CLOUD OF THINGS
User Guide MQTT

Version: 2.0
Datum: 03.01.2019

LIFE IS FOR SHARING.

TABLE OF CONTENT

1. INTRODUCTION

This manual describes the MQTT interface to Deutsche Telekom's Cloud of Things. It is aimed at firmware and embedded developers wishing to implement the interface to Cloud of Things on an IoT device.

A special programming language is not used. In principle, any programming language that has access to the required libraries is conceivable.

Note: In case of questions, please contact: cloudofthings@telekom.de.

2. PREREQUISITES - DEVELOPMENT ENVIRONMENT

Access to the MQTT interface depends on specific conditions and libraries, e.g.:

- HTTP-enabled cellular modem
- TLS 1.1/1.2
- MQTT 3.1 client library (https://github.com/mqtt/mqtt.github.io/wiki/libraries)
- Cryptography library supported by
    - TLS
    - AES128
    - CBC
    - SHA256
    - DHE
    - "Extended Tiny Encryption Algorithm"
    - optional: PresharedKeys

3. WHAT IS MQTT?

MQTT (**MQ Telemetry Transport** or **Message Queue Telemetry Transport**) is an open message protocol for machine-to-machine communication (M2M), which enables the transmission of telemetry data in the form of messages between devices, despite high delays or limited networks. The appropriate devices range from sensors and actuators, mobile telephones, embedded systems in vehicles or laptops through to fully-developed computers. The protocol was developed by “Andy Stanford-Clark” from “IBM” and “Arlen Nipper” from “Cirrus Link Solutions”.

Since 2013, MQTT has been approved as a standard by the Organization for the Advancement of Structured Information Standards (OASIS) as an Internet of Things protocol. The MQTT protocol is also known under previous names such as “WebSphere MQTT” (WMQTT), “SCADA protocol” or “MQ Integrator SCADA Device Protocol” (MQIsdp). The Internet Assigned Numbers Authority (IANA) reserves the Ports 1883 and 8883 for MQTT. MQTT messages can be encoded with the TLS protocol.

MQTT is a client-server protocol. After establishing a connection, clients send the server (“broker”) messages with a topic that hierarchically classifies the message; for example, “livingroom/fridge/temperature” or “car/wheel/3/tirepressure”. Clients can subscribe to topics whereby the server then forwards the received messages to the corresponding subscribers. In this example, the main topics are “livingroom” and “car”.

Note: Messages always consist of a topic and the message content.

Messages are sent with a definable quality of service:

- *at most once* (the message is sent once and may not arrive if the connection is interrupted).
- *at least once* (the message is sent until its receipt is confirmed and the recipient may receive it several times)
- *exactly once* (this ensures that the message will only arrive just once, even if the connection is interrupted)

In addition, the retain flag can be used to instruct the server to cache the message on this topic. Clients that are new subscribers to this topic are the first to receive the cached message.
When establishing a connection, clients can define a "last will" in the form of a message. If the connection to the client is lost, this message is published and sent to the appropriate subscribers.
MQTT is normally used via TCP and has a 2byte header. The first byte contains the message type (4 bit), the quality of service (2 bit) and a retain flag.

The following message types exist:

- CONNECT
- CONNACK
- PUBLISH
- PUBACK
- PUBREC
- PUBREL
- PUBCOMP
- SUBSCRIBE
- SUBACK
- UNSUBSCRIBE
- UNSUBACK
- PINGREQ
- PINGRESP

- DISCONNECT

The second byte contains the length of the rest of the MQTT packet. This is followed by a variable part that contains the MQTT topic. Finally, the payload arrives, i.e. the data content that is published under the topic.

Detailed information on the structure of MQTT messages can be found here: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html.

3.1. MQTT WITH THE TELEKOM BROKER

3.1.1. PREPARE TENANT FOR MQTT

The tenant must be prepared for the MQTT.
In case you want to make use of MQTT in an existing or new tenant, it must be set up. To set up your tenant please contact: cloudofthings@telekom.de.

3.1.2. BROKER URL AND DEVICE CREDENTIALS

The MQTT client - the device - must establish a connection to the Telekom MQTT broker. There is a URL available for this: nb-iot.ram.m2m.telekom.com.

Each device must first perform a self-registration (bootstrap). It receives its device credentials and device ID as part of this self-registration. The user data for bootstrapping on the MQTT Broker can be requested at: registration.cloud-of-things@telekom.de.

The user data for the normal-level connection is communicated to the device as part of the self-registration process to be performed. A distinction is made here between individually self-registered devices and mass-registered devices. Both approaches are described below.

3.1.3. MQTT TOPICS

An MQTT device communicates with the Cloud of Things on two prescribed topics, one topic for sending and one topic for receiving data.

In order to receive data, after successfully connecting to the broker behind the broker URL, the device subscribes to the topic: *mr/ICCID*.

Please note that operations can be lost if you do not use QoS 1.

ICCID is a placeholder for the own *ICCID*, which is either determined by the software when accessing the SIM card or communicated to the software, for example via a configuration file.

For devices without a SIM-card it is necessary to use the MAC-adress of the sending network device.

The device sends data to the CoT via publish commands against the topic: *ms/ICCID*.

3.1.4. MQTT SECURITY BASICS

The following MQTT security settings are prescribed or implemented when communicating with the Telekom MQTT broker:

- MQTT via TCP
- TLS v1.1/1.2 (Port 8883) on the broker (HiveMQ)
- QoS 0, 1 and 2
- Clean Session true and false

- Last Will not supported
- Retained Flag not supported
- Wildcard subscribes not permitted
- Subscriptions to the publish topics are not permitted ("write-only")

Topics other than the two previously mentioned are not available due to security reasons and can also not be created or subscribed by the device.

Please note that the use of "CleanSession=TRUE" can result in missing operations to the device.

3.2. DEVICE SELF-REGISTRATION

First, the device generates a sufficient random and temporary password with 128Bit/16 Byte length. Example: it initializes the randomizer with the current TIME (seconds since 1970). A random function then generates a password with 128 bits (16 bytes). This is followed by the randomizer generation and executing a sleeptime of 0 to 60 seconds (so that the TIME cannot be determined from the CONNECT time).

The device connects to the above-mentioned bootstrap user data against the bootstrap URL.

After successful connection, it subscribes to the *sr/ICCID* topic. *ICCID* is a placeholder for the own *ICCID*, which is either determined by the software when accessing the SIM card or communicated to the software, for example via a configuration file.

The device publishes the random password on the **ss/ICCID topic**: **tempPassword**

Note: the main-topic is ss/ and not ms/ as provided for normal communication

The following information will provide a better understanding of how the bootstrap process works internally.

