[Authentication 26](#_Toc453271380)

[Real Time Handling 27](#_Toc453271381)

## Change history

9.6.2016 RTC functionality added

24.5.2016 Some pictures and change history added

18.5.2016 Original version

## Setting up Azure Services

All resources will be set up on same data center, North-Europe region.

Resources have been named using project name Heating Monitor (HeatingMon) as base for each system.

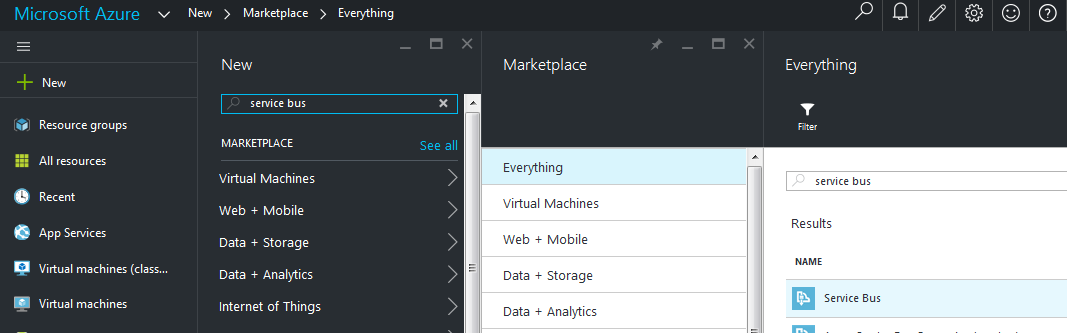
### Azure Active Directory (AD)

Not clear if this should be used and how.

### Service Bus / Event Hub

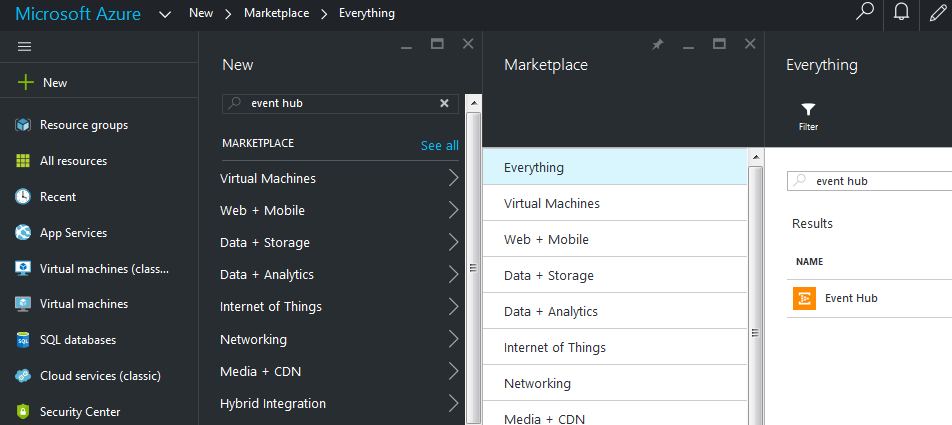
Either this (event hub) or IoT hub described in next chapter will be used.

Creating service bus



Service bus namespace is HeatingMon. When creating bus, the bus type (MESSAGING) and the messaging tier must be selected. In Service bus CONFIGURE menu, PRIMARY KEY is the key when sending data to system. The same key must be available for the sender devices. This means that key value must always stay same in this system. Otherwise there must be a method to send new key to measuring devices.

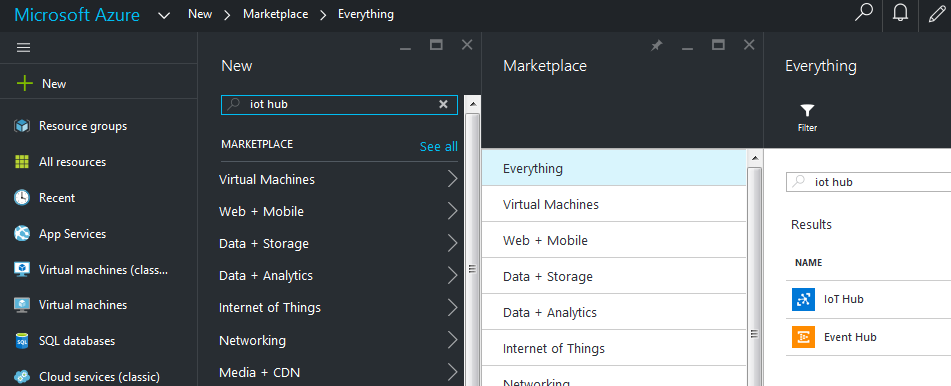
Creating Event Hub



Event hub name for the project is heatingmon. The only meaningful settings will be found in CONFIGURE menu. There is MESSAGE RETENTION which tells system how many days messages will be retained in the hub.

### IoT Hub

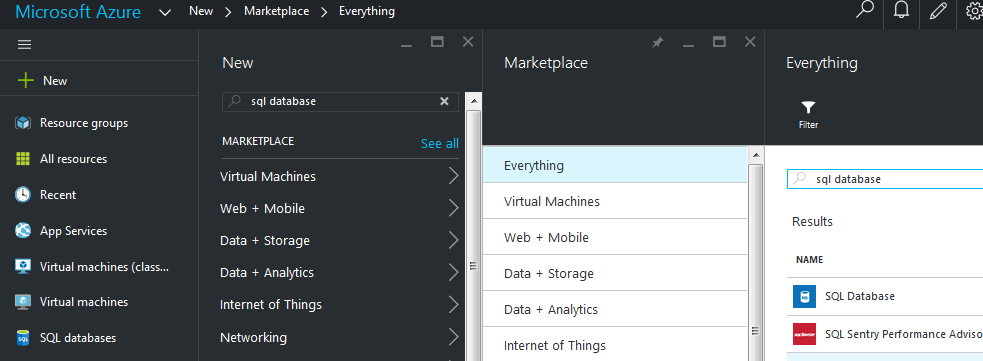
Crating IoT hub:



The name of our hub is HeatingMonIoT. There is not much to configure, but in “Shared access policies” menu, you can control access. You get the keys by clicking policy (device) name.

