

TITAN

Application Note 62

Sending data to DEXMA platform

Sending data to DEXMA platform

1. Detalles del escenario

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

Una de las prestaciones adicionales es la capacidad de datalogger donde el router 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 router 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 router 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 defecto por el router Titan).

Mediante la funcionalidad “**JSON Transformer Function Script**” el router Titan permite realizar un formateo de cualquier objeto JSON antes de su almacenamiento en su memoria interna. Esta característica permite por tanto acomodar los objetos JSON con el formato adecuado para cada plataforma. Realmente no es necesario transformar los datos en un formato JSON, puede transformarse en cualquier formato String.

En esta nota de aplicación se desarrollará el ejemplo completo para el envío de datos a la conocida plataforma de Dexma (<https://www.dexma.com/what-is-dexma-platform/>) la cual requiere de un formato especial de los JSON enviados y de una serie de cabeceras.

En esta nota de aplicación en concreto vamos a suponer que se deben leer 2 registros de 2 PLCs con comunicaciones Modbus conectados a un Webdyn EasyTunnel por su puerto RS485.



En concreto se pretende lo siguiente: cada 1 minuto, el dispositivo WebDyn-Easytunnel debe leer los registros Modbus con dirección 40000 y 40001 del PLC1 y los registros 40000 y 40001 del PLC2. En ambos PLCs el registro 40000 corresponde con la temperatura medida y el registro 40001 con la humedad. Las lecturas realizadas deben almacenarse en la memoria interna no volátil del Webdyn EasyTunnel (en su datalogger) y éste debe enviar dichos datos leídos a la plataforma DEXMA siempre que sea posible (haya cobertura, conectividad IP, ...) La comunicación con los PLCs es a través de un bus RS485 con una configuración 9600,8,N,1

2. Configuración de la Mobile WAN

El Webdyn EasyTunnel debe comunicarse con la plataforma DEXMA a través de comunicaciones 4G/3G/2G, por lo que la sección “**Mobile=>Basic Settings**” debe configurarse apropiadamente en función de la tarjeta SIM utilizada.

Mobile Basic Settings

Mobile WAN	Enabled (IP active)	Enable Wireless WAN interface
Sim Mode	SIM1	Sim selection
SIM1 APN:	movistar.es	SIM Card 1 APN
SIM1 Username:	MOVISTAR	SIM Card 1 username
SIM1 Password:	*****	SIM Card 1 password
SIM1 Pin:		SIM Card 1 PIN
SIM1 Auth:	None	SIM card 1 authentication
SIM2 APN:		SIM Card 2 APN
SIM2 Username:		SIM Card 2 username
SIM2 Password:		SIM Card 2 password
SIM2 Pin:		SIM Card 2 PIN
SIM2 Auth:	Auto	SIM card 2 authentication
Network selection:	Auto (4G/3G/2G)	Network selection
DNS selection:	Get DNS from Operator	
DNS1:	8.8.8.8	Preferred DNS1
DNS2:	8.8.4.4	Preferred DNS2

3. Configuración del puerto serie RS485

La comunicación con los dos PLCs se realizará a través de puerto serie RS485 a 9600,8,N,1, por lo que debe configurarse la sección “**Serial Settings => Serial Port2-RS485**” ajustando los parámetros como se indica a continuación.

Mobile

Status

Basic Settings

Keep Online

Ethernet

Basic Settings

Firewall

Authorized IPs

Serial Settings

Serial Port1-RS232

Serial Port2-RS485

SSL Certificates

External Devices

Logger configuration

ModBus Devices

Generic Serial Device

Temperature Sensor

IEC102 Meter

GPS Receiver

Plugins

Generic

Other

AT Command

DynDns

Private DynDns

Sms control

Periodic Autoreset

Time Servers

Remote Console

Snmn

Serial Gateway

Com2 Settings

Baudrate:

9600

Baudrate of serial port

Data bits:

8

Number of data bit

Parity:

none

Parity

Stop bits:

1

Number of stop bits

Timeout ms:

50

msec without serial data before sending (default: 50)

Allow local embedded AT commands

Ex.: <MTXTUNNEL>AT</MTXTUNNEL>

Allow remote embedded AT commands

Ex.: <MTXTUNNELR>AT</MTXTUNNELR>

Allow incoming GSM call (CSD Data Call)

Only TCP Server and TCP Client functions or Nothing. 2G (CSD) network required.

