



SmartREST

Last updated: 03/03/2026

This content applies to the latest CD version of Cumulocity.

Specifications contained herein are subject to change and these changes will be reported in subsequent versions.

Copyright © 2026 Cumulocity GmbH.

The name Cumulocity GmbH and all Cumulocity GmbH product names are either trademarks or registered trademarks of Cumulocity GmbH and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

This software may include portions of third-party products. Third-party terms are set out in a 3rd-party-licenses file linked to or included with each installation package.

Table of Contents

Table of Contents	3
INTRODUCTION	5
HOW DOES SMARTREST WORK?	5
THE BASIC SMARTREST PROTOCOL	6
HOW ARE TEMPLATES REGISTERED?	7
HOW ARE RESPONSES HANDLED?	7
SMARTREST 2.0	9
CHANGES FROM SMARTREST 1.0	9
SUPPORTED TEMPLATES	9
TEMPLATE COLLECTIONS	10
Creating templates via MQTT	10
Request templates	10
Response templates	14
USING A DEFAULT COLLECTION	16
SMARTREST 1.0	18
INTRODUCTION	18
MQTT CLIENTID	18
SENDING AND RECEIVING SMARTREST 1.0	18
RECEIVING OPERATIONS	19
LIMITATIONS	19
THE PROTOCOL	19
DATA FORMAT	20
PROCESSING MODE	21
TEMPLATES	21
REQUEST TEMPLATES	22
RESPONSE TEMPLATES	22
REGISTRATION PROCESS	22
SYNTAX	23
USING SMARTREST WITH MULTIPLE X-IDS	25
SENDING MESSAGES	25
RECEIVING MESSAGES	25
CHECKING IF TEMPLATES ARE REGISTERED	25
REGISTERING TEMPLATES	25
SMARTREST REAL-TIME NOTIFICATIONS	26
USING REAL-TIME NOTIFICATIONS WITH SMARTREST	26
SUBSCRIBING WITH MULTIPLE TEMPLATES	28
BUILT-IN MESSAGES	28
REQUEST MESSAGES	28
RESPONSE MESSAGES	29
HANDLING OF IDS	31
CONCEPT OF ID-LESS COMMUNICATION	31
Example 1: ID of the device	31
Example 2: ID of alarms	32
JSON VIA MQTT	34
TOPIC STRUCTURE	34
Topic actions	34
SUPPORTED ENDPOINT	35
EXAMPLES	35
Create new event	35
Create many events	35
Update event	36
Delete event	36
Create a measurement data point	36

ERROR HANDLING	36
RECEIVING OPERATIONS	36
MQTT STATIC TEMPLATES	38
Templates quick reference	38
AUTOMATIC DEVICE CREATION	40
HANDLING NON-MANDATORY PARAMETERS	40
PUBLISH TEMPLATES	40
Inventory templates (1xx)	40
Measurement templates (2xx)	47
Alarm templates (3xx)	48
Event templates (4xx)	50
Operation templates (5xx)	52
Platform capabilities templates (6xx)	54
SUBSCRIBE TEMPLATES	54
Inventory templates (1xx)	54
Operation templates (5xx)	55
Platform capabilities templates (6xx)	62
UPDATING OPERATIONS	62
MQTT QUICK REFERENCE	64
Connection	64
Topics	64
Topic format	64
Device registration	64
Template registration	64
Templates	64

INTRODUCTION

This section walks you through the SmartREST protocol, the data format used, as well as the anatomy and registration of SmartREST templates. Built-in messages as well as errors are also discussed. For a step-by-step description, see [Using the REST interface](#).

The Cumulocity REST APIs provide you with a generic IoT protocol that is simple to use from most environments. It can be ad-hoc adapted to any IoT use case and it uses standard Internet communication and security mechanisms. While this is a great leap forward over tailored IoT protocols with proprietary technologies, it poses some challenges to very constrained environments such as low-end microcontrollers or low-bandwidth communication channels.