### SQL Database (and Server)

Creating SQL database:



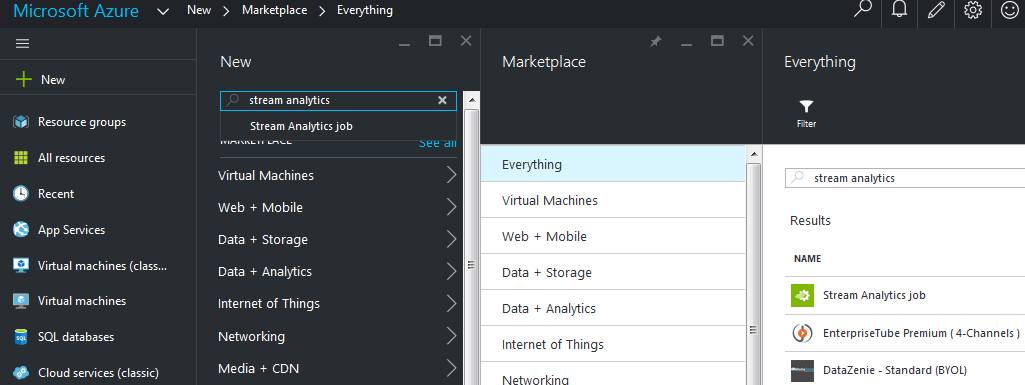
In QUICK CREATE menu you need to give name to the database (HeatingMonDB) and choose the server for the database. You may select “New SQL database server” or if you have any, use some previously created servers. If you select creating new, you will be asked admin login name and password for it. The automatically created server will get a funny name, for example fyn7a28mhq in our project. If you create server yourself, you may select also the name. However, then you need to take more care of server later yourself (Iaas service).

In server CONFIGURATION you need to set allowed IP addresses to manage the database. You should also allow WINDOWS AZURE SERVICES.

There is nothing in database settings that need to be set in menus. From database view, by selecting MANAGE, you can open Microsoft Azure Database management software that you can use managing database, add tables and columns. Another choice to manage database is Microsoft Visual Studio.

### Stream Analytics

Creating stream analytics job:



Stream Analytics is needed to move data from hub to safe. In this case we will move data to SQL database.

When this resource was created, the name was given HeatingMon, for IoT hub the name is HeatingMonIoT. As storage account was selected previously created SQL database. First thing to configure in stream handling is to select INPUTS tab and add new Data Stream job. In next page either Event Hub or IoT Hub should be selected depending which is in use. Following thing is to give some settings, especially the hub selection is important. Serialization format must be JSON and encoding UTF8.

Next task is the OUTPUTS menu where you need to add a job to output data to SQL database. Some settings again for output, where you need to give the database server name, login info and also the table where data will be put.

Last step is to define the query which takes care of moving data from input to output. That can be done in QUERY tab. There we have a query box where in our project following SQL-like commands need to be given:

SELECT

cust\_id AS [cust\_id],

device\_id AS [device\_id],

meas\_time AS [meas\_time],

System.Timestamp AS [save\_time],

triggering\_event AS [triggering\_event],

sensor0 AS [sensor0],

sensor1 AS [sensor1],

sensor2 AS [sensor2],

sensor3 AS [sensor3],

sensor4 AS [sensor4],

sensor5 AS [sensor5],

sensor6 AS [sensor6],

sensor7 AS [sensor7]

INTO

[heatingMonOutput]

FROM

[heatingMonInput]

Here is also some query testing tools with which you can do some trials for your query by inputting json-messages from text file.

### Web Application (and Webjob)

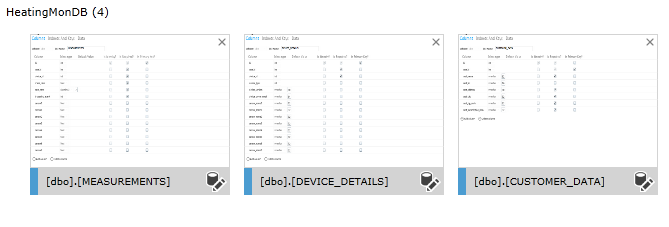
The purpose of web application is to handle all database reading and writing through controlled service so that not anybody can access the database directly.

When creating a Web app, you need to give database name and login info. Setting up the service you may start from CONFIGURE. Most of the settings are fine by default but, if possible you should turn ALWAYS ON option to ON state. However, this is not always possible depending on subscription and service plan. Other thing here is to add our client handler (HeatMonServer.js) to “default documents” list to tell system which while to start running automatically. The location on the list may affect to selection.

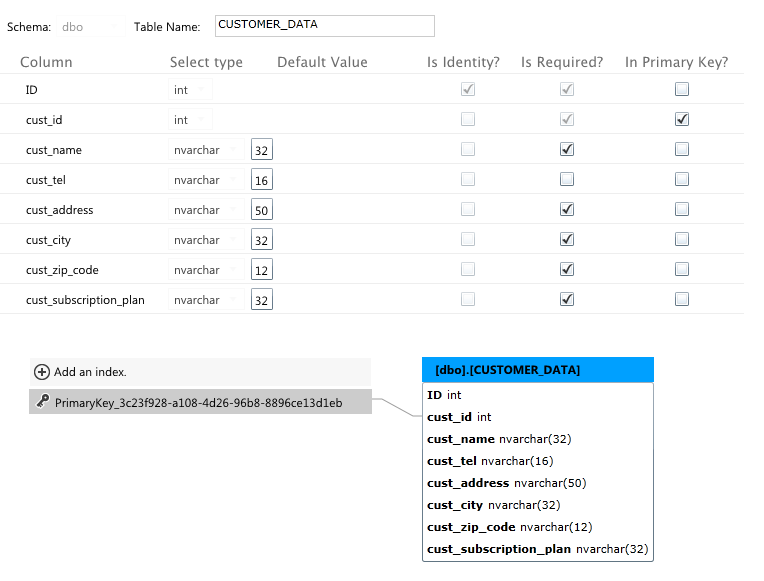
Configuration for rest services in web.config file must be set up to tell IIS (Internet Information Server) to which process REST-messages should be transferred.