Function: Nothing or used by External Device or Script

Function: Serial - IP Gateway (TCP Server)

TCP Local Port:

20011

Listening TCP Port (1 ... 65535)

Temporal client RS232

Wakeup

Temporal client time:

60

Seconds for temporal client

Temporal client Random

0

Seconds, Random time for temporal client Wakeup

SSL/TLS enabled

SSL/TLS Enabled (SSL Certs needed)

4. Configuración del Logger

El siguiente paso es configurar el datalogger interno del WebDyn EasyTunnel. Para ello debe acudir al menú **“External Devices => Logger configuration”**. La configuración debería ser similar a la que se muestra en la siguiente figura:

External Devices ▶ **Logger**

ID: Optional. Device identification

Send mode: Send mode (normally FIFO)

Time format: Time format used in timestamp logger data

Use script: ☒ Check for customized json using 'Json Transformer Script' in [Script section](#).

Use array: ☒ Check if you want to send more than one JSON per transmission.

Check date: ☐ Save data in Logger only if date has been set (check [Time Servers](#))

Communication mode: WEB PLATFORM (HTTP REST)

Enabled: ☒ Communication mode HTTP enabled

Mode: Method of sending data

Custom parameters: Optional. Ex: &a=1&b=2 only for "HTTP GET/PUT (PARAMETERS)" modes

Custom header1: Optional. Custom header1. For example: Content-type;application/json

Custom header2: Optional. Custom header2. For example: IDENTITY_KEY;YOUR_KEY

Custom header3: Optional. Custom header3.

Server: Destination URL. Example: www.mydomain.com/set.asp?data=

Server Username: Optional. Blank if no server authentication required

Server Password: Optional. Blank if no server authentication required

Cabe destacar los siguientes parámetros:

- “**Time format**” es necesario seleccionar formato “unix”, que es la soportada por la plataforma DEXMA.
- “**Use script**” es muy importante seleccionar esta casilla, pues al activarla se ejecutará el script que incorporaremos más adelante y que transformará el JSON estándar creado por el WebDyn-Easytunnel en un JSON con el formato necesario soportado por la plataforma DEXMA.
- “**Use array**” DEXMA permite el envío de datos en array, pues activamos también la casilla,
- “**Custom header1**” debe introducirse, tal y como se indica en <https://support.dexma.com/hc/en-gb/articles/360013772759-Using-the-insertion-API-to-introduce-data-in-a-gateway> el header “x-dexcell-sorce-token” que es distinto para cada dispositivo. En el caso de este ejemplo tiene el valor:

X-Dexcell-Source-Token;token123456789 (campos separados por “;”, no por “:”)

- “**Custom header2**” debe introducirse:

Content-Type;application/json (campos separados por “;”, no por “:”)

- “**Server**” aquí introduciremos la URL del servidor, que tendrá este formato:

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

donde mac-123456789 deberá ser substituido adecuadamente con el KEY obtenido en su plataforma DEXMA.

5. Configuración de la seccion Modbus

En esta sección de configuración “**External Devices => Modbus Devices**” configuraremos las lecturas Modbus que deben realizarse sobre los 2 PLCs

External Devices ▶ ModBus RTU / TCP

Enabled: ☒ Enable Modbus Devices

Serial Port: Serial Port 2 Select the connected serial port if needed

Logger: ☒ Check if logger must be used
Please, configure logger before using this option

SAVE CONFIG VIEW LOG

Dev. name / ID	Addr.	Command	Start @	Num word/bit	Reg Type	Period		
PLC1	1	0x04	40000	2	WORD	1	Del	Test
PLC2	2	0x04	40000	2	WORD	1	Del	Test

Device name / ID: PLC2 Insert the device name or ID

Address: 2 Modbus RTU address or IP:port address

Command: 0x04 Modbus read command

Start: 40000 Address of the first register

Number Words / Bits: 2 Words for command 0x03/0x04. Bits for 0x01/0x02

Reg Type: WORD Type of registers for command 0x03/0x04

Period: 1 Read period (minutes)

Debe habilitarse el servicio Modbus activando la casilla “**Enabled**”. Debe seleccionar “**Serial Port 2**”, pues la lectura se realizará a través del puerto RS485. Y también debe activarse la casilla “**Logger**”, pues los registros Modbus leídos de los PLCs deben almacenarse en el datalogger interno del Webdyn EasyTunnel.

