TITAN

Application Note 57

JSON Transformer Script

JSON Transformer Script

1. Scenario Details

TITAN-based devices 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, the TITAN-based device can store a number of types of records in its non-volatile memory in JSON format. These records can come from Modbus reads, SERIAL data captures via the RS232 / RS485 ports, or GPS positions, etc. These JSON-type records are stored in the TITAN-based device'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 metioned, the TITAN-based device stores the JSON records 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-based device).

The JSON Transformer Function Script enables the TITAN-based device 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.

2. The Default JSON Objects Used by the TITAN-based Device

The default JSON objects used by the TITAN-based device for various types of sends are listed below.

a) Example of the change of state of a digital input:

{“TYPE”:”GPIO”,”IMEI”:”869101054287806”,“TS”:”2022-06-09T14:53:04Z”,”ID”:0,”VALUE”:1,”DIR”:”INPUT”,”P”:”ID0001”}

b) Example of sending all I/Os periodically:

{“TYPE”:”IOS”,”IMEI”:”869101054287806”,”P”:”ID0001”,“TS”:”2022-06-09T14:51:00Z”,”IO0”:0,”IO1”:0,”CO0”:0}

c) Example of sending Modbus registers:

{“TYPE”:”MODB”,”IMEI”:”869101054287806”,“TS”:”2022-06-09T14:54:01Z”,”ID”:”DOS”,“A”:”192.168.1.28:502”,”ST”:”1”,”N”:”5”,”V”:[1,2,3,4,5],”P”:”ID0001”}

d) Example of sending status frames (DNS):

{“TYPE”:”DNS”,”IMEI”:”869101054287806”,”IP”:”88.28.221.24”,”P”:””,“CSQ”:21,”MOD”:””,

"VER":"5.2.6.07","IMSI":"214075536243578",“TECH":"4g","TS":"2022-06-09T14:56:00Z",
"CID":"214;07;219B;139770C","RSSI":"-73","RSRP":"-102","RSRQ":"-9"}

e) Example of sending frames using a SCRIPT:

{“TYPE":"SCRIPT","TS":"2022-06-09T15:02:02Z",“IMEI":"869101054287806","P":"ID0001","DA
TA":"1,2,3"}

f) Example of sending SERIAL port capture frames:

{“TYPE":"SERIAL","TS":"2022-06-09T15:02:02Z",“IMEI":"869101054287806","P":"ID0001","DATA":"4
143DF3412E0A0"}

g) Example of sending power loss frames (for devices with a battery/supercap):

{“TYPE":"POWER","TS":"2022-06-09T15:02:02Z",“IMEI":"869101054287806","P":"ID0001","P
OW":"0"}

h) Example of sending GPS position frames (for devices with GPS):

{“TYPE":"GPS",“IMEI":"869101054287806","P":"ID0001",“DATE":"2022-06-09","TIME":"10:01:00","L
AT":"41.3823","LON":"2.2126",“NS":"N","EW":"W","SPE","0.0","COU":"123","ALT":"0.00","STA":"2"}

3. EXAMPLE 1: Changing the format of the JSON object to send it to the MQTT platform, which requires a special JSON format.

Let's imagine an example scenario in which we need to read Modbus registers and send them to the MQTT platform, as indicated in Application Note ANV6_32

In that Application Note, the JSON format used after simply reading the Modbus registers would have the following format:

{“TYPE”:”MODB”,”ID”:”2”,”TS”:”17/06/2017 17:01:05”,”IMEI”:”357044060009633”,”P”:”12345678”,”A”:”2”,”ST”:”10”,”N”:”5”,”V”:[10,11,12,0,0]}

Now let's imagine that we must send the data to an MQTT platform that needs the following JSON format.