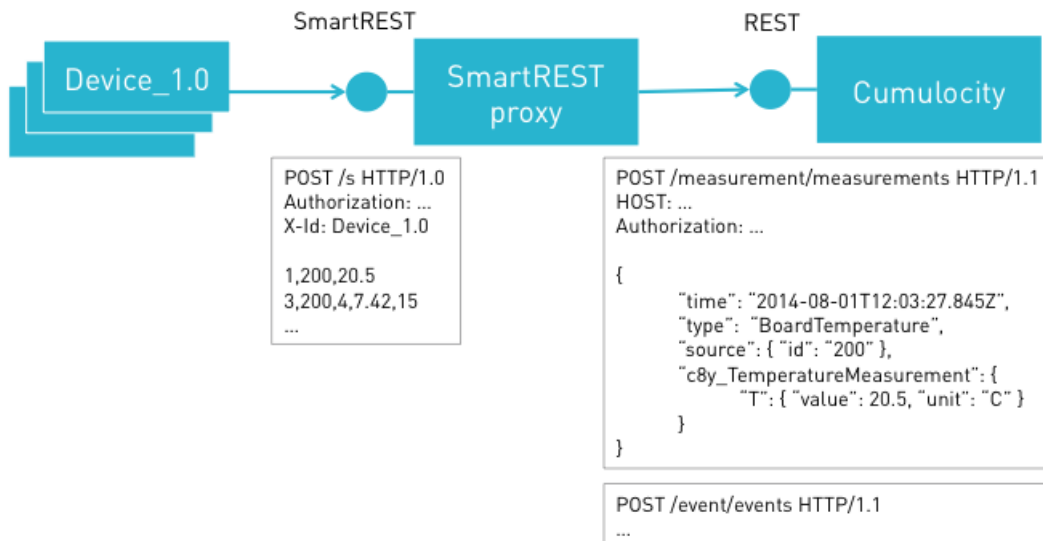
For these environments, Cumulocity offers the so-called SmartREST protocol. SmartREST combines the benefits of standard technology and tailored protocols:

- It continues to work on any network by using standard HTTP technology.
- It supports HTTP authentication and encryption.
- It still gracefully handles protocol versioning.
- Its network traffic usage is close to custom-optimized protocols by transferring pure payload data during normal operation.
- It is based on CSV (comma-separated values), hence, it is easy to handle from C-based environments.
- It supports server-generated timestamps for devices without clocks.

In the next section, we will discuss the concepts behind SmartREST and the basic protocol that is used. SmartREST is based on separating metadata from payload data by using templates, which are described below. Finally, we show how to send and receive data using SmartREST.

HOW DOES SMARTREST WORK?

The image below illustrates how SmartREST works. Devices and other clients connect to a dedicated SmartREST endpoint on Cumulocity and send their data in rows of comma-separated values. These rows are expanded by Cumulocity's SmartREST proxy into standard Cumulocity REST API requests. In the same way, responses from Cumulocity are compressed by the proxy from their original JSON format into comma-separated values before sending them back to the device.

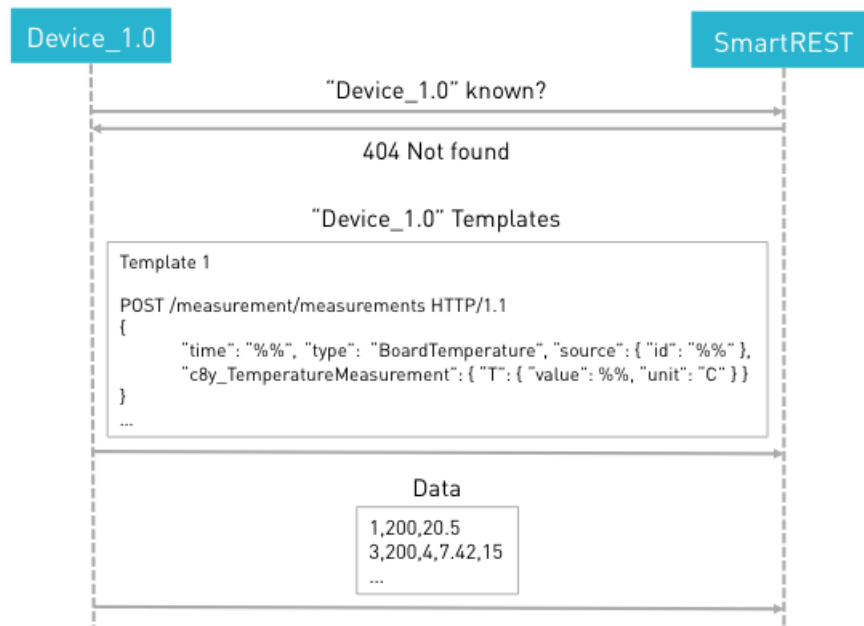


How can Cumulocity interpret comma-separated values into meaningful REST requests?