También deben crearse dos dispositivos que en este ejemplo hemos llamado PLC1 y PLC2.

En “**address**” debe indicarse la dirección Modbus de cada uno de los PLCs, por lo que en PLC1 pondremos la dirección 1 y en PLC2 la dirección 2. El comando Modbus que utilizaremos para la lectura de los registros 40000 y 40001 será el 0x04, por lo que seleccionaremos 0x04 en el campo “**Command**”. El registro iniciar a leer en ambos PLCs es el registro 40000, pues indicaremos en el campo “**Start**” el valor 40000. Pretendemos leer 2 registros Modbus de cada PLC (el 40000 y 40001) por lo que en el campo “**Number Words**” indicaremos 2.

El tipo de registro “**Reg Type**” lo seleccionaremos con “WORD” y en “**Period**” seleccionaremos 1, pues queremos lecturas de los registros Modbus cada minuto.

6. Configuración del SCRIPT para adaptar el JSON generado por el Webdyn EasyTunnel con formato estándar al formato necesario para la correcta comunicación con la plataforma DEXAM

El formato de JSON estándar generado por el Webdyn EasyTunnel con los datos leídos y con el que los enviaría a una plataforma tendrá un formato como el que sigue:

```
[
{
  "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"
}
]
```

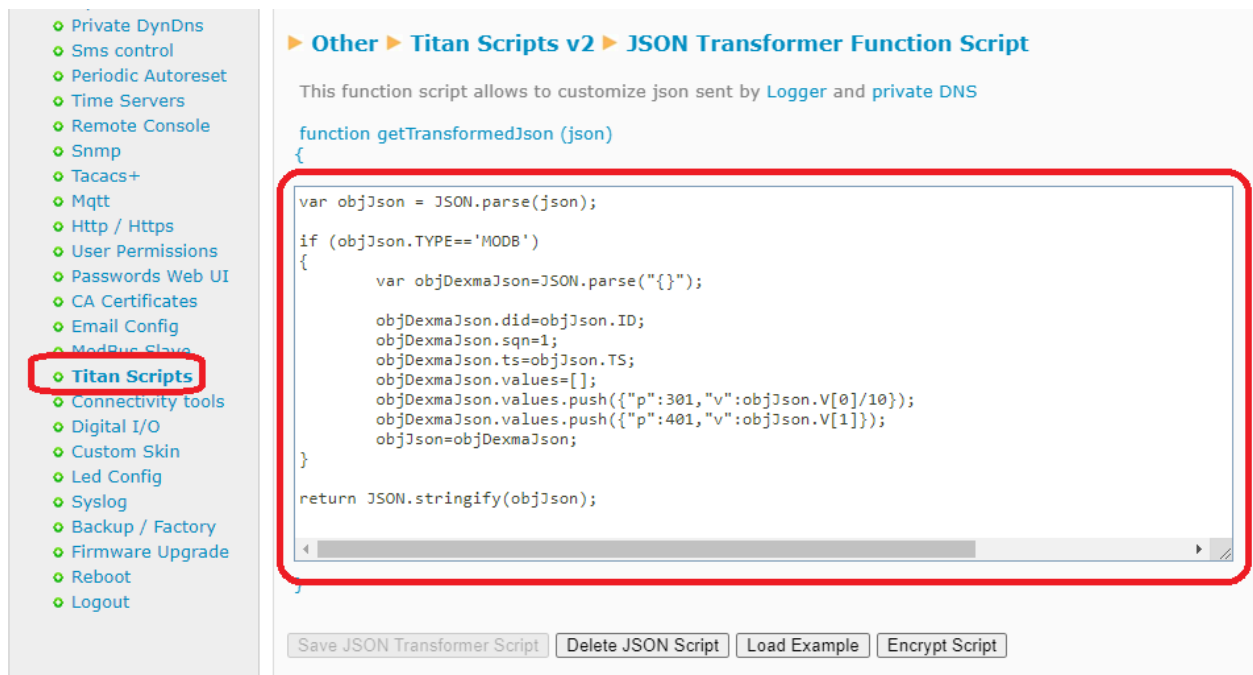
Pero el formato requerido por la plataforma DEXMA, tal y como se indica en sus manuales, (<https://support.dexma.com/hc/en-gb/articles/360013772759-Using-the-insertion-API-to-introduce-data-in-a-gateway>) debe ser:

```
[
{
  "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}]
}
]
```