{“IMEI”: “357044060009633”,

“data”: {

“TYPE”: “MODB”,

“ID”: “2”,

“TS”: “17/06/2017 17:01:05”,

“P”: “12345678”,

“A”: “2”,

“ST”: “10”,

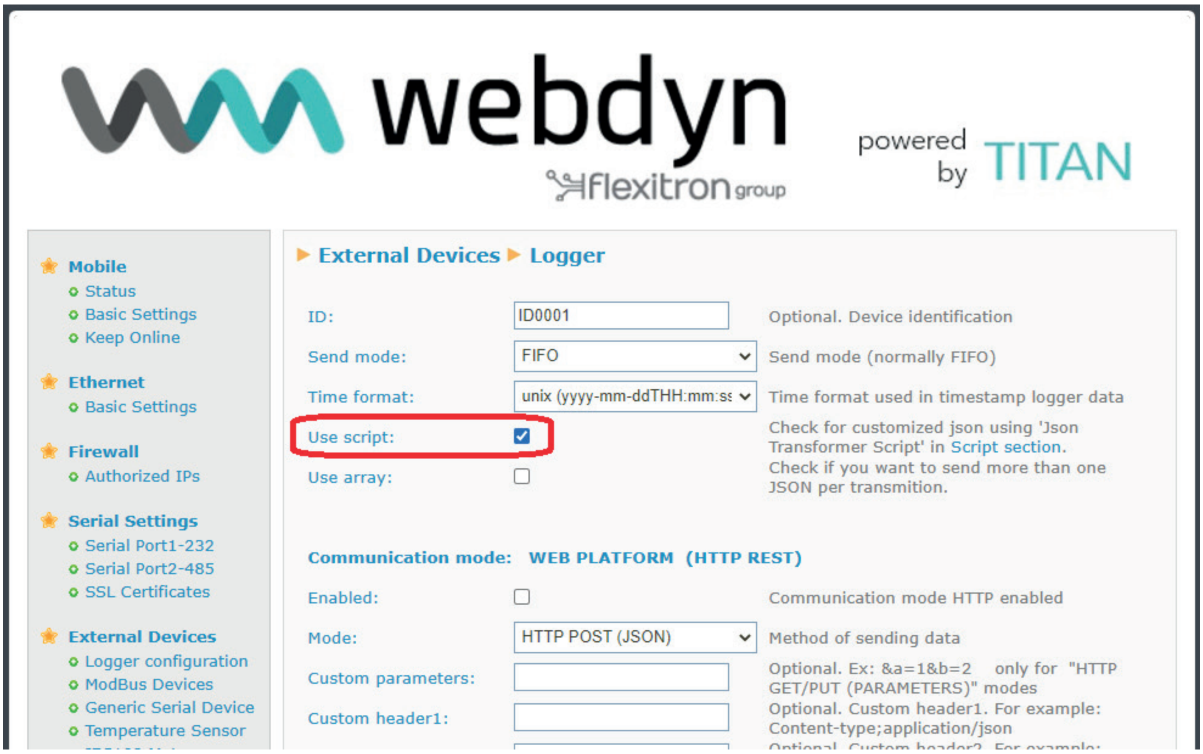
“N”: “5”,

“V”: [10, 11, 12, 0, 0]

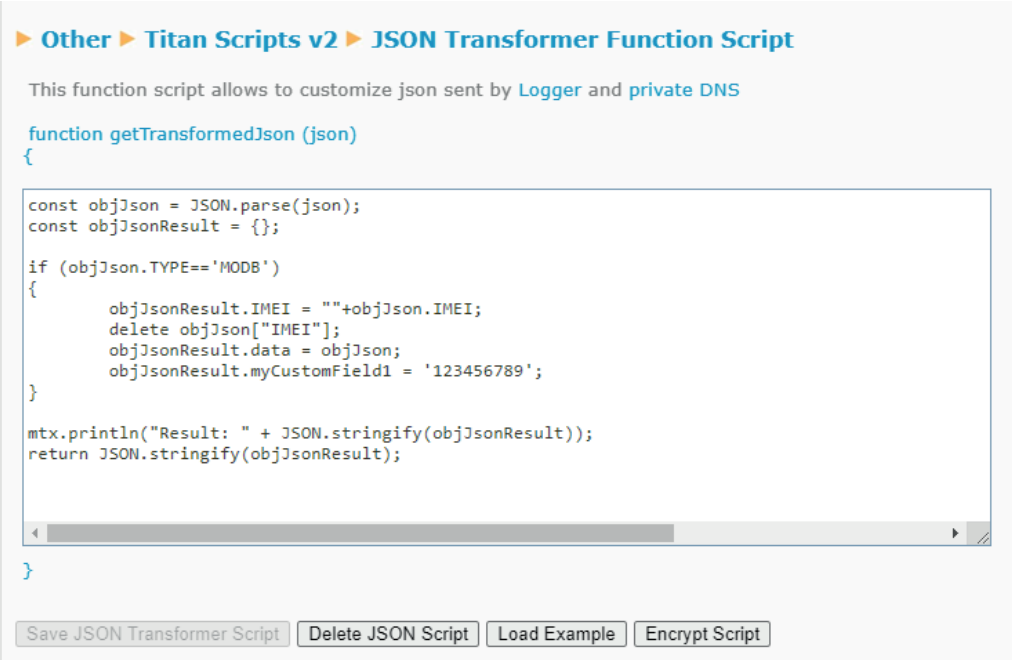
},

“customField1”: “123456789”}

To transform the original JSON to the JSON needed by the platform, two things must be done. The first is to indicate of the LOGGER section that you intend to use the JSON Transformer Script function in the configuration screen. This is done in the “OTHER >Logger Configuration” configuration menu, check the “Use Script” box as shown in the following figure:



You can then access the “OTHER > Powered by Titan Scripts” section and the JSON format conversion script. This function receives a String containing the input JSON as a parameter in the "json" variable (i.e. the JSON with the default format for the TITAN-based device). The function will return a String in the format that we need in order to store it on the device and to subsequently send it to the platform.