## SQL Database (HeatingMonDB)

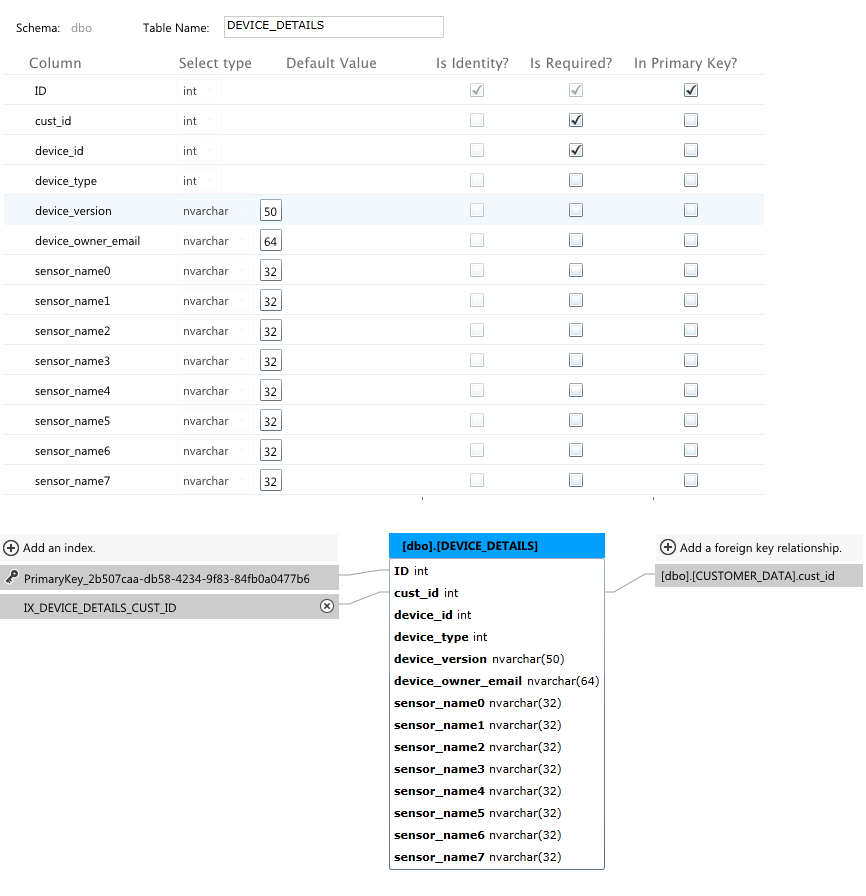
Currently there is three tables:



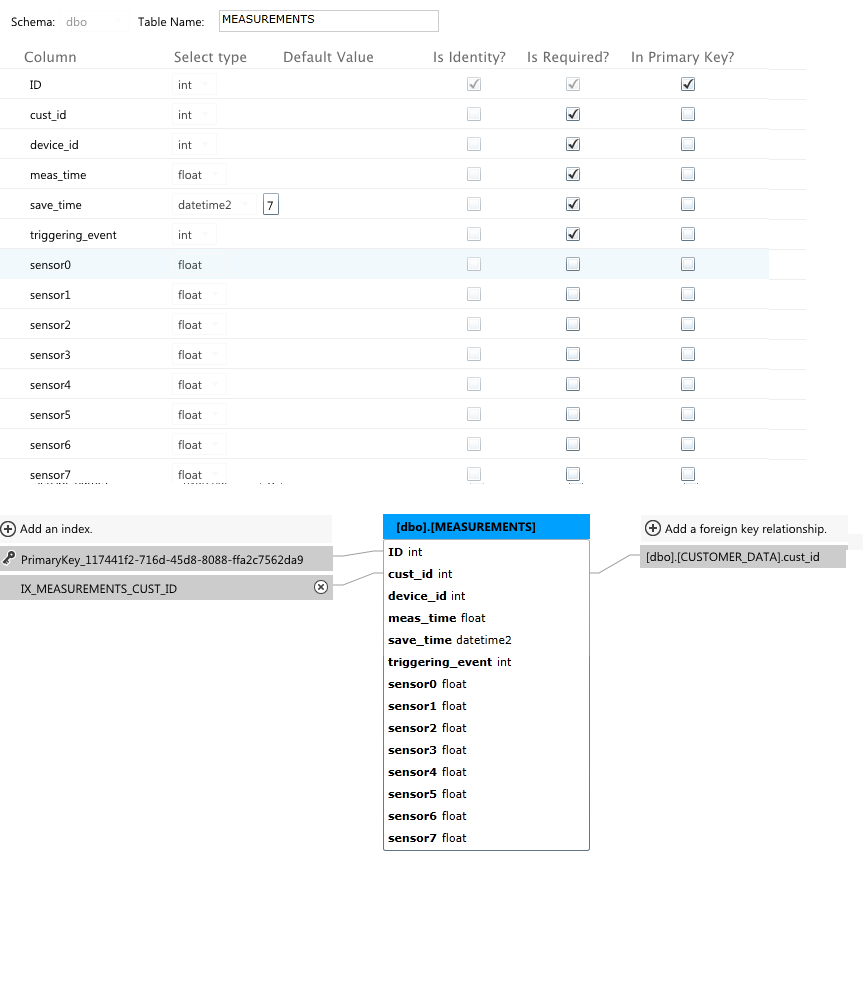
### CUSTOMER\_DATA



### DEVICE\_DETAILS



### MEASUREMENTS



## Related Software

### Device Software

Device functionality is built with BGScripts scripting language. This is short description how system currently works.

In startup

* device is set to minimum power saving mode which means that no power saving is done.
* LED0 is configured to show the wifi status (when LED is lit, connection is ok)
* -LED1 shows the network activity
* pushbutton PB0 is configured to start measurement and storing sequence when pushed. Should be done only when LED0 is lit and LED1 is blank
* wifi is turned on and connected to activity point (name currently hardcoded)
* when connection is on, both LEDs are lit
* udp (User Datagram Protocol) client will be created, socket opened for connection to NTP (Network Time Protocol) server.
* udp server will be started up to receive time and date from NTP.
* after udp connection is fine, a broadcast message will be sent from client to time server. When server responses, the message comes to our udp server and we caught the data in BGScript event (tcpip\_udp\_data). Date and time will be picked up from message, formatted and set to our RTC (Real Time Clock).
* finally udp sockets will be closed and LED1 will be turned off.
* device starts waiting for PB0 pressings