The bootstrap process queries and retrieves the credentials for the user **ICCID** from the CoT:

```
POST /devicecontrol/deviceCredentials
Content-Type : application/vnd.com.nsn.cumulocity.deviceCreden-
tials+json;ver=...
Authorization: Basic ...
{
  "id" : "ICCID"
}
```

As long as the **ICCID** has not been entered under *device management/device registration* in the UI of the platform and accepted with the Accept button, the bootstrap process will also not be informed of any credentials. As soon as the **ICCID** has been accepted in the user interface, the bootstrap process receives the credentials and sends them encoded to the device using the Extended Tiny Encryption Algorithm: **3rasfst4swfa**.

This encoded result is published under the *sr/ICCID* topic. Since the device is subscribed to this topic, it receives the encoded payload. Using the random password, the device can now decode the payload.

Please note that the encrypted string will be padded to the nearest multiple of eight. Various encrypted clocks have to be connected and the pad-bytes have to be removed afterwards.

The device saves the password at a suitable location. Its user name has the following format: **ICCID**.

As from this point in time, all further connections are made using these credentials via the broker URL.

3.3. MASS REGISTRATION OF DEVICES

The mass registration of devices is used to simplify the registration of a large number of devices. The devices are provisioned with a username and a password for logging on to the broker URL at the manufacturer during manufacture or initialization. There are small differences between the registration for just MQTT devices and NB-IoT devices:

To register the devices on the platform, the manufacturer must create or download the template from Cloud of Things a CSV file with the following content and structure and make it available to Telekom platform administration (you can download an example with a click on "download template"):

MQTT

The headlines for the **MQTT** CSV file are „**ID**", „**Credentials**", „**Type**", „**Name**", „**ICCID**", „**Shell**", „**com_Cumulocity_model_Agent**".

The lines are used as follows:

- **ID**: ID of the device as referencing within Cloud of Things.
- **Name**: Name of the device for the display within the device management.
- **Credentials**: Password that is used by the device for the registration in Cloud of Things.
- **Type**: Type of the device. If the device should be connected via MQTT the type "mqtt" have to be registered.
- **ICCID**: Is used as the device's registration name for Cloud of Things.
- **Shell**: Is a flag (0 or 1) whether to allow the sending of operations to a device or not (1=yes).
- **Com_cumulocity_model_Agent**: necessary to display the device within the inventory. The curly brackets are empty "{}".

An appropriate example could be:

```
ID;NAME;Credentials;TYPE;ICCID;Shell;com_cumulocity_model_Agent
8949020000130573726 7;Geraet1;LF2PWJoLG1Fz;mqtt;89490200001305737267;1;{}
```

NB-IoT

The headlines for the **MQTT** CSV file are „**ID**", „**Credentials**", „**Type**", „**Name**", „**ICCID**", „**Shell**", „**com_Cumulocity_model_Agent**",„**c8y_Mobile.imsi**", „**c8y_Mobile.imei**".

The columns are used as follows:

- **ID**: ID of the device as referencing within Cloud of Things (always IMSI for Nb-IoT)
- **Name**: Name of the device for the display within the device management.
- **Credentials**: Password that is used by the device for the registration in Cloud of Things (have to be 8 letters and UTF-8 conform).
- **Type**: Type of the device. If the device should be connected via MQTT the type "nbiot" have to be registered.
- **ICCID**: Is used as the device's registration name for Cloud of Things.
- **Shell**: Is a flag (0 or 1) whether to allow the sending of operations to a device or not (1=yes).
- **Com_cumulocity_model_Agent**: necessary to display the device within the inventory. The curly brackets are empty "{}".
- **C8y_Mobile .imsi:** IMSI that is assinged to your SIM card. You get the information with your SIM cards.
- **C8y_Mobile.imei**: IMEI of the device.

An appropriate example could look like this:

```
ID;CREDENTIALS;TYPE;NAME;ICCID;SHELL;com_cumulocity_model_Agent;c8y_Mobile.
imsi;c8y_Mobile.imei
123456789123456;LF2PWJOL;nbiot;Gerät1;123456789123456;1;{};123456789123456;
987654321098765
```

A line in the file must be available for every new device to be registered. The file is read on the platform by Telekom administration and the corresponding device accesses are created with the specified passwords on the tenant and the MQTT Connector.

Note: After the upload of the CSV in some cases it can take up to 10 minutes until the connection is successful.

The pre-registered devices do **not** perform self-registration but log in immediately with the pre-provisioned credentials to the broker URL.

3.4. DELETING A DEVICE

As part of a device's life cycle, it may be necessary to delete its credentials and its ManagedObject in the tenant. The device can establish this by not being able to connect to the broker URLs device credentials it communicated, although this was previously still possible.

In this case, the broker rejects the connection as unauthorized.

4	Connection Refused, bad user name or password
5	Connection Refused, not authorized

It is now the task of the device to perform self-registration again. The exact procedure of self-registration is described above.

The parent device (device_Parent) must never be deleted. Additionally, the device credentials of the parent device (device_Parent) must not be deleted or deactivated.

4. WHAT IS SMARTREST?

4.1. GENERAL INFORMATION

Normally, a device uses the Cloud of Things via a REST interface and sends POST, GET and PUT commands with JSON structures via HTTP to the body (payload). This method is proven and safe, but has the "disadvantage" that a lot of data must be sent back and forth, thus greatly increasing the transfer volume of the device on the SIM card.

MQTT was developed as a narrowband protocol for the requirements of devices in the Industry 4.0 environment, so it should keep the data volume to be transmitted as low as possible.

SmartRest was developed since this claim is not compatible with "communicative" REST/JSON. SmartRest is a method to reduce REST/JSON content to its essential data part and to save overheads in transmission. This occurs through the agreement between so-called SmartRest templates. A SmartRest template describes how to interpret sent data and translate it into JSON/REST.

In addition, the device must also define so-called response templates. These templates match to an expected response of the CoT in JSON/REST format. Here the actual net data from the JSON are extracted and formatted by the platform on the basis of the response template specification.

A device defines its own set of templates once and addresses this template set via a unique ID. The definition of a template sentence is called registration. Registration is carried out either by the device itself or by the admin user for all devices of a certain type.

The device manufacturer must develop and assemble this set of templates individually for the device. He has to decide if the device will do the template registration first after self-registration or if the admin user has to register the supplied template collection in advance on the CoT.

A device sends data by specifying which template (unique template ID) it wants to use and then sends the associated net data as a CSV. It receives a response from the CoT in the form of a response template ID and the corresponding net response data.

4.2. CREATING SMARTREST TEMPLATES

In order to communicate with SmartRest, there must always be a collection of pre-registered templates. There are both request and response templates.

A template collection must be registered in the platform. Registration can also be carried out by the device itself via MQTT. However, the direct REST interface of the respective tenant is recommended for this since a template collection for registration must always be sent completely in one command. This would lead to a very large MQTT payload.

Alternatively, of course, the tenant administrator can also register the template collection(s) for the devices on the platform.