In the current example, the script would be coded as follows:

//Create a JSON with the string passed as an argument (original JSON)

const objJson = JSON.parse(json);

//Create a JSON to return the result

const objJsonResult = {};

//If the JSON is of MODB (Modbus) type, we will build the new result JSON

if (objJson.TYPE==’MODB’)

{

    //Add the IMEI field to the result JSON.

    objJsonResult.IMEI = “”+objJson.IMEI;

    //Delete the IMEI field from the original JSON

    delete objJson[“IMEI”];

    //Add the data field and, as the value, we assign the original JSON without the IMEI

    objJsonResult.data = objJson;

    //Add a field and a custom value

    objJsonResult.myCustomField1 = ‘123456789’;

}

```
//Show the result for the standard output.
mtx.println("Result: " + JSON.stringify(objJsonResult));
//Return the String with the result, i.e. the formatted JSON
return JSON.stringify(objJsonResult);
```

The original JSON:

{"TYPE":"MODB","ID":"2","TS":"17/06/2017 17:01:05","IMEI":"357044060009633","P":"12345678","A":"2","ST":"10","N":"5","V":[10,11,12,0,0]}

Is converted to the following:

{"IMEI": "357044060009633",
    "data": {
        "TYPE": "MODB",
        "ID": "2",
        "TS": "17/06/2017 17:01:05",
        "P": "12345678",
        "A": "2",
        "ST": "10",
        "N": "5",
        "V": [10, 11, 12, 0, 0]
    },
    "customField1": "123456789"}

4. EXAMPLE 2: Changing the format of the JSON object to send it to the MQTT platform in a format compatible with MTX-Tunnel.

Let's imagine an example scenario in which we need to read Modbus registers and send them to the MQTT platform, as indicated in Application Note ANV6_32

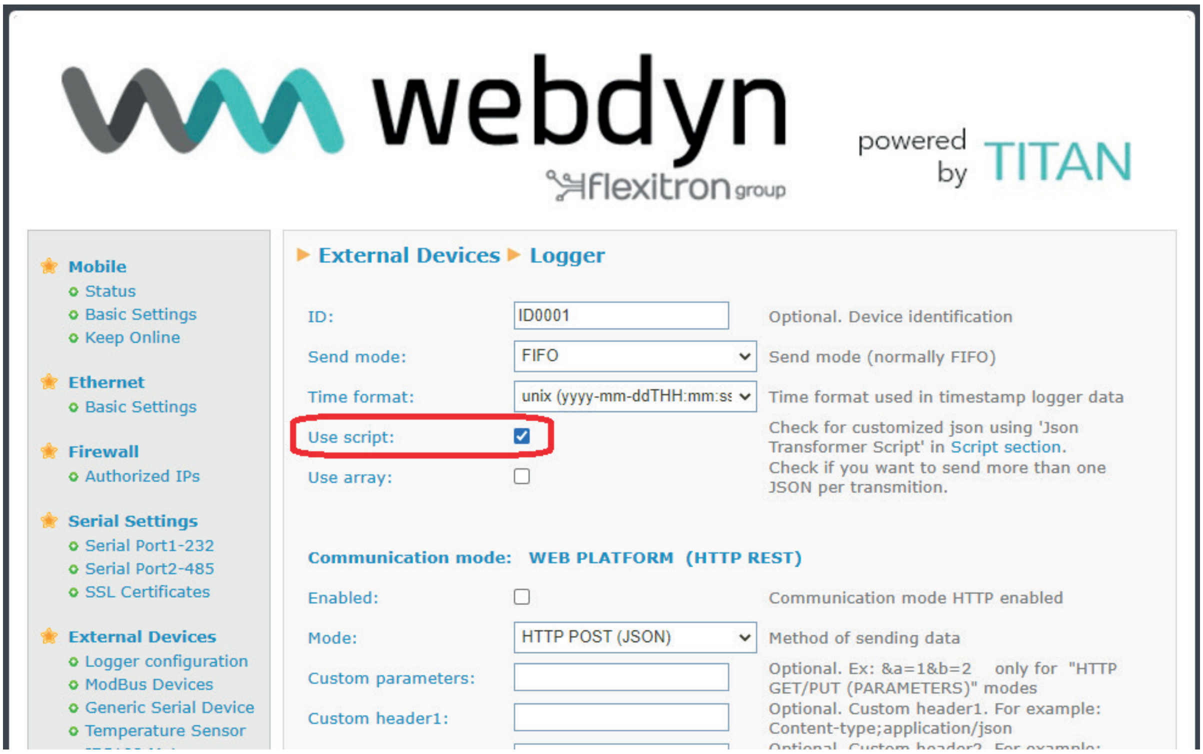
In that Application Note, the JSON format used after simply reading the Modbus registers would have the following format:

{"TYPE":"MODB","ID":"2","TS":"17/06/2017 17:01:05","IMEI":"357044060009633","P":"12345678","A":"2","ST":"10","N":"5","V":[10,11,12,0,0]}

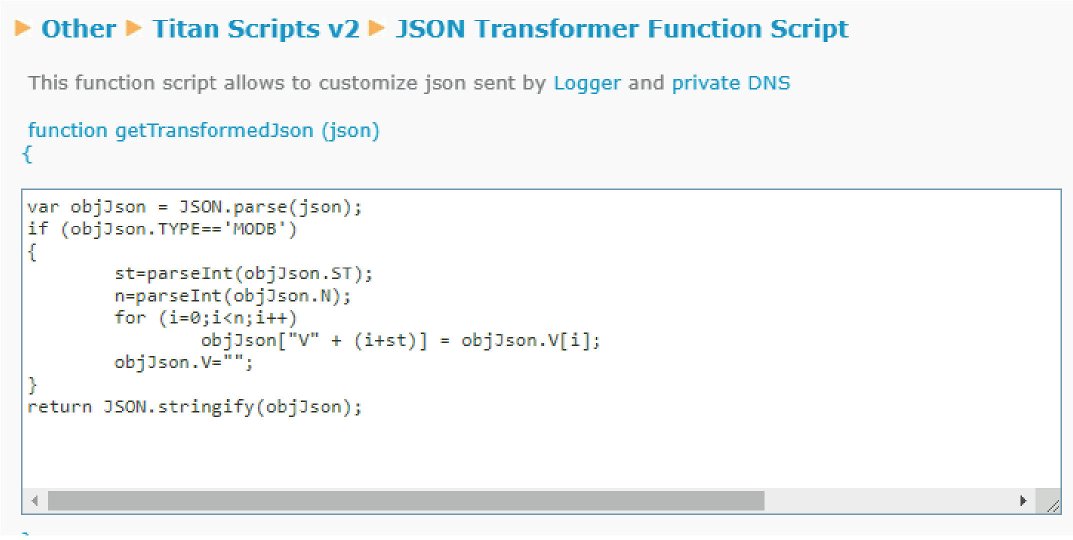
Now let's imagine that the data must be sent to an MQTT platform that uses the old data transfer format supported by MTX-Tunnel modems, where the Modbus registers are not sent as an array, but as independent variables:

{“TYPE”:”MODB”,”ID”:”2”,”TS”:”17/06/2017 17:01:05”,”IMEI”:”357044060009633”,”P”:”12345678”, “A”:”2”,”ST”:”10”,”N”:”5”,”V10”:10,”V11”:11,”V12”:12,”V13”:0,”V14”:0}

To transform the original JSON to the JSON needed by the platform, two things must be done. The first is to indicate of the LOGGER section that you intend to use the JSON Transformer Script function in the configuration screen. This is done in the “OTHER > Logger Configuration” configuration menu, check the “Use Script” box as shown in the following figure:



You can then access the “OTHER > Powered by Titan Scripts” section and the JSON format conversion script. This function receives a String containing the input JSON as a parameter in the "json" variable (i.e. the JSON with the default format for the TITAN-based device). The function will return a String in the format that we need in order to store it on the device and to subsequently send it to the platform.



In the current example, the script would be coded as follows:

//Create a JSON with the string passed as an argument (original JSON)

var objJson = JSON.parse(json);

//If the JSON is of MODB (Modbus) type, we will build the new result JSON

if (objJson.TYPE==’MODB’)

{

//We get the ST field, which contains the initial register number

st=parseInt(objJson.ST);

//We get the field N, which contains the number of records to be read

n=parseInt(objJson.N);

//We create the individual variables in the JSON

for (i=0;i<n;i++)

objJson[“V” + (i+st)] = objJson.V[i];

objJson.V=””;

}

//Return the String with the result, i.e. the formatted JSON

return JSON.stringify(objJson);

The original JSON:

{“TYPE”:”MODB”,”ID”:”2”,”TS”:”17/06/2017
17:01:05”,”IMEI”:”357044060009633”,”P”:”12345678”,”A”:”2”,”ST”:”10”,”N”:”5”,”V”:[10,11,12,0,0]}

Is converted to the following:

{“TYPE”:”MODB”,”ID”:”2”,”TS”:”17/06/2017 17:01:05”,”IMEI”:”357044060009633”,”P”:”12345678”
,”A”:”2”,“ST”:”10”,”N”:”5”,”V”:””,”V10”:10,”V11”:11,”V12”:12,”V13”:0,”V14”:0}