For that purpose, devices register templates with Cumulocity. The templates contain the expanded REST requests together with placeholders into which the Cumulocity SmartREST proxy consecutively inserts the comma-separated values. For responses, the templates describe which values to pick from the structured REST response to construct comma-separated values.

Templates are associated with software or firmware versions of a device. Usually, a particular implementation of a device or application can only issue a particular set of well-defined types of requests. All devices with the same implementation share the same set of request types. Hence, the templates can be defined at implementation time. To make the templates available to Cumulocity, the first device with a particular implementation will send its templates and makes them available for usage by all similar devices.

This process is illustrated in the image below. Assume that a device with an implementation version "Device_1.0" starts communicating through SmartREST. After retrieving its credentials, the device will ask the SmartREST proxy if its template is already known. If the template is not found on the server, the device will send its template in a single static text request to Cumulocity. Once this procedure has been carried out, all similar devices using that template can start communicating using SmartREST without re-sending the template to the server.



The example also roughly illustrates the translation process. In "Template 1", `"%%"` is a placeholder to be filled by the SmartREST proxy. `"time"` is filled with a server-side timestamp (see below). The remaining placeholders are filled with request data. The line `1,200,20.5` in the example request is interpreted as follows:

- The first column references the template to be used, in this case Template 1.
- `200` refers to the first free placeholder in the template, in this case the ID in the `"source"` element (The ID of the device that sends the measurement).
- `20.5` refers to the second free placeholder in the template, here the value of the temperature measurement.

THE BASIC SMARTREST PROTOCOL

The basic structure of all SmartREST requests is as follows:

- All requests are POST requests to the endpoint `/s`, regardless of what the requests finally translate to.
- The standard HTTP Basic Authorization header is used to authenticate the client.
- An additional `"X-Id:"` header is used to identify the implementation of the client, either as device type (such as "Device_1.0") or as an identifier returned by the template registration process.
- A request body contains rows of text in comma-separated value format. Each row corresponds to one request to the standard Cumulocity REST API.
- The response is always "200 OK".
- The response body again contains rows of comma-separated values. A row corresponds to a response from the Cumulocity REST API on a particular request.

Using the above example, a SmartREST request would be as follows:

```

POST /s HTTP/1.1
Authorization: <AUTHORIZATION>
X-Id: Device_1.0

1,200,20.5
  
```

And the corresponding response would be:

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked

20,0
```

To match the requests and responses, a response line contains – next to the error code – the line of the request that the response answers. In this example, `20` indicates “OK” and `0` refers to the first line of the request.

HOW ARE TEMPLATES REGISTERED?

As described above, a client using SmartREST will first ask if its SmartREST templates are already known to the server. This is done with an empty SmartREST request:

```
POST /s HTTP/1.1
Authorization: <AUTHORIZATION>
X-Id: Device_1.0
```

If the device implementation is known, the response will return an ID that can be used as “shorthand” in the “X-Id” header of later requests.

```
HTTP/1.1 200 OK

20,<id>
```

If the device implementation is unknown, the response will be:

```
HTTP/1.1 200 OK

40,"No template for this X-ID"
```

In this case, create all templates used in your device implementation.

```
POST /s HTTP/1.1
Authorization: <AUTHORIZATION>
X-Id: Device_1.0

10,1,POST,/measurement/measurements,application/vnd.com.nsn.cumulocity.measurement+json,,%%,NOW UNSIGNED NUMBER,{ "time": "%%",
"type": ... }
```

In this example, `10` refers to a request template (whereas “11” would refer to a response template). The template is number `1`, so SmartREST requests using this template have a “1” in their first column. The template refers to a `POST` request to the endpoint `/measurement/measurements` with a content type of `application/vnd.com.nsn.cumulocity.measurement+json`. The placeholder used in the template is `%%`. The placeholders are a time stamp (`NOW`), an unsigned number and a general number. Finally, the last column contains the body of the template to be filled in a POST request.

HOW ARE RESPONSES HANDLED?

The above example illustrated the handling of requests and request templates. For responses, [JSONPath](#) expressions translate Cumulocity REST responses into CSV. Assume, for example, that a device has a display and can show a message on it. An operation to update the message would look like:

```
{
  "c8y_Message": {
    "text": "Hello, world!"
  },
  "creationTime": "2019-02-25T08:32:45.435+01:00",
  "deviceId": "8789602",
  "status": "PENDING"
}
```

On the client side, the device mainly needs to know the text to be shown. In JSONPath, the `"text"` property is extracted using the following syntax:

```
$.c8y_Message.text
```

In this syntax, `$` refers to the root of the data structure and the dot (`.`) selects an element from the data structure. For more details, refer to the [JSONPath](#) reference.

A device usually queries for all operations that are associated with it and that have a status of PENDING. The standard Cumulocity response to such queries looks like:

```
{
  "operations": [{
    "c8y_Message": {
      "text": "Hello, world!"
    },
    "creationTime": "2014-02-25T08:32:45.435+01:00",
    "deviceId": "8789602",
    "status": "PENDING"
  }, {
    "c8y_Relay": {
      "status": "OPEN"
    }
  }
]}
}
```

The response contains a list of operations which can have different types. To work with such a structure, use the following response template:

```
11,2,$.operations,$.c8y_Message,$.c8y_Message.text
```

This means, value by value:

- 11: This is a response template.
- 2: It is the template number 2.
- `$.operations`: The response is a list and the list's property is "operations".
- `$.c8y_Message`: This template applies to responses with the property `c8y_Message`.
- `$.c8y_Message.text`: The text will be extracted from the message and will be returned.

The SmartREST client will thus get the following response:

```
HTTP/1.1 200 OK

2,0,"Hello, world!"
```

That is, the response was created using template 2, the template to translate display message operations. The response refers to the first request sent. The actual message to set is "Hello, world!".

SMARTREST 2.0

This section describes the SmartREST 2.0 payload format that can be used with the Cumulocity MQTT implementation.

SmartREST 2.0 was designed to make use of the MQTT protocol, so it may reduce the payload even more than the SmartREST 1.0 via HTTP.

SmartREST 2.0 is only available via MQTT and it offers the following MQTT topics for the main communication:

To publish messages:

```
s/uc/<X-ID>
```

To publish messages in transient mode:

```
t/uc/<X-ID>
```

To publish messages in quiescent mode:

```
q/uc/<X-ID>
```

To publish messages in CEP mode:

```
c/uc/<X-ID>
```

Refer to [Processing mode](#) for more information about transient, quiescent & CEP data processing.

To subscribe for responses:

```
s/dc/<X-ID>
```

The topics for creating templates are described in [Creating templates via MQTT](#).

CHANGES FROM SMARTREST 1.0

In its base, SmartREST 2.0 is like the previous version: a CSV-like payload format that is backed by previously created templates to finally create the targeted JSON structure.

Several changes in the functionality have been made:

- Templates no longer contain IDs of objects (instead, IDs will be resolved by, for example, MQTT ClientId)
- Managed objects can be created and retrieved directly with external IDs
- Creating request templates now uses JSON path (like response templates)
- Support for lists in responses
- Responses also return if only part of the patterns were found
- Declaring a default X-ID for the connection

SUPPORTED TEMPLATES

SmartREST 2.0 lets you create templates for the following matching HTTP methods:

API	GET	POST	PUT
Inventory	x	x	x
Alarm		x	x

API	GET	POST	PUT
Event		x	
Measurement		x	
Operation			x

Additionally, you can create templates to return certain values from responses and operations.

TEMPLATE COLLECTIONS

A template collection is a set of request and response templates that specifies a device communication protocol. Each collection is referenced by a unique ID (called X-ID).

Creating templates via MQTT

Like in SmartREST 1.0, you must pass all templates in a collection in one message. After the creation of a template collection, it can no longer be modified through MQTT.

When creating templates, the client must publish to the following topic:

```
s/ut/<X-ID>
```

To verify if a template collection exists, the client can subscribe to the topic:

```
s/dt
```

When subscribed, the client can send an empty message to the creation topic which will trigger a new message about the creation status of this X-ID.

Examples

Empty publish to s/ut/myExistingTemplateCollection

```
20,myExistingTemplateCollection,<ID of collection>
```

Empty publish to s/ut/myNotExistingTemplateCollection

```
41,myNotExistingTemplateCollection
```

Request templates

A request template contains the following basic fields:

Field	Data type	Possible values	Mandatory	Description
messageld	String		Y	Unique ID to reference the template within the collection
method	String	GET PUT POST	Y	Whether to get, update or create data

Field	Data type	Possible values	Mandatory	Description
api	String	INVENTORY MEASUREMENT ALARM EVENT OPERATION	Y	Cumulocity API to be used
response	Boolean	true false	N	Whether the request should trigger response templates. For GET templates by default true otherwise by default false
mandatoryValues	List<String>		Y	Values for the mandatory fields on the API. The values depend on the API and method the template uses
customValues	List<CustomValue>		N	Custom values that should be added to the object

A request template lists all the fragments in the object structure (mandatory and custom) that should be added when creating or updating the data. It can set fixed values in the template that will then be replaced by the server. If it does not set the value in the template, the value must be included in the publish message (this includes mandatoryValues).

INFO

If the message rate limit per second is exceeded, the requests are delayed and kept in queue. If the queue limit number is exceeded, the client messages are rejected and the client is disconnected.

Example

We create a template to create a measurement like this (measurements have two mandatory values: type and time)