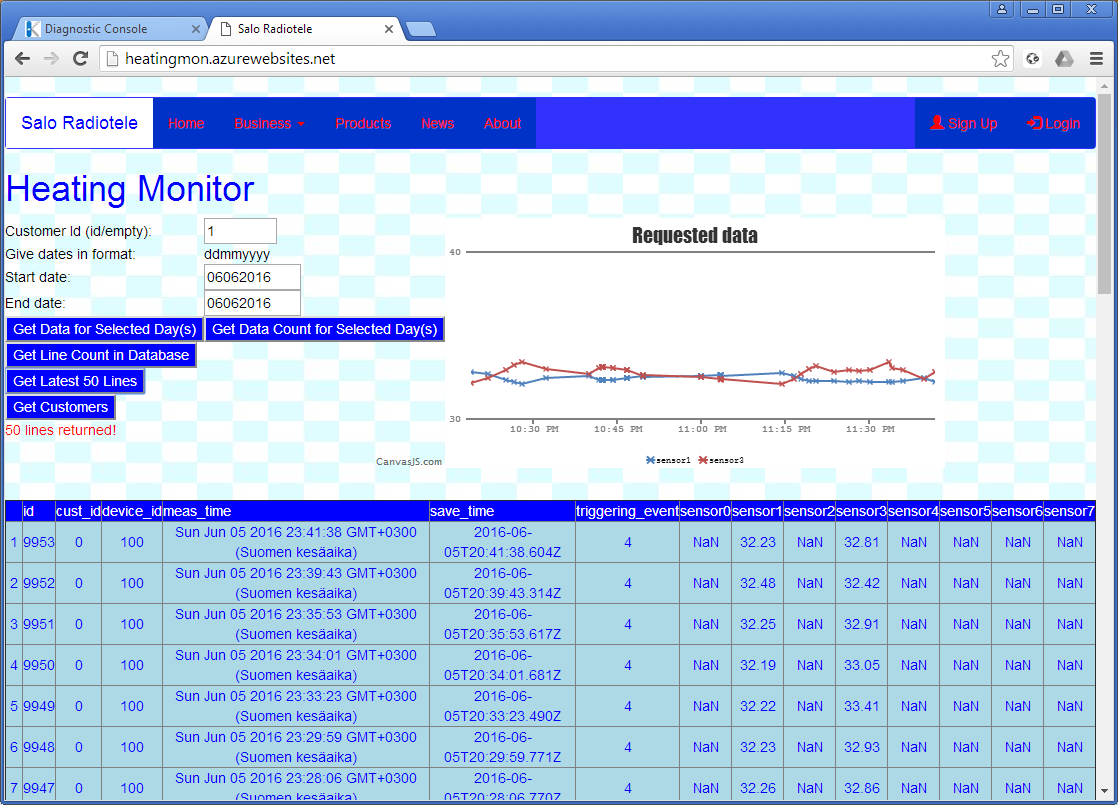
PB0 push

* when pushed, turns on activity LED1
* the device reads local temperature and humidity sensors
* connects to Azure service (currently web page, not IoT hub, because required security settings do not fit into TCP message !)
* gets current time from RTC and calculates time stamp for the measurement
* builds TCP (Transmission Control Protocol) message containing time and measurements (most of other parameters are currently hardcoded)
* sends message to Azure
* after receiving the response, disconnects the Azure socket
* turns off LED1
* goes waiting a new PB0 push

### Web Server Software

HeatMonServer.js

### Client Software

Web page prototype (HeatingMon.html) is set up in Azure to demonstrate the monitoring functionalities together with service process to handle REST-queries to database. 

Purely in demonstration purposes I have developed a command line program in node environment that creates records and sends them to Azure service bus.

Following is format the data must be sent:

-Request options----------------------------------

{ hostname: 'heatingmon.servicebus.windows.net',

port: 443,

path: '/heatingmon/messages',

method: 'POST',

headers:

{ Authorization: 'SharedAccessSignature sr=https%3A%2F%2Fheatingmon.servicebu

s.windows.net%2Fheatingmon%2Fmessages&sig=AXFqrfzxtspQdbI%2BTAiIhLQ6c73YRb1Ceggh2oqvN1s%3D&se=1463823130&skn=heatingmon',

'Content-Length': 191,

'Content-Type': 'application/atom+xml;type=entry;charset=utf-8' } }

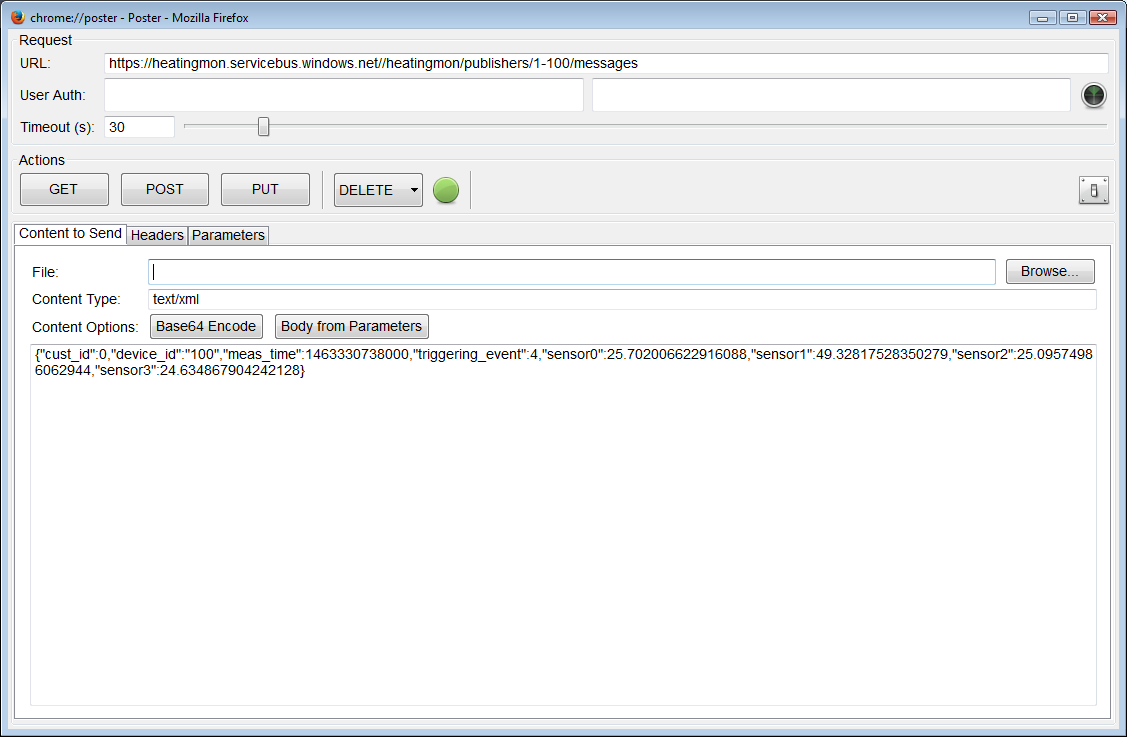
-Payload------------------------------------------

