webdyn

Flexitron group

TITAN

Application Note 62

Sending data to DEXMA platform

Sending data to DEXMA platform

1. Scenario details

TITAN routers have all the typical functionalities of 4G/3G/2G routers, as well as a series of added features that make them one of the most feature-packed routers on the market.

One of the added features is the datalogger, where the TITAN router can store a number of types of records in its non-volatile memory in JSON format. These records can come from Modbus readings, SERIAL data captures via the RS232 / RS485 ports, or GPS positions, etc. These JSON-type records are stored in the TITAN router's internal non-volatile memory and can subsequently be sent to remote platforms via protocols such as HTTP, HTTPS, MQTT, MQTTS, FTP and FTPS

As mentioned, the TITAN router stores the JSON registers in its internal memory in a proprietary format by default. This can sometimes be a problem when communicating with platforms that expect to receive information in a certain format (i.e. a format other than JSON, the one used by the TITAN router).

The "**JSON Transformer Function Script**" enables the TITAN router to format any JSON object before it is stored in the internal memory. This means JSON objects can be converted to the appropriate format for each platform. You don't really need to convert the data into a JSON format, since it can be converted into any String format.

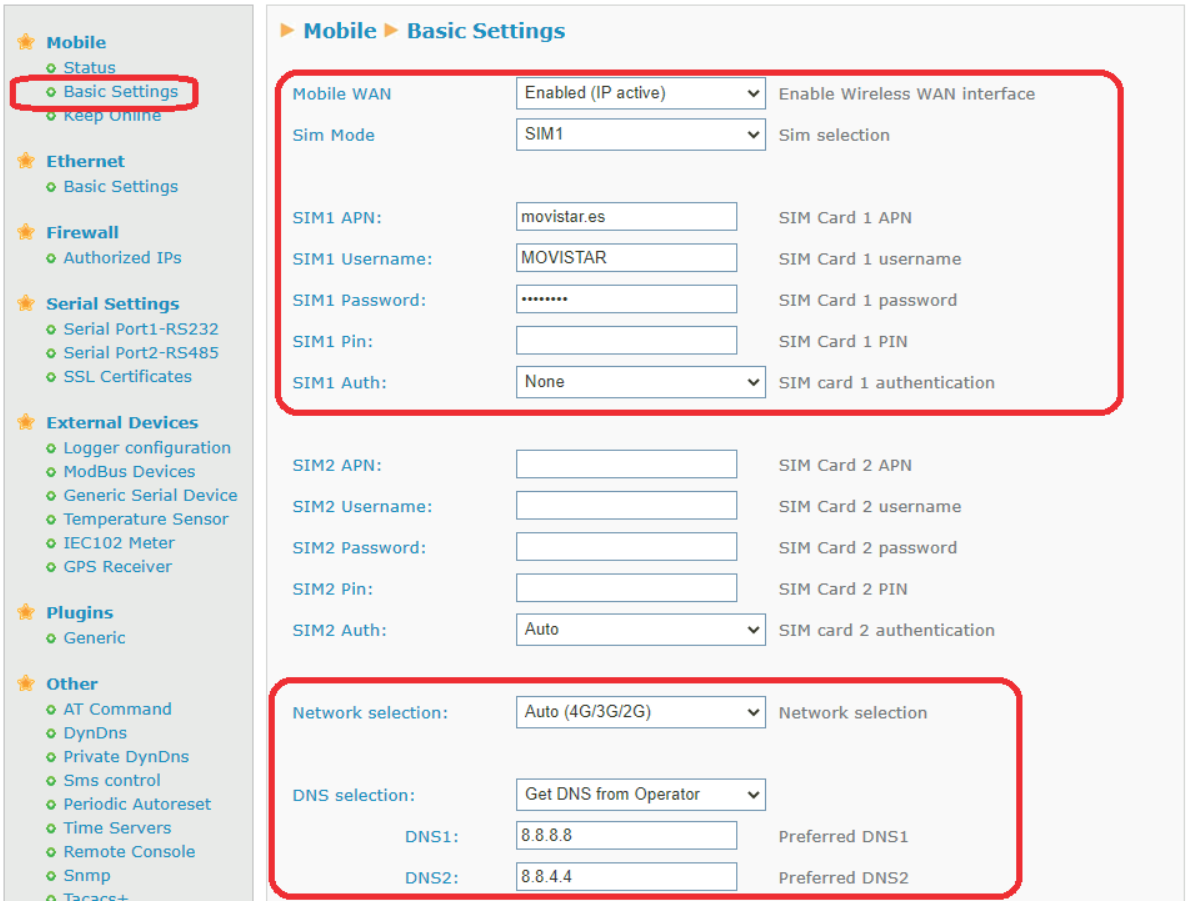
In this application note, we will guide you through an entire example of how send to data to the well-known Dexma platform (https://www.dexma.com/what-is-dexma-platform/), which requires the sent JSONs to be in a special format and a series of headers.