4.2.1. REQUEST TEMPLATES

Request templates have the following structure:

```
"10,<ID>,<METHOD>,<URI>,<CONTENT>,<ACCEPT>,<PLACEHOLDER>,<PARAMS>,<TEMPL
ATE>"
```

Explanatory notes:

10:	fixed ID, which tags a request template
<ID>:	a unique number that is used to address the template
<METHOD>:	the HTTP method used, POST - GET- PUT - DELETE
<URI>:	address that is addressed in the platform

<CONTENT>:	the content type to be sent
<ACCEPT>:	the accepted/expected content type of the response
<PLACEHOLDER>:	the place holder characters for parameters to be transferred
<PARAMS>:	a list of all parameter types
<TEMPLATE>:	the actual template with fixed parts and placeholders

Here is an example of a template description that changes the status of an operation:

```
10,211,PUT,/devicecontrol/operations/%%,application/vnd.com.nsn.cumuloci
ty.operation+json,application/vnd.com.nsn.cumulocity.operation+json,%%,U
NSIGNED STRING,"{""status"": ""%%""}"
```

And another example that generates a measurement in the platform:

```
10,410,POST,/measurement/measurements,application/vnd.com.nsn.cumulocity
.measurement+json,application/vnd.com.nsn.cumulocity.measurement+json,&&
,STRING STRING NUMBER STRING STRING STRING STRING,"{ ""&&"": {""&&"": {
""value"": &&, ""unit"": ""&&"" } }, ""time"": ""&&"", ""source"": {
""id"": ""&&"" }, ""type"": ""&&"" }"
```

4.2.2. RESPONSE TEMPLATES

Response templates have the following structure:

"11,<ID>,<BASE>,<COND>,<VALUE>[,<VALUE>]"

Explanatory notes

11:	fixed ID, which tags a request template
<ID>:	a unique number that is used to address the template
<BASE>:	the base JSON path showing the object from which the values are extracted
<COND>:	the conditional JSON path, used here as a filter
<VALUE>[,<VALUE>]:	JSON path showing exactly the value to be extracted

Here is an example that formats measurements read by the platform:

```
11,401,$.measurements,"$.Spannung","$.time","$.Spannung.0.unit","$.Spann
ung.0.value"
```

Another example that formats pending operations of a given ManagedObject:

```
11,201,$.operations,"$.c8y_Command","$.id","$.c8y_Command.text"
```

4.2.3. TEMPLATE COLLECTIONS

All templates of a device type to be used are summarized in a template collection below a so-called *X-ID* and registered in a bundled form. The X-ID is also used to indicate which registered template collection should be used. Template collections for an X-ID cannot be changed, only deleted and recreated.

A X-ID consists of digits and letters, thus at least one letter must be used. Only numeric X-IDs are not allowed.

Registration of a template collection is carried out by a device itself via MQTT payload or the tenant admin does this by means of a REST POST:

```
POST https://tenant.platform/s
```

The basic Auth is transferred to an HTTP header field. The template collection with the associated X-ID is in the body of the REST call. The X-ID must be specified in the first line together with the fixed ID **15**. All templates to be registered then follow line by line.

An example is given below:

```
15,X-ID

10,100,POST,/inventory/managedObjects,application/vnd.com.nsn.cumulocity
.managedObject+json,application/vnd.com.nsn.cumulocity.managedObject+jso
n,,,"{""name"":""Test De-
vice"",""type"":""com_example_TestDevice"",""c8y_IsDevice"":{}}"

10,101,POST,/inventory/managedObject

.

.

.
```

If the device registers the template itself via MQTT, the entire template collection must be transferred to an MQTT payload. It is essential to first carry out the fixed ID **15** and specification of the X-ID. Thereafter, the individual template registrations are each separated by a line separator character:

```
15,X-
ID\n10,100,POST,/inventory/managedObjects,application/vnd.com.nsn.cumulo
city.managedObject+json,application/vnd.com.nsn.cumulocity.managedObject
+json,,,"{""name"":""Test De-
vice"",""type"":""com_example_TestDevice"",""c8y_IsDevice"":{}}"\n10,101
,POST,/inventory/managedObject\n.....
```

The positive result of the platform after template registration is:

```
20,Templatesammlung-MO-ID
```

The template collection has been created in the platform under *template collection MO ID*.

Changes to a registered template collection are not possible. The template collection can, however, be deleted by deleting it on the *template collection MO ID* in the ManagedObjects and then recreating it.

For this purpose, each template collection should contain a template for deleting the collection:

```
10,TID,DELETE,/inventory/managedObjects/&&,,,&&,UNSIGNED,
```

The MQTT payload for retrieving this template is:

```
15,X-ID\nTID,Templatesammlung-MO-ID
```

Subsequently, the corrected or supplemented template collection can be re-registered as described above.

4.3. SELF-CREATION OF THE DEVICE

A device must create itself once in the platform after receiving its credentials. To do this, it first logs in to the broker URL with its username and password.

All further steps are carry out by the publish function of MQTT payload to the **ms/ICCID** topic**.** In the payload, the fixed ID **15** and specification of which X-ID should be addressed always occur at the beginning. This is followed by a line separator character **\n** and indication of the template ID to be addressed.

Finally, the transfer parameters are listed, separated by commas.

The device first checks, if there already exists a registration for its **ICCID** or serial number (hardware replacement). In the process, it uses, for example, the following template:

```
10,TID,GET,/identity/externalIds/c8y_Serial/%%,,application/vnd.com.nsn.
cumulocity.externalId+json,%%,STRING,
```

In order to receive a suitable response from the platform, a response template must exist:

```
11,RTID,,"$.externalId","$.externalId","$.managedObject.id"
```

The device sends the following message as payload via MQTT:

```
15,X-ID\nTID,ICCID
```

Here it is necessary to enter the own serial number. If a registration for this serial number already exists, a response - as MQTT Publish to the topic *mr/ICCID* - is received with the following content:

```
87,1,X-ID\nRTID,2,ICCID,MO-ID
```

The device should now use this ManagedObject ID as its own MO ID for all further actions.

If there is no registration, an error message is

```
50,2,404,Not Found
```

returned via *mr/CCID.* What that means for the device is that, now it must first create itself as a ManagedObject.