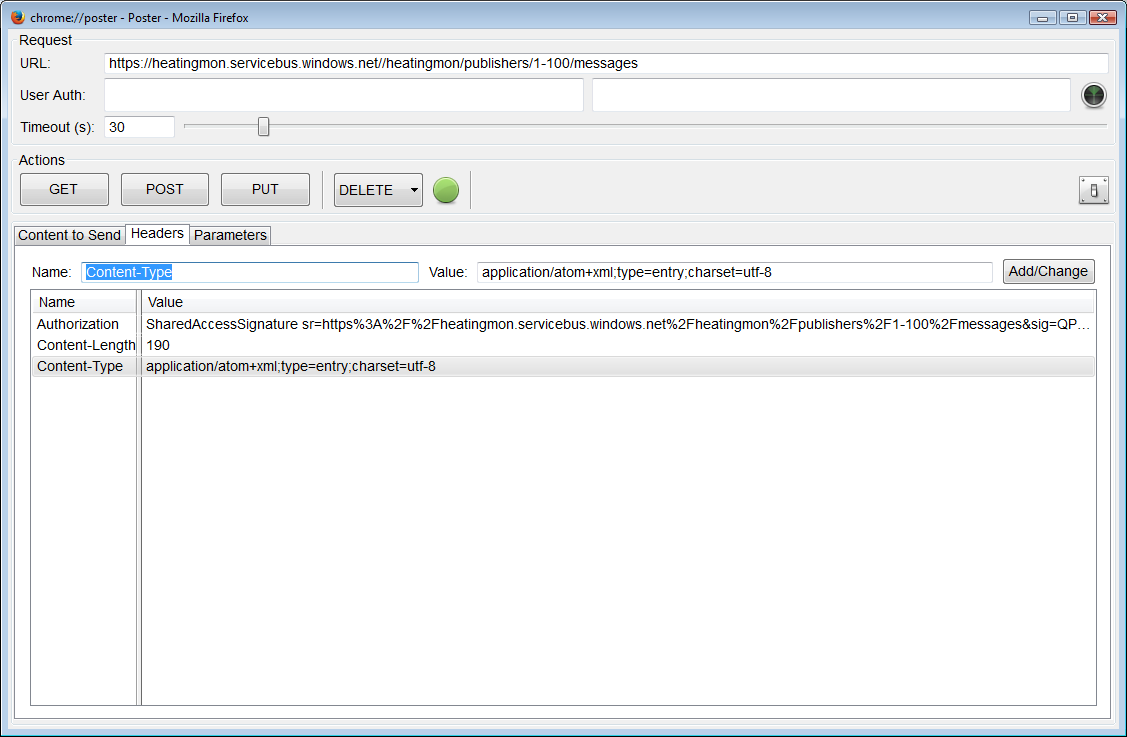
{"cust\_id":4,"device\_id":"100","meas\_time":1462527170000,"triggering\_event":2,"s

ensor0":26.0433927080594,"sensor1":48.47145873354748,"sensor2":26.52142556221224

4,"sensor3":24.464965699939057}

Messages can be sent to Azure also manually using Poster: (note that Authorization key must be correct and active)





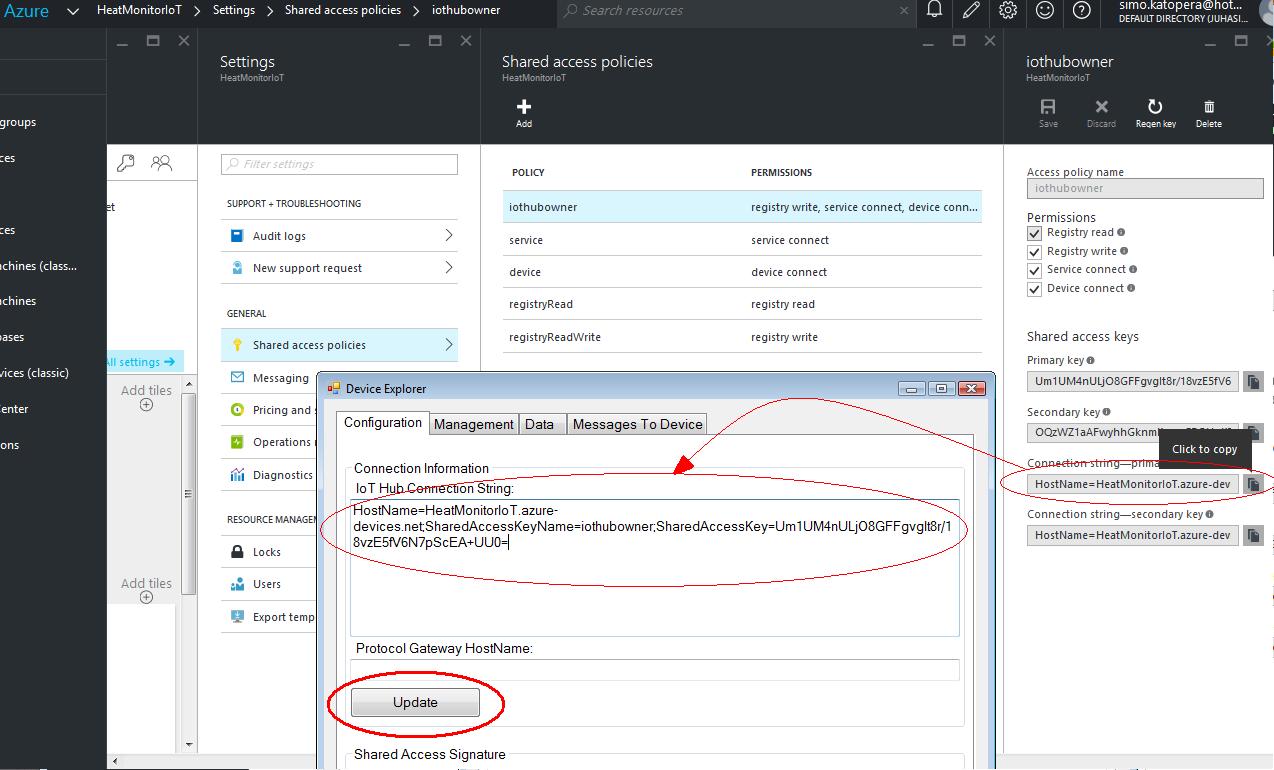
**Authorization format:** SharedAccessSignature sr=https%3A%2F%2Fheatingmon.servicebus.windows.net%2Fheatingmon%2Fpublishers%2F1-100%2Fmessages&sig=QPSRfwZQMSBGhQrz3cMhc%2F9ZZWDij4gI%2BfyjOOiM0XY%3D&se=1464627147&skn=heatingmon

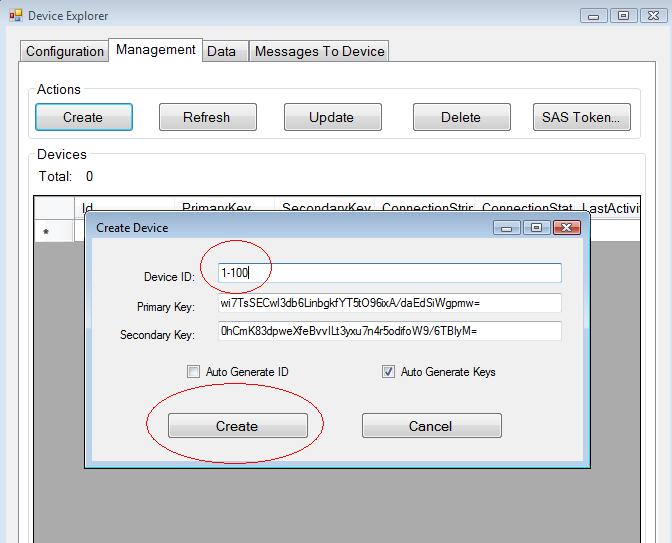
## Using IoT Hub

We can use iot hub instead of combinations of service bus and event hub . With Device Explorer (in GitHub) you can control devices connected to your iot hub. Each iot hub has got a Device Identity Registry which data need to be modified to allow/disallow devices connect to hub. The field in registry are deviceId, generationId, etag, auth, auth.symkey, status, statusReason, statusUpdateTime, connectionState, connectionStateUpdatedTime and lastActivityTime (see <https://azure.microsoft.com/en-us/documentation/articles/iot-hub-devguide/>).