In this particular application note, we will assume that 2 registers are to be read from 2 PLCs with Modbus communications connected to a Webdyn EasyTunnel via their RS485 port.

The main aim is for the WebDyn-Easytunnel device to read the Modbus registers with addresses 40000 and 40001 of PLC1, and registers 40000 and 40001 of PLC2, every minute. In both PLCs, register 40000 corresponds to the measured temperature, and register 40001 to the humidity level. The readings taken must be stored in the internal non-volatile memory of the Webdyn EasyTunnel (in its datalogger), which must send the read data to the DEXMA platform whenever possible (coverage, IP connectivity, etc.). Communication with the PLCs is carried out via an RS485 bus with a 9600,8,N,1 configuration

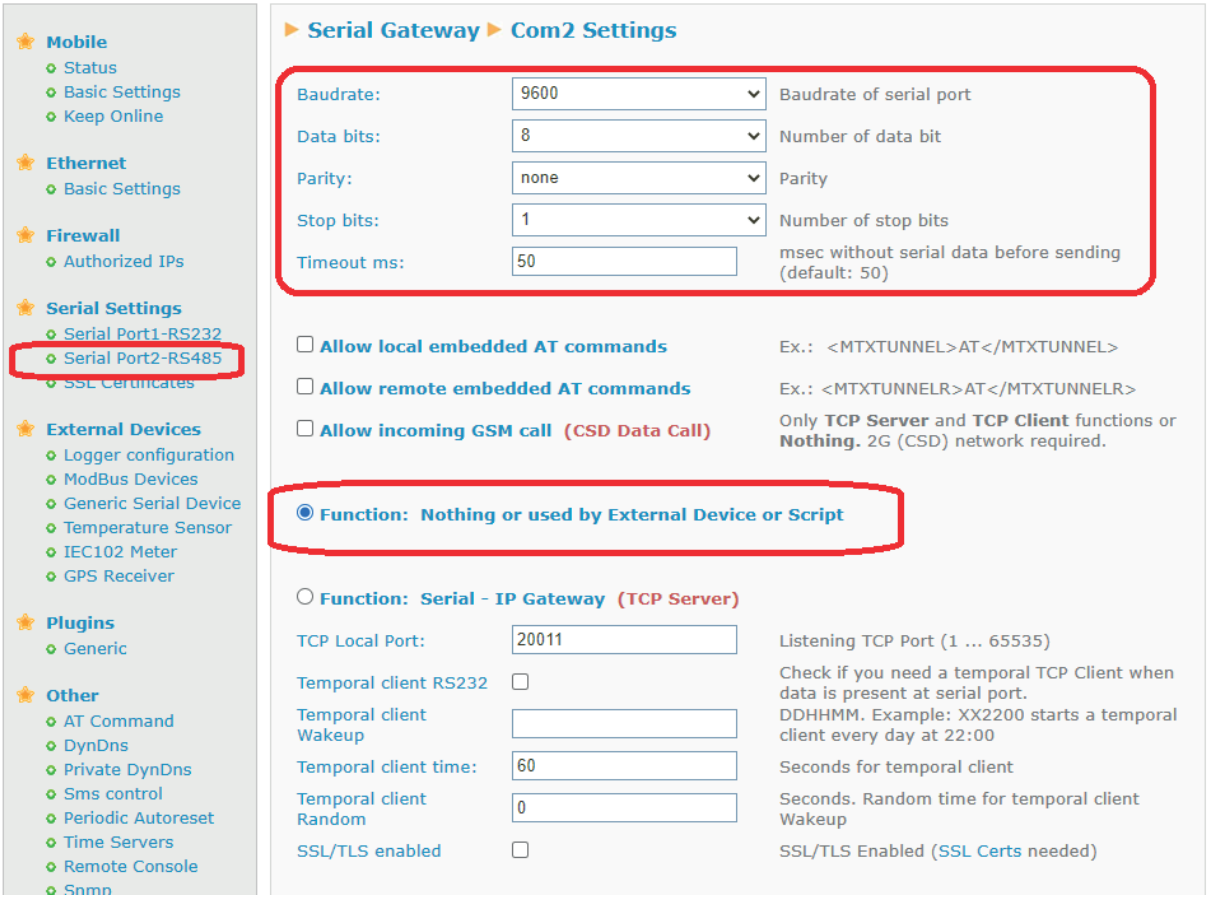
2. WAN mobile configuration

The Webdyn EasyTunnel must communicate with the DEXMA platform via 4G/3G/2G communications, so the "**Mobile=>Basic Settings**" section must be configured correctly according to the SIM card used.