There is a template registration for this describing the MO and its basic properties:

```
10,TID,POST,/inventory/managedObjects,application/vnd.com.nsn.cumulocity
.managedObject+json,application/vnd.com.nsn.cumulocity.managedObject+jso
n,&&,STRING
STRING,"{""name"":""&&"",""type"":""&&"",""c8y_IsDevice"":{}}""":{}"":{}
,""com_cumulocity_model_Agent"":{}}"
```

In order to receive the ID of the MO as a response, a suitable response template has to exist:

```
11,RTID,,"$.c8y_IsDevice","$.id"
```

The device sends an MQTT message for self-creation with the following payload:

```
15,X-ID\nTID,ICCID,DeviceName
```

The platform reports a RC and the ManagedObjectId, when the creation of the MO has been processed:

```
87,1,X-ID\nRTID,2,MO-ID
```

The device is now registering its ICCID or any other clear serial no. like its MAC-Id in *globalIds* as *externalId*. There is a template for this:

```
10,TID,POST,/identity/globalIds/%%/externalIds,application/vnd.com.nsn.c
umuloci-
ty.externalId+json,application/vnd.com.nsn.cumulocity.externalId+json,%%
,STRING STRING,"{ ""type"" : ""c8y_Serial"", ""externalId"" : ""%%"" }"
```

The device sends an MQTT message with the following payload for self-creation in the external IDs:

```
15,X-ID\nTID,MO-ID,ICCID
```

Note: without the registration of the ICCID/serial no. in the *globalIds* the device cannot automated receive operations.

The device then updates its ManagedObject and enters device-specific data such as ICCID, IMSI, software and firmware releases, its type and hardware serial number (see external IDs):

```
10,TID,PUT,/inventory/managedObjects/&&,application/vnd.com.nsn.cumuloci
ty.managedObject+json,application/vnd.com.nsn.cumulocity.managedObject+j
son,&&,UNSIGNED STRING STRING STRING STRING STRING STRING STRING STRING
STRING
STRING,"{""c8y_Hardware"":{""model"":""&&"",""revision"":""&&"",""serial
Num-
ber"":""&&""},""c8y_SoftwareList"":[{""name"":""Application"",""version"
":""&&"",""url"":""none""},{""name"":""Bootloader"",""version"":""&&"","
"url"":""none""},{""name"":""Bluetooth"",""version"":""&&"",""url"":""no
ne""},{""name"":""Modem"",""version"":""&&"",""url"":""none""}],""c8y_Mo
bile"":{""imei"":""&&"",""iccid"":""&&"",""imsi"":""&&""}}"
```

The device sends an MQTT message with the following payload to update the ManagedObject:

```
15,X-ID\nTID, 300,MO-
Id,Model,RevNr,HardwareSerialNumber,AppVersion,BLVersion,BTVersion,ModVe
rsion,IMEI,ICCID,IMSI
```

Then the device should make a further update of its ManagedObject and enter its supported operations. These are of course device specific:

```
10,TID,PUT,/inventory/managedObjects/&&,application/vnd.com.nsn.cumuloci
ty.managedObject+json,application/vnd.com.nsn.cumulocity.managedObject+j
son,&&,UNSIGNED,"{""c8y_SupportedOperations"": [
""c8y_Configuration"",""c8y_SendConfiguration"",""c8y_DownloadConfigFile
"",""c8y_UploadConfigFile"",""c8y_Software"",""c8y_SoftwareList"",""c8y_
Firm-
ware"",""c8y_FirmwareList"",""c8y_SystemCommand"",""c8y_Command"",""c8y_
Restart"" ] }"
```

The device sends an MQTT message with the following payload to update the MO:

```
15,X-ID\nTID,MO-Id
```

The device now exists on the platform following correct creation and can now send measurements, alarms, etc.

4.4. SENDING MEASUREMENTS

Once a device has been completely created on the platform, it can begin to send measurements, events, and alarms.

Templates must also be registered in the template collection for these sending processes. Below is an example of a template for sending a measurement and its use in the MQTT payload.

Assuming template registration exists:

```
10,TID,POST,/measurement/measurements,application/vnd.com.nsn.cumulocity
.measurement+json,application/vnd.com.nsn.cumulocity.measurement+json,&&
,STRING STRING NUMBER STRING STRING STRING STRING,"{ ""&&"": {""&&"": {
""value"": &&, ""unit"": ""&&"" } }, ""time"": ""&&"", ""source"": {
""id"": ""&&"" }, ""type"": ""&&"" }"
```

Now the device receives a measured value from a temperature sensor and publishes it as a payload with MQTT on ms/ICCID:

```
15,X-ID\nTID,Temperatur,0,395.3,°C,2017-09-
14T12:37:00.000,MO_ID,c8y_TemperatureMeasurement
```

What is important here is the timestamp, which communicates the exact time of the measurement. The first parameter denotes the entire measured value, as it is then listed in the user interface of the platform.

The second parameter "0" is the designation for the measured value series or the curve identifier. For example, this could be used to summarize temperature values from different sensors in a measured value graph and clearly identify their individual curves.

This is followed by the actual value measured, in this case 395.3. Then the unit, here °C. The timestamp of the measurement is followed by indication of your own ManagedObject so that the measured value in the platform references the correct device. Finally, the type of the measured value is given. Fixed platform constants or own constants can be used here

The result of the measured value transfer is published to the device as an MQTT message on the *mr/ICCID* topic.

4.5. SENDING EVENTS

Templates must also be registered in the template collection for sending events. Below is an example of a template for sending the current geolocation of the device and its use in the MQTT payload.

```
10,TID,POST,/event/events,application/vnd.com.nsn.cumulocity.event+json,
application/vnd.com.nsn.cumulocity.event+json,&&,STRING STRING NUMBER
NUMBER NUMBER,"{ ""time"": ""&&"", ""type"": ""c8y_LocationUpdate"",
""text"": ""Position changed."", ""source"": { ""id"": ""&&"" },
""c8y_Position"": { ""lng"": ""&&"", ""alt"": ""&&"", ""lat"": ""&&"" }
}"
```

The device publishes its current geolocation as PositionUpdateEvent via the payload of an MQTT message to **ms/ICCID**:

```
15,X-ID\nTID,2017-09-14T12:37:00.000,MO-ID,7.1424255,77,50.746994
```

The first parameter returned here is the X-ID of the template collection. This is followed by a line separator and then the template ID followed by a timestamp with the time of the event.

After the reference to the own ManagedObject, this is completed by the 3 parameters longitude, altitude and latitude.

In the same way any kind of event can be sent. Events are freely configurable in the platform. Minimal parameters must however be indicated:

<time>:	"2011-09-06T12:03:27.845+02:00"
<type>:	EventType
<text>:	""free text about the event"
<source>:	{ "id" : *MO-ID* }

4.6. SENDING ALARMS

Alarms are events that require particular attention from users or even trigger additional alarm mechanisms such as SMS dispatch.

