

TITAN

Nota de aplicación 57

JSON Transformer Script

JSON Transformer Script

1. Detalles del escenario

Los equipos basados en TITAN disponen de todas las funcionalidades típicas de un equipo 4G/3G/2G pero además cuentan con una serie de prestaciones adicionales que lo convierten en uno de los equipos con más prestaciones del mercado.

Una de las prestaciones adicionales es la capacidad de datalogger donde el equipo basado en TITAN almacena en su memoria no volátil distintos tipos de registros en un formato JSON. Estos registros almacenados pueden provenir de lecturas Modbus o de capturas de datos SERIE por sus puertos RS232 / RS485, de posiciones GPS, etc. Estos registros de tipo JSON son almacenados por el equipo basado en TITAN en su memoria interna no volátil para ser posteriormente enviados a una plataforma remota mediante comunicaciones tipo HTTP, HTTPS, MQTT, MQTTS, FTP, FTPS.

Como se indica, el equipo basado en TITAN almacena en su memoria interna dichos registros JSON en un formato predeterminado y propietario. En ocasiones esto puede ser un problema para la comunicación con aquellas plataformas que esperan recibir la información en un determinado formato (es decir, con un formato distinto de JSON al utilizado por el equipo basado en TITAN).

Mediante la funcionalidad JSON Transformer Function Script el equipo basado en TITAN permite realizar un formateo de cualquier objeto JSON antes de su almacenamiento en la memoria interna. Esta característica permite acomodar los objetos JSON con el formato adecuado para cada plataforma.

2. Objetos JSON predeterminados del equipo basado en TITAN

A continuación, se muestran los objetos JSON predeterminados que utiliza el equipo basado en TITAN para los diferentes tipos de envío.

a) Ejemplo de cambio de estado de una entrada digital

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

b) Ejemplo de envío periódico de todas las E/S

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

c) Ejemplo de envío de registros Modbus:

```
{“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) Ejemplo de envío de tramas de estado (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) Ejemplo de envío de tramas desde los SCRIPT:

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

f) Ejemplo de envío de tramas de capturas de puerto SERIE:

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

g) Ejemplo de envío de tramas de pérdidas de alimentación (para equipos con batería/supercap)

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

h) Ejemplo de envío de tramas de posición GPS (para equipos con 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. EJEMPLO 1: Cambio de formato del objeto JSON para envío a plataforma MQTT, la cual precisa de un formato JSON específico.

Imaginemos que disponemos del escenario ejemplo de lectura de registros Modbus y envío a plataforma MQTT como se indica en la Nota de Aplicación ANV6_32

En esa nota de aplicación el formato JSON utilizado tras una lectura simple de registros Modbus tendría el siguiente formato:

```
{“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]}
```

Pero imaginemos que los datos deben enviarse a una plataforma MQTT que necesita el siguiente formato de JSON.

```
{“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”}
```

Para poder transformar el JSON original al JSON indicado por la plataforma deben hacerse dos cosas. La primera es indicar en la pantalla de configuración de la sección de LOGGER que se pretende usar la función JSON Transformer Script. Esto se realiza desde la pantalla del menú de configuración “OTHER >Logger Configuration” activando la casilla “Use Script” como se indica en la figura siguiente:

The screenshot shows the webdyn configuration interface. The top header includes the webdyn logo, the Flexitron group logo, and the text 'powered by TITAN'. On the left is a sidebar menu with categories: Mobile (Status, Basic Settings, Keep Online), Ethernet (Basic Settings), Firewall (Authorized IPs), Serial Settings (Serial Port1-232, Serial Port2-485, SSL Certificates), and External Devices (Logger configuration, ModBus Devices, Generic Serial Device, Temperature Sensor). The main content area is titled 'External Devices > Logger'. It contains several configuration fields: 'ID' (text input with 'ID0001'), 'Send mode' (dropdown menu with 'FIFO'), 'Time format' (dropdown menu with 'unix (yyyy-mm-ddTHH:mm:ss)'), 'Use script' (checkbox, checked, highlighted with a red box), and 'Use array' (checkbox, unchecked). Below these is the 'Communication mode' section, set to 'WEB PLATFORM (HTTP REST)', with 'Enabled' (checkbox, unchecked) and 'Mode' (dropdown menu with 'HTTP POST (JSON)'). There are also fields for 'Custom parameters' and 'Custom header1'. Explanatory text for each field is provided on the right side of the form.

Después puede accederse a la sección “OTHER > Powered by Titan Scripts” para introducir el script de conversión de formato JSON. Esta función recibe como parámetro en la variable “json” un String con el JSON de entrada (es decir, el JSON con el formato estándar del equipo basado en TITAN. La función debe devolver como resultado un String con el formato que deseemos utilizar para el almacenamiento en el equipo y el envío posterior a la plataforma.

► Other ► Titan Scripts v2 ► JSON Transformer Function Script

This function script allows to customize json sent by [Logger](#) and [private DNS](#)