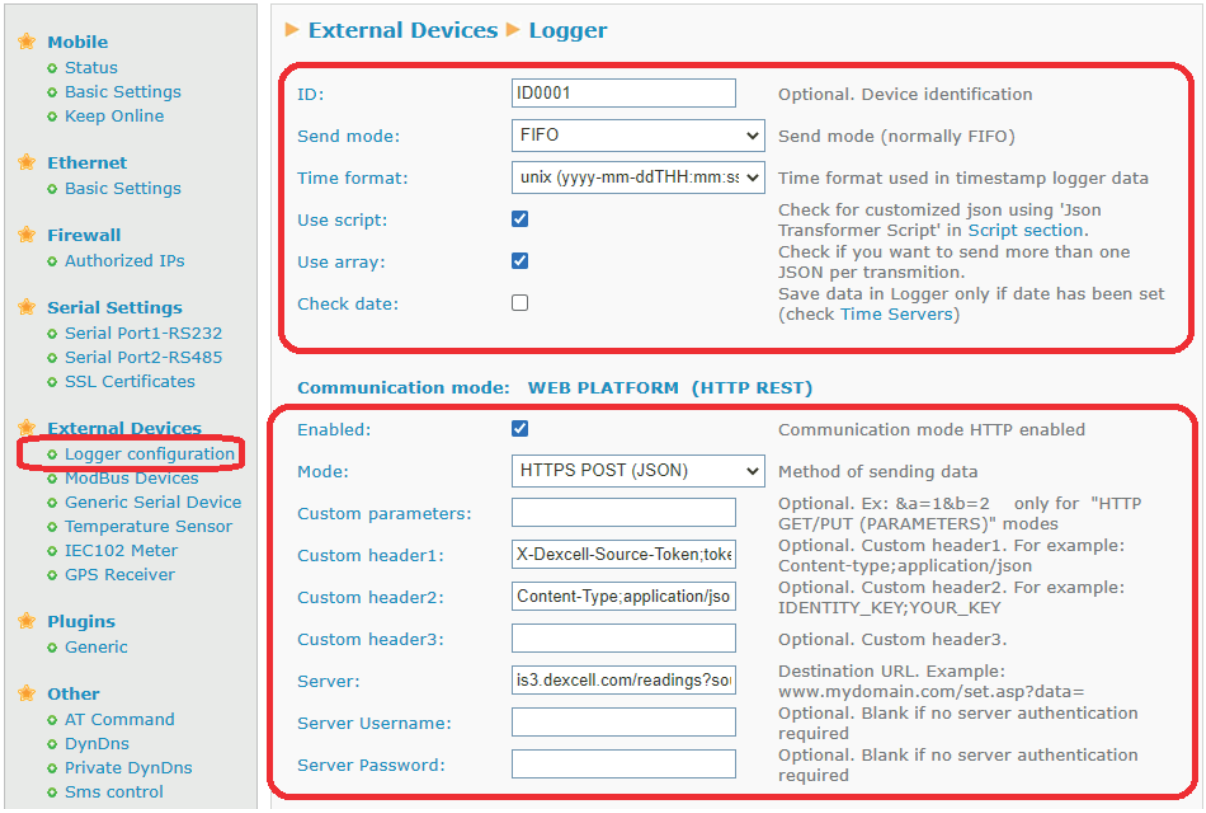
3. Configuring the RS485 serial port

The two PLCs will be connected to 9600,8,N,1 via the RS485 serial port, so the "**Serial Settings => Serial Port2-RS485**" section must be configured by setting the parameters as shown below.



4. Logger configuration

The next step is to configure the internal datalogger of the WebDyn EasyTunnel. Go to the "**External Devices => Logger configuration**" menu. The configuration should be similar to the one shown in the screenshot below:



The following parameters are particularly important:

- "**Time format**": you must select the "unix" format, which is supported by the DEXMA platform.
- "**Use script**": you must check this box, because activating it will run the script that you will integrate later. This script will also convert the standard JSON created by the WebDyn-Easytunnel into a JSON with the required format supported by the DEXMA platform.
- "**Use array**": DEXMA allows data to be sent in an array, so you must also check this box.
- "**Custom header1**": you must enter this as indicated in https://support.dexma.com/hc/en-gb/articles/360013772759-Using-the-insertion-API-to-introduce-data-in-a-gateway the header "x-dexcell-sorce-token", which is different for each device. It has the following value in this example:

X-Dexcell-Source-Token;token123456789 (fields separated by ";", not ":")

- "**Custom header2**": you must enter the following:

Content-Type;application/json (fields separated by ";", not ":")