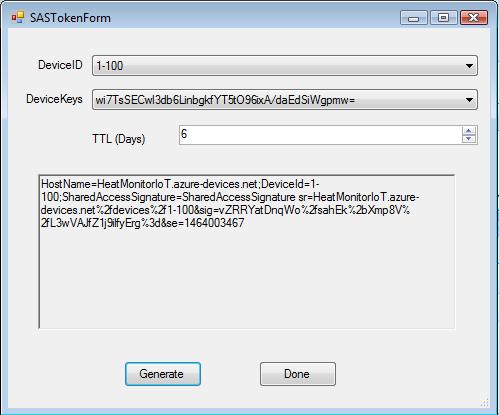
### Device Explorer

Here is how you get Device Explorer in use after loading and installing it. Go to IoT Hub “Shared Access Policies” in Azure portal. Select “iothubowner” policy and copy the Connection string for primary (or secondary) key by clicking “Click to copy” button on the right. Paste it on Device Explorer Configuration sheet’s “IoT Hub Connection String” field. Click Update button to take settings in use.



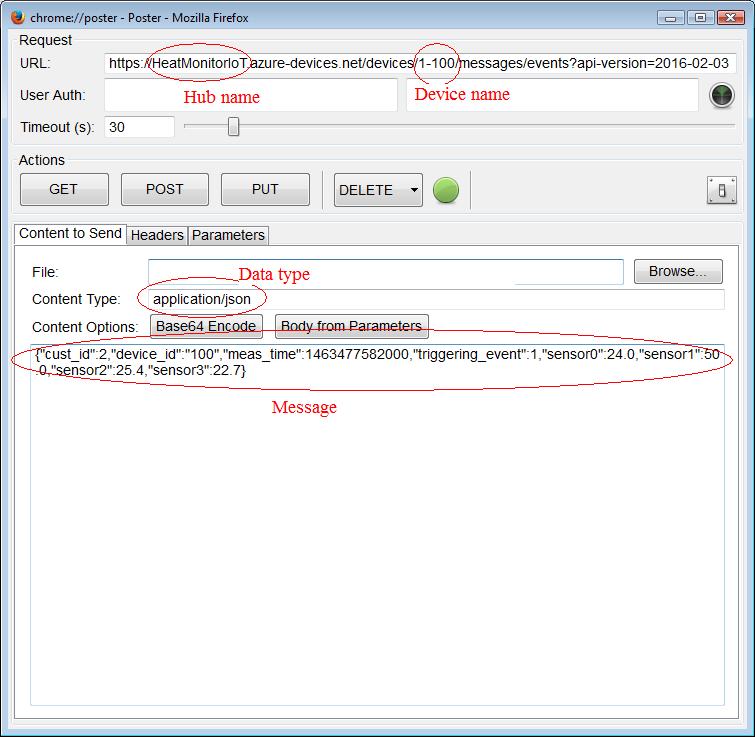
Go to Management sheet on Device Explorer. Click Create button. Give name for device and select Create. Every Device ID should be unique to later allow to control them separately.

Next you should generate “SAS Token” by clicking to button. Select the device you want to create token, give it duration and click “Generate”:

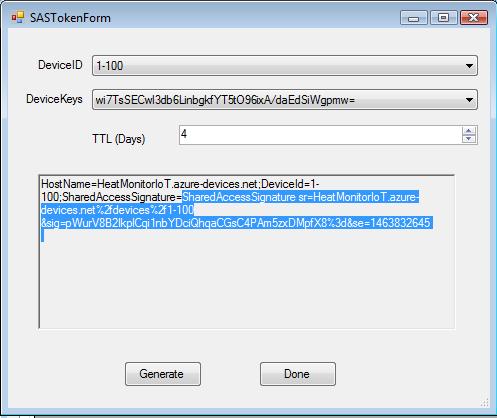


After these you should be able to monitor messages from device to hub (Data) and give commands to device from Device Explorer (Messages to Device).