Sending alarms also requires prior registration of appropriate templates:

```
10,TID,POST,/alarm/alarms,application/vnd.com.nsn.cumulocity.alarm+json,
application/vnd.com.nsn.cumulocity.alarm+json,&&,STRING STRING STRING
STRING STRING STRING,"{ ""type"": ""&&"", ""time"": ""&&"", ""text"":
""&&"", ""status"": ""&&"", ""severity"": ""&&"", ""source"": { ""id"":
""&&"" } }"
```

The device sends an alarm by publishing an MQTT message with the appropriate payload on **ms/ICCID**:

```
15,X-ID\nTID,Kesseltemperatur,2017-09-14T12:37:00.000,"Kesseltemperatur
stark erhoeht",ACTIVE,MAJOR,MO-ID
```

Here, too, specification of the X-ID is followed by a line separator character after indicating the fixed ID 15. After specifying the template ID, the alarm type is indicated (boiler temperature), then the timestamp of the alarm time and an alarm text.

Then the alarm status (ACTIVE, ACKNOWLEDGED or CLEARED) and its importance (CRITICAL, MAJOR, MINOR or WARNING), always in capital letters.

The own ManagedObject ID comes at the end so that the referencing device can be found.

4.7. CREATING AND LINKING A CHILD DEVICE

Devices, particularly gateway devices, can have child devices. A device must create its child devices on the platform itself and link them to its ManagedObject.

Two templates should be available in the template collection of the device for this purpose. Template A creates a child-device as a ManagedObject on the platform. Please note that the fragment **"c8y_IsDevice: {}"** is not set so that the child device is not visible in the device list:

```
10,TID1,POST,/inventory/managedObjects,application/vnd.com.nsn.cumulo
city.managedObject+json,application/vnd.com.nsn.cumulocity.managedObj
ect+json,&&,STRING,"{""name"":""&&"",""com_IsChildDevice"":""{}""}"
```

A response template is available to receive the new MO ID of the child device as a response from the platform:

```
11,RTID,,"$.com_IsChildDevice","$.id"
```

Template B creates the link from the ManagedObject of the device to the ManagedObject of the child device:

```
10,TID2,POST,/inventory/managedObjects/%%/childDevices,application/vn
d.com.nsn.cumulocity.managedObjectReference+json, applica-
tion/vnd.com.nsn.cumulocity.managedObjectReference+json,%%,STRING
STRING,"{ ""managedObject"" : { ""self"" :
""https://.../inventory/managedObjects/%%"" } }"
```

The device first creates a child device per MQTT:

```
15,X-ID\nTID1,"Child-Device-Name"
```

The platform send the following as response:

```
87,1,X-ID\nRTID,2,Child-MO-ID
```

Now the device has to link the MO ID of the child device to its own ManagedObject. To do this, it retrieves template B with the following payload:

```
15,X-ID\nTID2,MO-ID,Child-MO-ID
```

The child device is now linked to the device and can be selected and viewed in the UI of the platform under "Child Devices". In addition, further properties of the child device can now be stored and changed on its ManagedObject. It behaves exactly like a device ManagedObject.

4.8. OPERATIONS

Operations are instructions that are sent via the platform to a device. The device must interpret the operation, execute it and report back results or a status. Furthermore, the device can report that an operation is executed by sending an appropriate status update.

Operations are published to the device via the *mr/ICCID* topic. Since the device is subscribed to this topic, it automatically receives new operations.

As soon as an operation is configured for a device it will be sent at the next connection. It is not possible to cancel an operation once it is drawn up. All drawn up operations will be sent by the next connection to the broker.

The binding prerequisite for each template collection is that a GET template is registered under template ID **500**, which picks up new operations from the platform. The device does not use this template, complete control is provided by the MQTT Connector.

Note: In the case of various existing template-collections the template with the ID 500 must exist in exact the same template-collection, which the device during the first action has used. For instance, the check of the own **ICCID**/serial no. in the ***externalIds***.. It is the same for all response-templates for operations.

```
10,500,GET,/devicecontrol/operations/%%,,application/vnd.com.nsn.cumuloc
ity.operation+json,%%,UNSIGNED,
```

Please note that the device cannot receive automated operations if this template is not registered with the ID 500. Furthermore, the first request must be send with a template from the template collection in which this template is registered.

In addition, the device establishes which format it expects information on certain operations on the basis of response templates in its template collection:

```
11,TID,,"@.,$.operations,"$.c8y_Command","@.id","@.c8y_Command.text"
```

This response template can, for example, generate the following content as a response:

```
87,1,X-ID\nTID,2,OP-ID,ls -l
```

An operation with the following JSON is created on the platform:

```
{
        "creationTime": "2017-09-14T13:01:27.740+02:00",
        "deviceId": MO-ID,
        "deviceName": "DeviceName",
        "id": OP-ID,
        "status": "PENDING"
        "c8y_Command": {
                "result": " ",
                "syntax": null,
                "text": "ls -l"
        },
        "description": "Execute shell command"
}
```

The device must now execute the operation - in this case, the shell command "ls -l" and report back any results and the result status (SUCCESSFUL or FAILED).

Again, template registrations must be defined in the template collection. The following example shows a template which can be used to report back the result of the above operation and the updated status:

```
10,TID,PUT,/devicecontrol/operations/%%,application/vnd.com.nsn.cumuloci
ty.operation+json,application/vnd.com.nsn.cumulocity.operation+json,%%,U
NSIGNED STRING STRING STRING,"{""status"": ""%%"", ""c8y_Command"":
{""result"": ""%%"",""syntax"": """",""text"": ""%%""}}"
```

The device publishes an MQTT message with the following payload on the **ms/ICCID** topic:

```
15,dbeTest-01\nTID,OP-ID,"SUCCESSFUL","total 28496 -rw-r--r-- 1 user
staff 474559 10 Mai 10:08 D-Route5.gpx drwxr-xr-x 6 user staff 192 18
Sep 10:11 Distribute -rw-r--r-- 1 user staff 6959276 8 Aug 11:59 GTECli-
ent.jar","ls -l"
```

The most important feature for all actions is the OperationID (OP ID), which was transmitted together with the operation. With the OP ID, operations are clearly referenced on the platform.

4.9. HANDLING OF OPERATIONS FOR CHILD DEVICES

Operations that are configured for an assigned child device on the platform are also published to the device with the topic *mr/ICCID*.

The MO-ID in the response is equal to the MO-ID of the child device.

Hence, the parent device (gateway) has to know the MO-IDs of its assigned child devices to be able to forward the operations to its child devices. The responses and status updates of the operations must be send or processed by the parent device to the platform for its child devices.