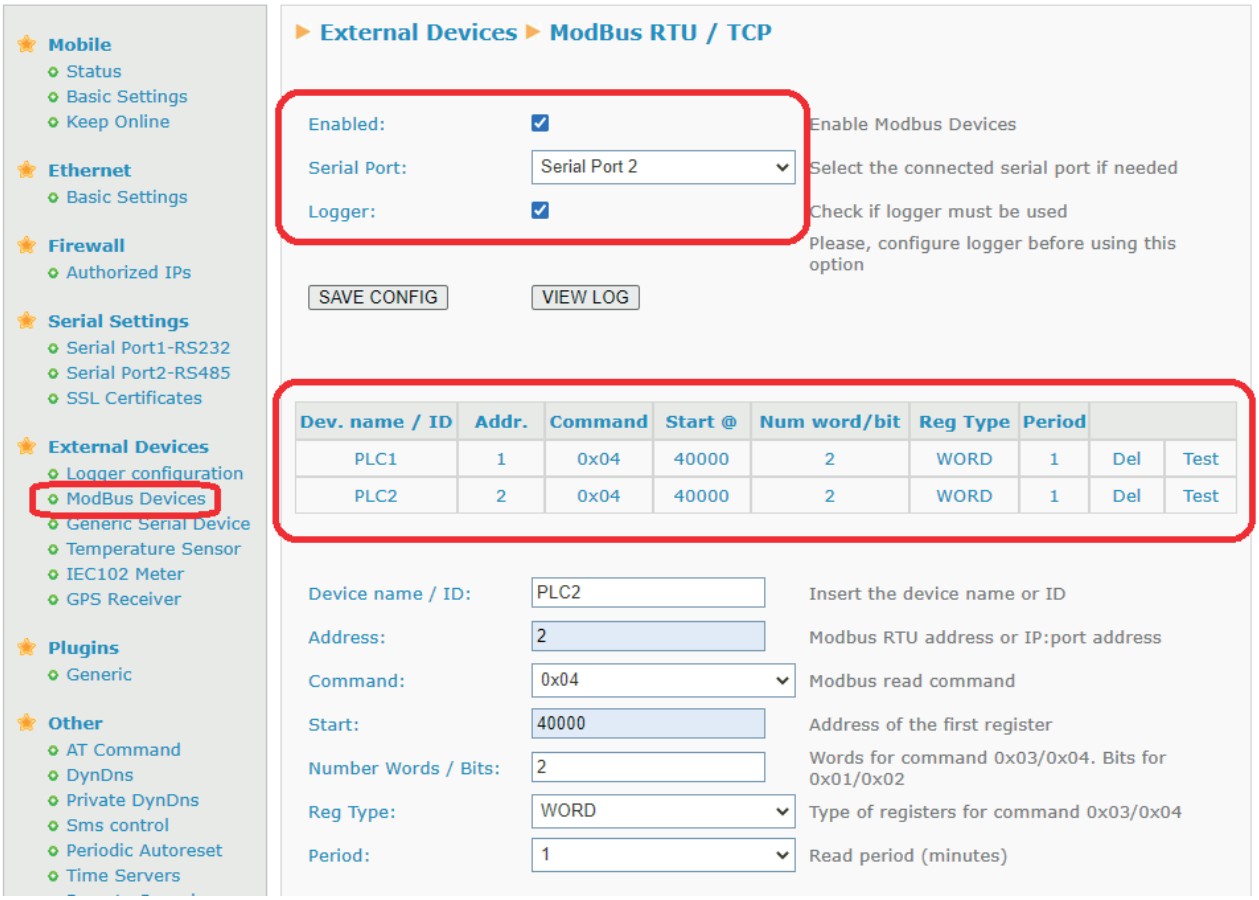
- "**Server**": enter the URL of the server here, which will take the following format:

is3.dexcell.com/readings?source_key=mac-123456789,

where mac-123456789 will need to be replaced accordingly by the KEY obtained on your DEXMA platform.

5. Configuring the Modbus section

In this configuration section ("**External Devices => Modbus Devices**"), you will configure the Modbus readings to be performed on the 2 PLCs



You will also enable the Modbus service by checking the "**Enabled**" box. You must select "**Serial Port 2**", since the reading will be made via the RS485 port. The "**Logger**" box must also be activated, since the Modbus registers read from the PLCs must be stored in the internal datalogger of the Webdyn EasyTunnel.

Two devices must also be created, which we have called PLC1 and PLC2 in this example.

In the "**address**" field, you must indicate the Modbus address of each of the PLCs, so "1" in PLC1 and "2" in PLC2. The Modbus command you will use to read registers 40000 and 40001 will be 0x04, so select 0x04 in the "**Command**" field. The register that will start to be read in both PLCs is register 40000, so enter 40000 in the "**Start**" field. The aim is to read 2 Modbus registers from each PLC (40000 and 40001), so enter 2 in the "**Number Words**" field.

Select “WORD” for the register type (“**Reg Type**”) and 1 for the “**Period**”, because we want to receive readings from the Modbus registers every minute.