```
function getTransformedJson (json)
{
  const objJson = JSON.parse(json);
  const objJsonResult = {};

  if (objJson.TYPE=='MODB')
  {
    objJsonResult.IMEI = ""+objJson.IMEI;
    delete objJson["IMEI"];
    objJsonResult.data = objJson;
    objJsonResult.myCustomField1 = '123456789';
  }

  mtm.println("Result: " + JSON.stringify(objJsonResult));
  return JSON.stringify(objJsonResult);
}
```

Save JSON Transformer Script

Delete JSON Script

Load Example

Encrypt Script

Para el caso del ejemplo que nos ocupa el script tendría el siguiente código:

//Creamos un JSON con el string pasado por argumento (JSON original)

```
const objJson = JSON.parse(json);
```

//Creamos un JSON para devolver el resultado

```
const objJsonResult = {};
```

//Si el JSON es de tipo MODB (Modbus) construiremos el nuevo JSON de resultado

```
if (objJson.TYPE=='MODB')
```

```
{
```

```
  //Añadimos el campo IMEI al JSON del resultado.
```

```
  objJsonResult.IMEI = ""+objJson.IMEI;
```

```
  //Borramos el campo IMEI del JSON original
```

```
  delete objJson["IMEI"];
```

```
  //Añadimos el campo data y como valor asignamos el JSON original sin el IMEI
```

```
  objJsonResult.data = objJson;
```

```
  //Añadimos un campo y un valor personalizado
```

```
  objJsonResult.myCustomField1 = '123456789';
```

```
}
```

//Mostramos el resultado por la salida estándar.

```
mtx.println("Result: " + JSON.stringify(objJsonResult));
```

//Devolvemos el String con el resultado, es decir, el JSON formateado

```
return JSON.stringify(objJsonResult);
```

Es decir, el JSON original:

```
{"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]}
```

Queda transformado en el siguiente:

```
{"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. EJEMPLO 2: Cambio de formato del objeto JSON para envío a plataforma MQTT con compatibilidad formato MTX-Tunnel.

Imaginemos que disponemos del escenario ejemplo de lectura de registros Modbus y envío a plataforma MQTT como se indica en la Nota de Aplicación ANV6_32

En esa nota de aplicación el formato JSON utilizado tras una lectura simple de registros Modbus tendría el siguiente formato:

```
{"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]}
```