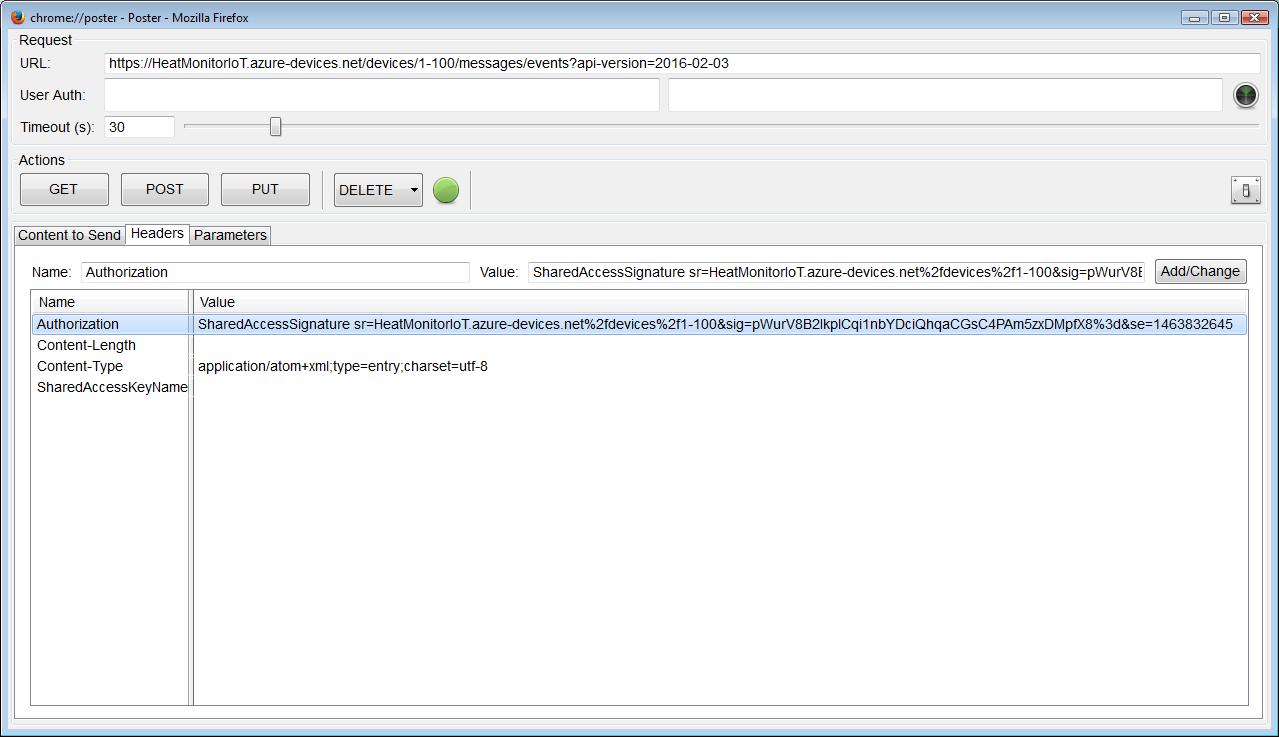
### Posting messages to Azure IoT hub

Sending test data to hub can be easily done using Poster plugin. To send messages, you need to set up Poster as follows. Give the path to IoT hub in URL field. The hub name and device name depend on your hub. In “Content to Send” page, the content type should be set to “application/json”. The data to be sent should be written in the last box in json format. (If the field “meas\_time” is not updated the values in database will show invalid time values.)

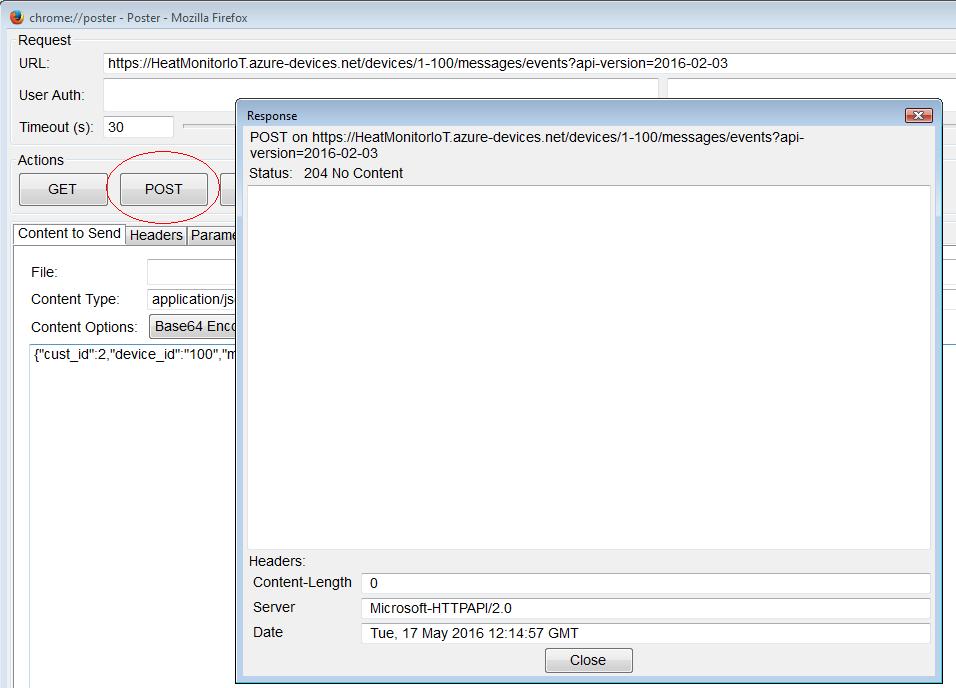
On Headers sheet you need to add “Content-Type” and set the value to “application/atom+xml;type=entry;charset=utf-8” and “Authorization”. The value for Authorization can be copied from Device Explorer’s “SAS token form:



This is what Headers page should look like (don’t care about empty Content-Length and SharedAccessKeyName fields).



Now you can push Post button and response should look like this:



Here, the “Status: 204 No Content” tells that IoT hub received data successfully but did not send any data back.

## Authentication

To be able to send encrypted messages to Event Hub in (https mode) we need to somehow solve how the authentication should be done in our system.

Following is a collection of information about the issue this far.

Event Hub in Azure has got primary and secondary keys that are of format:

wi7TsSECwl3db6LinbgkfYT5tO86ixA/daEdSiWgpmw=

Connection string from Event Hub settings looks like this:

HostName=HeatMonitorIoT.azure-devices.net; DeviceId=1-100; SharedAccessKey=wi7TsSECwl3db6LinbgkfYT5tO86ixA/daEdSiWgpmw=

Where “DeviceId” is the name of our device, same name must exist as device in the url:

https://HeatMonitorIoT.azure-devices.net/devices/1-100/messages/events?api-version=2016-02-03

and SharedAccessKey is either primary or secondary key of the Hub

SAS Token generator in “Device Explorer” creates for our device(1-100) token like this:

HostName=HeatMonitorIoT.azure-devices.net;DeviceId=1-100;SharedAccessSignature=SharedAccessSignature sr=HeatMonitorIoT.azure-devices.net%2fdevices%2f1-100&sig=B9YjvwnmTnY9K3c95GEHq3A8XIZDzqzRz%2fLMh8NGRR8%3d&se=1466185738

This is the data that must be inserted to http message (in Poster, for example).

The calculation of sig (SAStoken) is shown on Azure web-pages:

https://azure.microsoft.com/en-us/documentation/articles/iot-hub-sas-tokens/

Obviously WGM110 uses pem-file to make tcpip\_ssl\_connect –function pass. Pem files contain encrypted certificates and private keys. Short explanation: http://how2ssl.com/articles/working\_with\_pem\_files/

* Is it the shared signature that needs to be put into pem?
* What exactly should be put in pem file?
* How to create the pem file?

## Real Time Handling

To get measurement times to be saved into database with results, we need to read time each time we make a measurement.