6. SCRIPT configuration to convert the JSON generated by the Webdyn EasyTunnel with a standard format to the format required for correct communication with the DEXAM platform

The standard JSON format generated by the Webdyn EasyTunnel with the data read, and which will send this data to a platform, will take the following format:

[

{“IMEI”:”869101054286683”,”TYPE”:”MODB”,”TS”:”2022-11-11T12:17:00Z”,”ID”:”PLC1”,”A”:”1”,”ST”:”40000”,”N”:”2”,”V”:[225,62],”P”:”ID0001”},

{“IMEI”:”869101054286683”,”TYPE”:”MODB”,”TS”:”2022-11-11T12:17:01Z”,”ID”:”PLC2”,”A”:”2”,”ST”:”40000”,”N”:”2”,”V”:[225,62],”P”:”ID0001”}

]

But the format required by the DEXMA platform, as indicated in its manuals, (https://support.dexma.com/hc/en-gb/articles/360013772759-Using-the-insertion-API-to-introduce-data-in-a-gateway) must be:

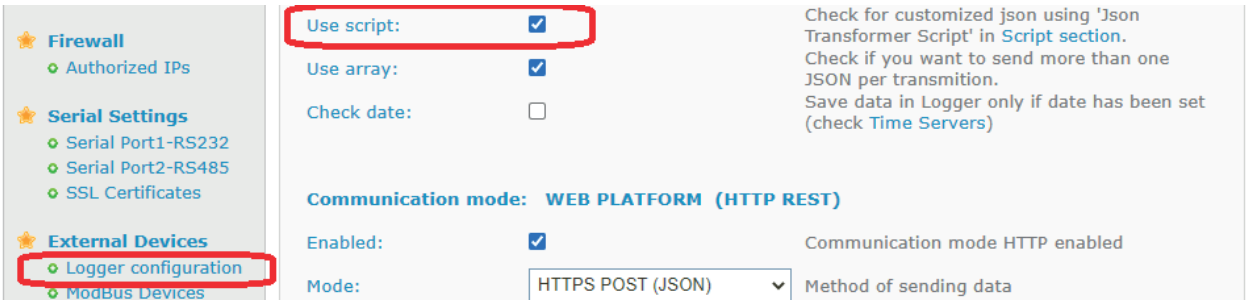
[

{“did”:”PLC1”,”sqn”:1,”ts”:”2022-11-11T12:20:01Z”,”values”:[{“p”:301,”v”:22.5},{“p”:401,”v”:62}]},

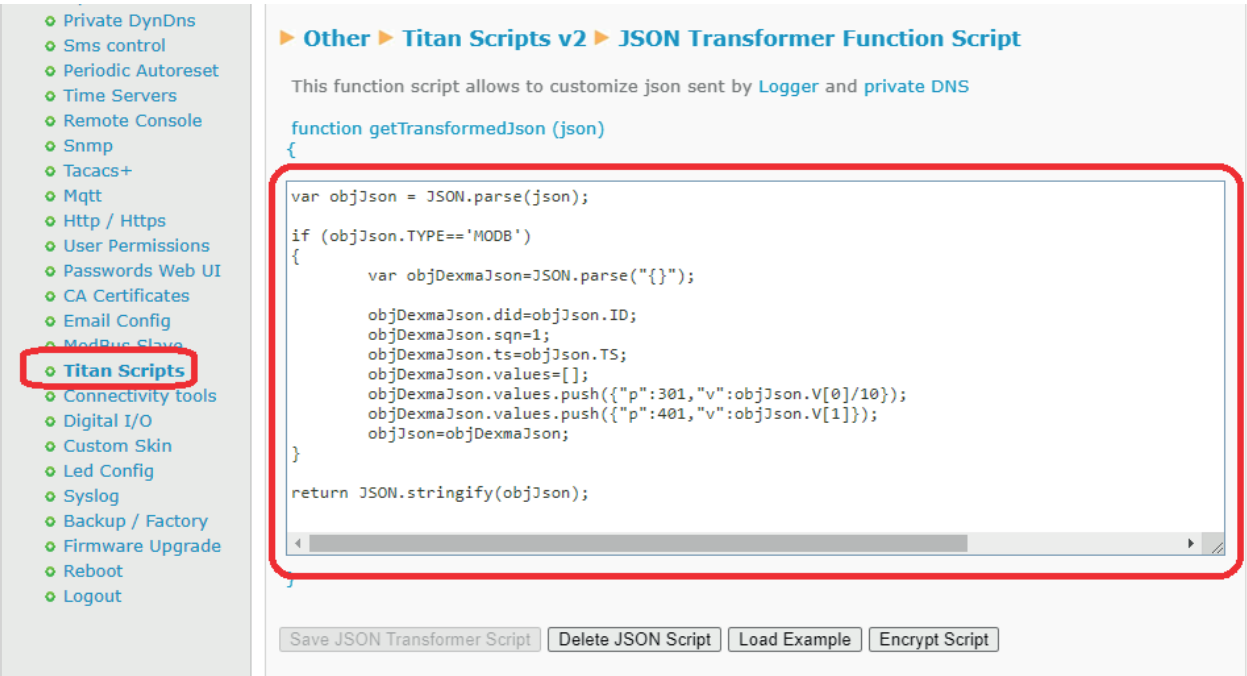
{“did”:”PLC2”,”sqn”:1,”ts”:”2022-11-11T12:20:01Z”,”values”:[{“p”:301,”v”:22.5},{“p”:401,”v”:62}]}