4.10. RETRIEVE OWN TENANT

If the connected tenant is needed by a self-registered device, for instance for a firmware download via HTTPS connection, this can be done by two templates:

```
10,800,GET,/user/currentUser,,,,,
11,801,,"$.userName","$.self"
```

When the device sends the SR request

```
800
```

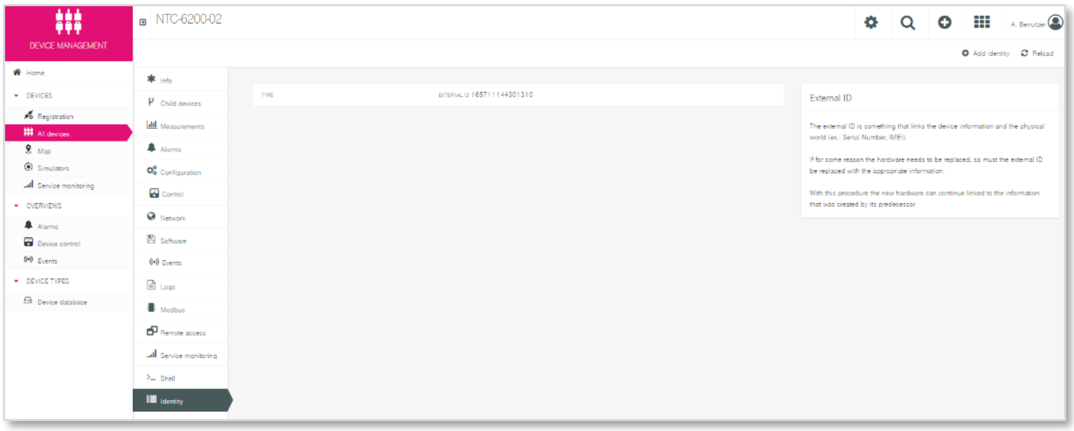
it gets the answer:

```
801,1,http://<yoururl>.ram.m2m.telekom.com/currentUser
```

Through this answer the current tenant can be determined.

4.11. CHANGING HARDWARE

Over the time, there will be hardware changes due to defects. Nevertheless, the already collected data should stay and should be processes as usual.
This is guaranteed by self-registration of the replacement device or a new registration of a so-called "identity" in CoT.
The creation of an identity has to be done in the device management application in CoT. Choose "all devices" and select the appropriate device. In the device menu select the "identity" tab and "add identity." (E.g.: serial number)



TYPE EXTERNAL ID 165711144301310

External ID

The external ID is something that links the device information and the physical world (ex.: Serial Number, IMEI).

If for some reason the hardware needs to be replaced, so must the external ID be replaced with the appropriate information.

With this procedure the new hardware can continue linked to the information that was created by its predecessor

Note that it is necessary to use the same type as of the old identity. Only a new "external_ID" must be created.

The next step is to delete the old device credentials in the tab "device credentials". Now, the new device starts a self-registration and it **must** use the deviceuser/ICCID of the changed device.
New credentials for this user can be created with the SelfReg and the old MO can use these new credentials, it is just essential that the DeviceUser has the same name.

4.12. SMARTREST BINARY PAYLOAD

SmartRest messages can also be binary coded to reduce the data volume.

The following MQTT topics are available for this:

bs/ICCID	for sending messages
br/ICCID	for receiving messages

An MQTT binary payload request message has the following structure:

```
<XID><TID><Payload>
```

where:

<XID>	the X-ID of the template collection (required field, 0-terminated string)
<TID>	the ID of the SmartRest request template in the template collection (required field, 16-b
<Payload>	raw data (dependent on the template)

The raw data consists of values that can have the following data types in accordance with the SmartRest templates:

STRING	(0-terminated, encoding: UTF-8)
UNSIGNED	(32-bit unsigned int, byte order: big-endian)
INTEGER	(32-bit signed int, byte order: big-endian)
NUMBER	(32-bit float; byte order: big-endian)
DATE	(32-bit Unix timestamp, to the second; byte order: big-endian)

An MQTT message can consist of multiple MQTT binary payload messages:

```
<XID><TID><Payload><XID><TID><Payload><XID><TID><Payload>...
```

A binary coded message defined in this way can be parsed as follows:

- At the beginning, the 0-terminated string is extracted and interpreted as '<XID>'
- The '<TID>' template ID is determined from the next two bytes
- Note: in contrast to the normal SmartRest, where the TID can be up to 4 bytes long, binary requests can only access TIDs up to 65535.
- The template collection with '<XID>' is used to determine and provide the SmartRest template with '<TID>.' Based on the sequence of data types defined in the SmartRest template, the next bytes of the message are converted into the corresponding data types. This data is used to generate a traditional SmartRest MQTT message.
- The generated message is then sent to the Cloud of Things.
- If the byte array contains further bytes after parsing the first message, then these are interpreted as another message and processed in the same way.

In order to use the MQTT binary payload standard efficiently, you should make sure that the STRING data type is avoided if possible.

Accordingly, the SmartRest request templates should be adapted so that data such as numbers and dates is assigned to suitable data types and values such as status, measurement data description, and type no longer have to be transferred as variable data, but are predefined in the template. For example, you could dispense with transferring the status of an operation by defining a template for each status.

1. Optimized template for sending the result of a successfully executed operation:

```
10,410,PUT,/devicecontrol/operations/%%,application/vnd.com.nsn.cumulocity.
opera-
tion+json,application/vnd.com.nsn.cumulocity.operation+json,%%,UNSIGNED
STRING  STRING,"{""status"":  ""SUCCESSFUL"",  ""c8y_Command"":  {""result"":
""%%"",""syntax"": """",""text"": ""%%""}}"
```

2. Optimized template for sending the result of a failed operation:

```
10,411,PUT,/devicecontrol/operations/%%,application/vnd.com.nsn.cumulocity.
opera-
tion+json,application/vnd.com.nsn.cumulocity.operation+json,%%,UNSIGNED
STRING   STRING,"{""status"":   ""FAILED"",   ""c8y_Command"":   {""result"":
""%%"",""syntax"": """",""text"": ""%%""}}"
```

This way, more templates are needed. These templates can also be used to send traditional UTF-8 string encoded SmartRest MQTT messages.
For better readability, the values in the following message examples are separated by a space. Values of the data type STRING are enclosed in quotation marks that will not appear in the payload. A date value of type DATE is displayed as '<TS>' (=Timestamp, to the second).
An MQTT binary payload message that sends the result of an operation using template '(1)' and sets the status to "SUCCESSFUL" then looks like this:

```
"myXID" 410 11203 "README.md Device93707.properties" "ls"
```

As hexadecimal notation:

```
6D 79 58 49 44 00
01 9A
00 00 2B C3
52 45 41 44 4D 45 2E 6D 64 20 44 65 76 69 63 65 39 33 37 30 37 2E 70 72 6F 70 65 72 74 69 65 73 00
6C 73 00
```

Note that the message must, of course, really contain the corresponding byte values. This means: 109, 121, 88, 73, 68, 0, etc., and not the characters "6," "D,", "7," etc. in the hexadecimal format. This is only selected here to also show non-printable bytes.