### Getting the time

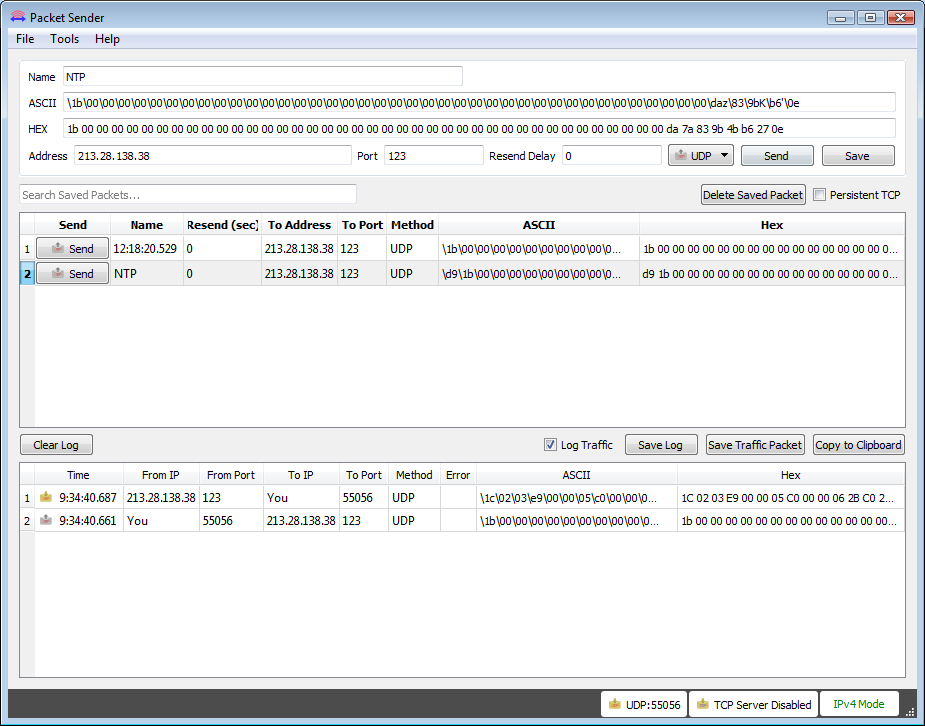
To be sure that Gecko WGM110 RTC (Real Time Clock) is running correct time it will be updated periodically from NTP (Network Time Protocol) server.

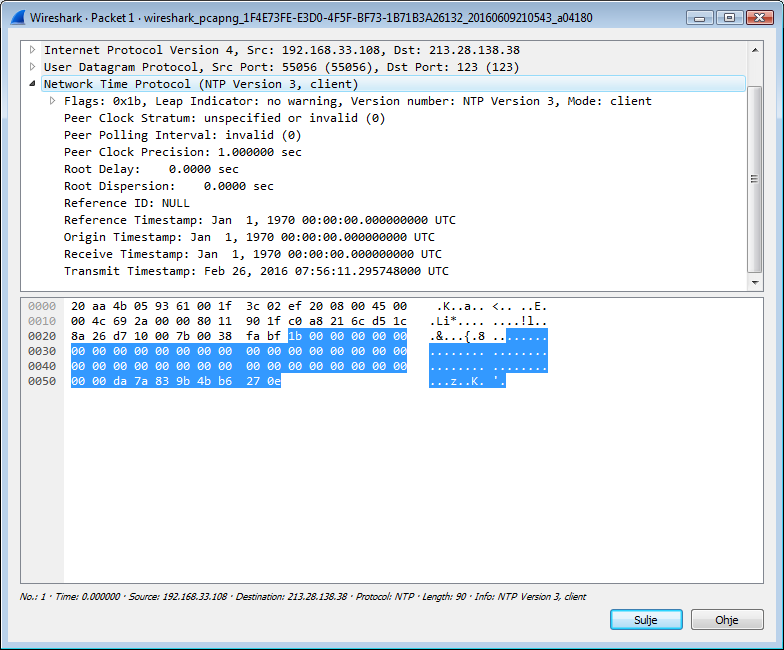
BGScript reads the time from NTP server (name given in device’s configuration) using UDP (User Datagram Protocol) request. Because our system does not require very accurate timing information, we don’t use any adjusting algorithm, but just use directly the time we get from server on one request.

At the moment, the time is updated (only) in bootup, but it could be done perhaps once a day or every time when time looks wrong.

To be able to use udp in BGScript, we need to setup both a client to send messages and a server to catch responses. By default in this system, the two sockets use different ports. There is a way to directed them to same location.

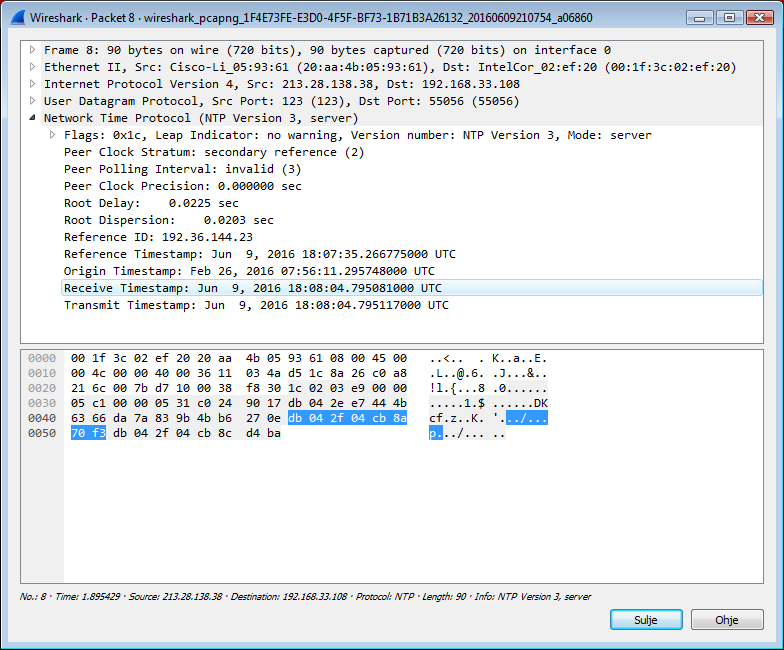
Currently BGScript uses very simple request message where only version field is set to some meaningful value. See following pictures:





In BGScript file, the very same message looks like this: "\x1B\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xda\x7a\x83\x9b\x4b\xb6\x27\x0e"

And the response is shown in following picture:

From the response the painted bytes contain the time (one of the times) NTP server returns. The first four bytes contain the time in seconds (db042f04) from January the 1st 1900. We do not care the rest (parts of second) because our RTC clock does not care. Converting hex to decimal, that value is some 3674484484 seconds. From that value we then calculate (in BGScript) the current date and time in h:m:s d:m:y format and save the values to RTC registers.

To make calculations easier, we use year 2016 as start point for our calculations. One of the most important reasons to use smaller values is that amount of seconds from NTP server is too big to fit in 32 bits variables that we use in BGScript. What we do next is subtract 3660595200 seconds from the count and get that in this example we are 3674484484-3660595200 = 13889284 seconds away from 1.1.2016. From this value we then calculate the amount of days, months, years, hours, minutes and seconds and set all those to RTC.

### Using the time

When we make measurements, we also read from our RTC the current time. This time will be inserted to the message we send to the cloud. Unfortunately, the time in database is in different format than in our clock. This means we need to do some calculations.