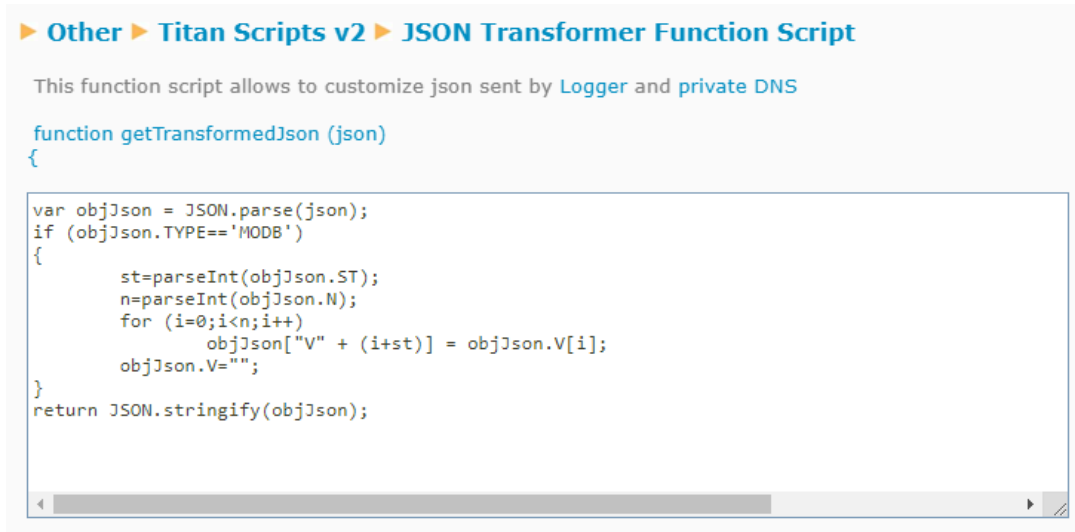
Pero imaginemos que los datos deben enviarse a una plataforma MQTT que utiliza el formato antiguo de envío de datos soportado por los módems MTX-Tunnel con el siguiente formato, donde los registros Modbus no se envían en un array, sino en variables independientes:

```
{"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"}
```

Para poder transformar el JSON original al JSON indicado por la plataforma deben hacerse dos cosas. La primera es indicar en la pantalla de configuración de la sección de **LOGGER** que se pretende usar la función **JSON Transformer Script**. Esto se realiza desde la pantalla del menú de configuración **"OTHER > Logger Configuration"** activando la casilla **"Use Script"** como se indica en la figura siguiente:

The screenshot shows the 'Logger Configuration' page in the webdyn interface. The page has a sidebar on the left with navigation links: Mobile (Status, Basic Settings, Keep Online), Ethernet (Basic Settings), Firewall (Authorized IPs), Serial Settings (Serial Port1-232, Serial Port2-485, SSL Certificates), and External Devices (Logger configuration, ModBus Devices, Generic Serial Device, Temperature Sensor). The main content area is titled 'External Devices > Logger'. It contains several configuration fields: ID (ID0001), Send mode (FIFO), Time format (unix (yyyy-mm-ddTHH:mm:ss)), Use script (checked), Use array (unchecked), Communication mode (WEB PLATFORM (HTTP REST)), Enabled (unchecked), Mode (HTTP POST (JSON)), Custom parameters, and Custom header1. The 'Use script' checkbox is highlighted with a red rectangle. The 'Communication mode' is set to 'WEB PLATFORM (HTTP REST)'. The 'Mode' is set to 'HTTP POST (JSON)'. The 'Custom parameters' and 'Custom header1' fields are empty.

Después puede accederse a la sección “OTHER > Powered by Titan Scripts” para introducir el script de conversión de formato JSON. Esta función recibe como parámetro en la variable “json” un String con el JSON de entrada (es decir, el JSON con el formato estándar del equipo basado en TITAN. La función debe devolver como resultado un String con el formato que deseemos utilizar para el almacenamiento en el equipo y el envío posterior a la plataforma.



Para el caso del ejemplo que nos ocupa el script tendría el siguiente código:

//Creamos un JSON con el string pasado por argumento (JSON original)

```
var objJson = JSON.parse(json);
```

//Si el JSON es de tipo MODB (Modbus) construiremos el nuevo JSON de resultado

```
if (objJson.TYPE=='MODB')
```

```
{
```

//Obtenemos el campo ST, donde está el número de registro inicial

```
st=parseInt(objJson.ST);
```

//Obtenemos el campo N, donde está el número de registros a leer

```
n=parseInt(objJson.N);
```

//Creamos todas las variables individuales en el JSON

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

```
    objJson["V" + (i+st)] = objJson.V[i];
```

```
    objJson.V="";
```

```
}
```

//Devolvemos el String con el resultado, es decir, el JSON formateado

```
return JSON.stringify(objJson);
```

Es decir, el JSON original:

```
{“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]}
```

Queda transformado en el siguiente:

```
{“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}
```