]

A small SCRIPT must therefore be created to convert the data. Remember that this conversion script will be run because you previously checked the “**Use script**” box in the “**External Devices” => Logger Configuration**” section



To enter the code for the data conversion script, go to the "**Other => Titan Scripts**" menu. The conversion script must be entered in the "**JSON Transformer Function Script**" section.



As indicated in the manual, this function to format the JSON is executed BEFORE storing the 1-read data in the internal datalogger and the JSON becomes a parameter with the standard format of the device, i.e. for example:

{"IMEI":"869101054286683","TYPE":"MODB","TS":"2022-11-11T12:17:00Z","ID":"PLC1","A":"1","ST":"40000","N":"2","V":[225,62],"P":"ID0001"}

The conversion script must convert this previous JSON into the following one to be compatible with the DEXMA platform.

{"did":"PLC1","sqn":1,"ts":"2022-11-11T12:20:01Z","values":[{"p":301,"v":22.5},{"p":401,"v":62}]}

This is what the SCRIPT code would look like (with comments added).

```
//Create a JSON-type object variable, because the "json" variable that receives the //function as an argument is of type "String".
var objJson = JSON.parse(json);
```

```
//If it is indicated in the original JSON received that it is a Modbus data JSON ...
if (objJson.TYPE==’MODB’)
{
        //Create a new JSON object where you will store the data
        //converted
        var objDexmaJson=JSON.parse("{}");

        //Use the JSON ID field as the did field of the DEXMA platform
        //original (i.e. PLC1 and PLC2)
        objDexmaJson.did=objJson.ID;

        //sqn field, always 1.
        objDexmaJson.sqn=1;

        //Use the same timestamp as the original JSON
        objDexmaJson.ts=objJson.TS;

        //Add an array to the JSON where you will store the values for
        //temperature and humidity.
        objDexmaJson.values=[];

        //Add parameter 301 to the array, as well as the temperature value, which //corresponds to the first register read (40000), stored in position //0 of the original JSON array (highlighted in green on the previous page) //Divide the value by 10 to make it 1 //decimal.
        objDexmaJson.values.push({"p":301, "v":objJson.V[0]/10});

        // Add parameter 401 to the array, as well as the humidity value, which corresponds to the second register read (40001), stored in position 1 of the original JSON array (highlighted in red on the previous page)
        objDexmaJson.values.push({"p":401, "v":objJson.V[1]});
```

```
        //Then, copy the converted JSON to return it as a result
        // of the conversion function.
        objJson=objDexmaJson;
}
//Return the JSON in String format as a result of the //conversion function.
return JSON.stringify(objJson);
```

Once the script has been entered and saved, simply restart the device and everything should work correctly. If you have any issues, check the device LOGs and whether you need to enter any digital certificate in the "Other => Ca Certificates" section.

Any questions?

Please direct your enquiries to iotsupport@mtxm2m.com