Binary payloads can be sent, for example, for mosquitto_pub with the parameter –stdin-file/-s:

```
cat getOperation.bin | mosquitto_pub -t "bs/ICCID" -h BROKER_HOST -u ICCID
-P PASSWORD -p 8883 --cafile CERTIFICATE_FILE -s
```

Optimized template and MQTT binary payload message for sending the electricity measurement data:

```
10,300,POST,/measurement/measurements,application/vnd.com.nsn.cumulocity.me
asure-
ment+json,application/vnd.com.nsn.cumulocity.measurement+json,&&,NUMBER
DATE UNSIGNED,"{ ""Spannung"": {""U"": { ""value"": &&, ""unit"": ""Volt""
}  },   ""time"":   ""&&"",   ""source"":   {   ""id"":   ""&&""   },""type"":
""c8y_VoltageMeasurement"" }"
```

```
"myXID" 300 11.3583 <TS> 42503
```

Optimized template and MQTT binary payload message for sending the temperature measurement data:

```
10,305,POST,/measurement/measurements,application/vnd.com.nsn.cumulocity.me
asure-
ment+json,application/vnd.com.nsn.cumulocity.measurement+json,&&,NUMBER
DATE  UNSIGNED,"{  ""Temperatur"":  {""T"":  {  ""value"":  &&,  ""unit"":
""°Celsius""  }  },  ""time"":  ""&&"",  ""source"":  {  ""id"":  ""&&""
},""type"": ""c8y_TemperatureMeasurement"" }"
```

```
  "myXID" 305 22.5 <TS> 42503
```

4.12.1. ERROR HANDLING

In addition to the SmartRest error codes listed in the appendix, the following error messages may occur when processing the MQTT binary payload message.
The error messages use the same format as the Cumulocity SmartREST error messages and are sent to the device via the 'br/ICCID' topic.

4.12.2. PLACEHOLDERS IN ERROR MESSAGES

The following error messages are displayed with placeholders in angle brackets ('<' and '>').
When the error message is delivered, these placeholders are filled with concrete values.

The following placeholders are used in error messages:

<XID>	The X-ID of the referenced template
<Template-ID>	The template ID of the referenced template
<Binary-Message-Index>	A binary payload may contain multiple binary messages. This is the index of the binary message considered when the error occurred. Indexing starts with 1. Contains the value 'unknown' if the binary message index is unknown
<Byte-Position>	The number of read bytes when an error occurs. It is **not** guaranteed that the cause of the error is to be found at exactly this byte position. Contains the value 'unknown' if the byte position is unknown
<Parameter-Position>	Position of the parameter (which could not be parsed) in the binary message. Parameter indexing starts with 1
<Parameter-Type>	The expected type of a parameter as defined in the SmartRequest template

4.12.3. MISSING X-ID

If the X-ID does not exist in the message or is not 0-terminated:

```
    42 Malformed Request: no X-Id found in binary message '<Binary-Message-
Index>' around byte '<Byte-Position>'
```

4.12.4. MISSING TEMPLATE ID

If the template ID of a message cannot be determined:

```
  42 Malformed Request: no Template ID found in binary message '<Binary-
Message-Index>' around byte '<Byte-Position>'
```

4.12.5. SMARTREQUEST TEMPLATE NOT FOUND

If the determined template ID or X-ID cannot be found in the cache:

```
  43 Invalid template identifier: template with X-ID '<X-ID>' and Template
ID '<Template-ID>' not found in cache
```

The cause may be that

- the referenced template was not defined in the Cloud of Things tenant
- the template has not yet been loaded from the tenant; in this case the problem should disappear after a few minutes

4.12.6. INVALID PARAMETERS

If one of the supplied parameters cannot be parsed correctly:

```
  45 Invalid parameter at position '<Parameter-Position>': Expected type
'<Parameter-Type>' in binary message '<Binary-Message-Index>' around byte
'<Byte-Position>'
```

5. SMARTREST CODES

5.1. SMARTREST RESPONSE CODES

On each SmartRest command, the platform sends a result as an MQTT payload to the *mr/ICCID* topic. The codes are explained below.

20	*SmartREST* Template MO GId	Echo response message. Template was found or has been created and everything is OK.
40	*None*	Template not found.
41	Line number (optional	Template creation error.
42	Line number	Malformed request line
43	Line number	Invalid message identifier
45	Line number	Invalid message arguments.
50	Linenumber *HTTP* response code	Server error. This message occurs when an error happened between the *SmartREST* proxy and the platform.
87	amount of lines, X-Id	Indicates which X-Id was used to create the amount of following respons lines.

5.2. SMARTREST ERROR CODES

41	Cannot create templates for already existing template object
41	Duplicate message identifiers are not allowed
41	Bad request template definition
41	Bad response template definition
41	Bad value type: ...
41	Bad pattern
41	Not a valid message identifier for template creation
41	Invalid JsonPath
41	Using JsonPath to refer to a list of objects is not allowed for SmartRest
41	Using Filters (?) in JsonPath is not allowed for SmartRest
41	No content type supported for {GET or DELETE} templates.
41	No template string supported for {GET or DELETE} templates.
41	No content type found for {POST or PUT} templates.
41	No template string found for {POST or PUT} templates.
41	Values are only supported for templates with placeholder.
42	Malformed Request
43	Invalid message identifier
45	No arguments supported
45	Wrong number of arguments
45	Value is not a {value type}: {value}