Por lo tanto hay que crear un pequeño SCRIPT para adaptar los datos. Recuerde que este script de transformación se ejecutará porque anteriormente en la sección “**External Devices => Logger Configuration**” seleccionamos la casilla “**Use script**”

The screenshot shows the configuration interface of Webdyn EasyTunnel. On the left sidebar, under 'External Devices', 'Logger configuration' is selected. In the main configuration area, the 'Use script' checkbox is checked and highlighted with a red box. Below it, 'Use array' is also checked. 'Check date' is unchecked. The 'Communication mode' is set to 'WEB PLATFORM (HTTP REST)'. Below that, 'Enabled' is checked. The 'Mode' dropdown is set to 'HTTPS POST (JSON)'. To the right, there is a note: 'Check for customized json using 'Json Transformer Script' in Script section. Check if you want to send more than one JSON per transmission. Save data in Logger only if date has been set (check Time Servers)'.

Bien, para introducir el código del script de transformación de datos debe acudir al menú **“Other => Titan Scripts”**. Concretamente el script de transformación debe introducirse en la sección **“JSON Transforme Function Script”**



Como se indica en el manual, esta función para formatear el JSON es ejecutada ANTES de almacenar los datos de 1 lectura en el datalogger interno y se pasa como parámetro el JSON con el formato estándar del dispositivo, es decir, por ejemplo:

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

Bien, pues el script de transformación debe transformar ese anterior JSON en el siguiente para ser compatible con la plataforma DEXMA.

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

El código del SCRIPT sería el siguiente (con comentarios en añadidos).

```
//Creamos una variable objeto de tipo JSON, pues la variable “json” que recibe la //función como argumento es de tipo “String”.
```

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

//Si en el json original recibido, se indica que es un JSON de datos Modbus ...

if (objJson.TYPE=='MODB')

{

//Creamos un nuevo objeto JSON, donde almacenaremos los datos

//transformados

var objDexmaJson=JSON.parse("{}");

//Como campo did de la plataforma DEXMA, utilizaremos el campo ID del JSON

//original (es decir, PLC1 y PLC2)

objDexmaJson.did=objJson.ID;

//El campo sqn, siempre a 1.

objDexmaJson.sqn=1;

//Utilizamos el mismo timestamp que el JSON original

objDexmaJson.ts=objJson.TS;

//Añadimos al JSON un array donde almacenaremos los valores de

//temperatura y humedad.

objDexmaJson.values=[];

//Añadimos al array el parámetro 301 junto con el valor de la temperatura, que //corresponde con el primer registro leído, el 40000, almacenado en la posición //0 del array del json original (resaltado en verde en la página anterior) //Dividimos el valor por 10 para que tenga 1 //decimal.

objDexmaJson.values.push({"p":301,"v":objJson.V[0]/10});

//Añadimos al array el parámetro 401 junto con la humedad, que corresponde con el segundo registro leído, el 40001, almacenado en la posición 1 del array del json original (resaltado en rojo en la página anterior)

objDexmaJson.values.push({"p":401,"v":objJson.V[1]});

```
//Finalmente copiamos el JSON transformado, para devolverlo como resultado
//de la función de transformación.
objJson=objDexmaJson;
}
//Devolvemos el JSON en formato String como resultado de la función de transformación.
return JSON.stringify(objJson);
```

Una vez introducido el script y guardado basta con reiniciar el equipo. ya debería funcionar todo correctamente. En caso de problemas consulte los LOG del equipo y revise si necesita introducir algún certificado digital en la sección “Other => Ca Certificates”.

¿Más dudas?

Escríbanos sus consultas a iotsupport@mtxm2m.com