5.3. EXAMPLE OF A TEMPLATE COLLECTION

```
15,TestTemplates-01
10,100,POST,/inventory/managedObjects,application/vnd.com.nsn.cumulocity.mana
gedOb-
ject+json,application/vnd.com.nsn.cumulocity.managedObject+json,,,"{""name"":
""Test Device"",""type"":""com_example_TestDevice"",""c8y_IsDevice"":{}}"
10,101,POST,/inventory/managedObjects,application/vnd.com.nsn.cumulocity.mana
gedOb-
ject+json,application/vnd.com.nsn.cumulocity.managedObject+json,&&,STRING,"{"
"name"":""&&"",""com_IsChildDevice"":""{}""}"
10,102,POST,/inventory/managedObjects/%%/childDevices,application/vnd.com.nsn
.cumulocity.managedObjectReference+json,application/vnd.com.nsn.cumulocity.ma
nagedObjectReference+json,%%,STRING STRING,"{ ""managedObject"" : { ""self""
: ""https://.../inventory/managedObjects/%%"" } }"
10,103,POST,/inventory/managedObjects,application/vnd.com.nsn.cumulocity.mana
gedOb-
ject+json,application/vnd.com.nsn.cumulocity.managedObject+json,,,"{""name"":
""Modbus Test Device"",""type"":""POSIX
Agent"",""c8y_IsDevice"":{},""com_cumulocity_model_Agent"":{},""c8y_ModbusCon
figuration"": {""protocol"": ""TCP"",""transmitRate"": 10,""pollingRate"":
10},""c8y_SerialConfiguration"": { ""baudRate"": 19200,""stopBits"":
1,""parity"": ""E"",""dataBits"": 8} }"
10,110,POST,/inventory/managedObjects,application/vnd.com.nsn.cumulocity.mana
gedOb-
ject+json,application/vnd.com.nsn.cumulocity.managedObject+json,&&,STRING
STRING,"{""name"":""&&"",""type"":""&&"",""c8y_IsDevice"":{},""com_cumulocity
_model_Agent"":{}}"
11,111,,"$.c8y_IsDevice","$.id"
11,112,,"$.com_IsChildDevice","$.id"
10,120,POST,/identity/globalIds/%%/externalIds,application/vnd.com.nsn.cumulo
city.externalId+json,application/vnd.com.nsn.cumulocity.externalId+json,%%,ST
RING STRING,"{ ""type"" : ""c8y_Serial"", ""externalId"" : ""%%"" }"
10,130,GET,/identity/externalIds/c8y_Serial/%%,,application/vnd.com.nsn.cumul
ocity.externalId+json,%%,STRING,
11,135,,"$.externalId","$.externalId","$.managedObject.id"
10,200,GET,/devicecontrol/operations?deviceId=%%&nocache=true&status=PENDING,
,application/vnd.com.nsn.cumulocity.operationCollection+json,%%,UNSIGNED,
11,201,$.operations,"$.c8y_Command","$.id","$.c8y_Command.text"
11,202,$.operations,"$.c8y_Restart","$.id","$.description"
11,203,$.operations,"$.c8y_Firmware","$.id","$.c8y_Firmware.url"
11,205,,"@.c8y_Command","@.id","@.c8y_Command.text"
11,206,,"@.c8y_Restart","@.id","@.description"
11,207,,"@.c8y_Firmware","@.id","@.c8y_Firmware.url"
10,211,PUT,/devicecontrol/operations/%%,application/vnd.com.nsn.cumulocity.op
eration+json,application/vnd.com.nsn.cumulocity.operation+json,%%,UNSIGNED
STRING,"{""status"": ""%%""}"
10,212,PUT,/devicecontrol/operations/%%,application/vnd.com.nsn.cumulocity.op
eration+json,application/vnd.com.nsn.cumulocity.operation+json,%%,UNSIGNED
STRING STRING STRING,"{""status"": ""%%"", ""c8y_Command"": {""result"":
""%%"",""syntax"": """",""text"": ""%%""}}"
10,300,PUT,/inventory/managedObjects/&&,application/vnd.com.nsn.cumulocity.ma
nagedOb-
ject+json,application/vnd.com.nsn.cumulocity.managedObject+json,&&,UNSIGNED
STRING STRING STRING STRING STRING STRING STRING STRING STRING
STRING,"{""c8y_Hardware"":{""model"":""&&"",""revision"":""&&"",""serialNumbe
r"":""&&""},""c8y_SoftwareList"":[{""name"":""Application"",""version"":""&&"
",""url"":""none""},{""name"":""Bootloader"",""version"":""&&"",""url"":""non
e""},{""name"":""Bluetooth"",""version"":""&&"",""url"":""none""},{""name"":"
"Mo-
dem"",""version"":""&&"",""url"":""none""}],""c8y_Mobile"":{""imei"":""&&"",""iccid"":""&&"",""imsi"":""&&""}}"
10,310,PUT,/inventory/managedObjects/&&,application/vnd.com.nsn.cumulocity.managedOb-
ject+json,application/vnd.com.nsn.cumulocity.managedObject+json,&&,UNSIGNED,"{""c8y_SupportedOperations"": [
""c8y_Configuration"",""c8y_SendConfiguration"",""c8y_DownloadConfigFile"",""c8y_UploadConfigFile"",""c8y_Software"",""c8y_SoftwareList"",""c8y_Firmware"",""c8y_FirmwareList"",""c8y_SystemCommand"",""c8y_Command"",""c8y_Restart"" ]
}"
10,320,PUT,/inventory/managedObjects/&&,application/vnd.com.nsn.cumulocity.managedOb-
ject+json,application/vnd.com.nsn.cumulocity.managedObject+json,&&,UNSIGNED
STRING STRING STRING,"{""c8y_Firmware"": {""name"": ""&&"", ""version"":
""&&"", ""url"": ""&&""} }"
10,350,DELETE,/inventory/managedObjects/&&,,,&&,UNSIGNED,
10,400,GET,/measurement/measurements?source=%%&pageSize=1000,,,%%,UNSIGNED,
11,401,$.measurements,"$.Spannung","$.time","$.Spannung.0.unit","$.Spannung.0.value"
11,402,$.measurements,"$.Kesseltemperatur","$.time","$.Kesseltemperatur.0.unit","$.Kesseltemperatur.0.value"
10,410,POST,/measurement/measurements,application/vnd.com.nsn.cumulocity.measurement+json,application/vnd.com.nsn.cumulocity.measurement+json,&&,STRING
STRING NUMBER STRING STRING STRING STRING,"{ ""&&"": {""&&"": { ""value"":
&&, ""unit"": ""&&"" } }, ""time"": ""&&"", ""source"": { ""id"": ""&&"" },
""type"": ""&&"" }"
10,450,GET,/measurement/measurements?source=%%&dateFrom=%%&dateTo=%%&pageSize=1000,,,%%,UNSIGNED STRING STRING,
10,500,GET,/devicecontrol/operations/%%?nocache=true,,application/vnd.com.nsn.cumulocity.operation+json,%%,UNSIGNED,
10,550,GET,/alarm/alarms?source=%%&pageSize=1000,,,%%,UNSIGNED,
11,551,$.alarms,"$.severity","$.time","$.status","$.text","$.type"
10,560,POST,/alarm/alarms,application/vnd.com.nsn.cumulocity.alarm+json,application/vnd.com.nsn.cumulocity.alarm+json,&&,STRING STRING STRING STRING
STRING STRING,"{ ""type"": ""&&"", ""time"": ""&&"", ""text"": ""&&"", ""sta-
tus"": ""&&"", ""severity"": ""&&"",  ""source"": { ""id"": ""&&"" } }"
10,600,GET,/event/events?source=%%&pageSize=1000,,,%%,UNSIGNED,
11,601,$.events,"$.c8y_Position","$.time","$.c8y_Position.lng","$.c8y_Position.alt","$.c8y_Position.lat"
10,610,POST,/event/events,application/vnd.com.nsn.cumulocity.event+json,application/vnd.com.nsn.cumulocity.event+json,&&,STRING STRING NUMBER NUMBER NUM-
BER,"{ ""time"": ""&&"", ""type"": ""c8y_LocationUpdate"", ""text"": ""Posi-
tion changed."", ""source"": { ""id"": ""&&"" }, ""c8y_Position"": { ""lng"":
""&&"", ""alt"": ""&&"", ""lat"": ""&&"" } }"
10,900,GET,/measurement/measurements?source=%%&dateFrom=%%&dateTo=%%&pageSize=1000,,,%%,UNSIGNED STRING STRING,
```