```
# 10,msgId,method,api,response,type,time,custom1.path,custom1.type,custom1.value
10,999,POST,MEASUREMENT,,c8y_MyMeasurement,,c8y_MyMeasurement.M.value,NUMBER,
```

This template defines one additional custom property for the measurement. It leaves two fields empty in the template declaration (time and the custom property), so to use the template the client must send these two values:

```
999,2016-06-22T17:03:14.000+02:00,25
# We can also use server time by leaving the time empty
999,,25
```

The following sections will get into more detail of how to create and use different templates.

GET templates

GET templates for the inventory do not need any mandatory or custom values. Instead, they use two different fields.

With SmartREST 2.0 you have the option to either get an object from inventory by its ID or by an external ID directly. Therefore, instead of the fields **mandatoryValues** and **customValues**, the following two fields are used:

Field	Data type	Possible values	Mandatory	Description
-------	-----------	-----------------	-----------	-------------

Field	Data type	Possible values	Mandatory	Description
byId	Boolean	true false	Y	Whether the GET should be executed by Cumulocity ID (=true) or externalId (=false)
externalIdType	String		N	Sets a fixed externalIdType if the template calls by externalId

This enables you to query inventory in three different ways:

By Cumulocity ID

```
# Creation:
10,999,GET,INVENTORY,,true
# Usage:
999,123456
```

By external ID with a fixed type in the template

```
# Creation:
10,999,GET,INVENTORY,,false,c8y_Serial
# Usage:
999,myDeviceImei
```

By external ID without fixed type in the template

```
# Creation:
10,999,GET,INVENTORY,,false
# Usage:
999,c8y_Serial,myDeviceImei
```

POST templates

POST templates require a different set of mandatory values based on the API:

API	mandatory values
MEASUREMENT	type, time
EVENT	type, text, time
ALARM	type, text, status, severity, time
INVENTORY	externalIdType

This results in the following minimal template creations:

```
# Creation:
10,100,POST,MEASUREMENT,false,c8y_MyMeasurement,,c8y_MyMeasurement.M.value,NUMBER,
10,101,POST,EVENT,,c8y_CustomEvent,mytext,,
10,102,POST,ALARM,,c8y_CustomAlarm,mytext,ACTIVE,MAJOR,
# Usage:
100
101
102
```

Creating data on the inventory optionally includes the creation of an externalId for that object. This is controlled by the mandatory value

externalIdType.

❗ IMPORTANT

POST Inventory templates start with the value of the externalId after the msgId. Leaving this column empty will result in not creating an external ID.

```
# Creation:
10,100,POST,INVENTORY,,c8y_MySerial
# Usage:
# Create object with externalId c8y_MySerial/mylmei
100,mylmei
# Create object with externalId c8y_MySerial/mylmei
101,mylmei,c8y_MySerial
# This message will result in not creating an external ID
101,,c8y_MySerial
```

PUT templates

PUT templates for inventory follow the same logic as the GET templates, and with the addition that you can also use custom values for PUT.

```
# Creation:
# 10,msgId,method,api,response,byId,externalIdType,custom1.path,custom1.type,custom1.value
10,999,PUT,INVENTORY,,false,c8y_Serial,c8y_MyCustomValue,STRING,
# Usage:
999,myDeviceIimei,myValue
```

The PUT template for alarms uses the type of the alarm to find the alarm to update. It will first check the ACTIVE alarms and, if there is no ACTIVE alarm, it will check the ACKNOWLEDGED alarms.

```
# Creation:
# 10,msgId,method,api,response,type,custom1.path,custom1.type,custom1.value
10,999,PUT,ALARM,,c8y_MyCustomAlarm,status,ALARMSTATUS
# Usage:
999,CLEARED
```

PUT templates for operations use the fragment of the operation to find the operation. It will first check the EXECUTING operations and, if there is no EXECUTING operation, it will check the PENDING operations.

```
# Creation:
# 10,msgId,method,api,response,fragment,custom1.path,custom1.type,custom1.value
10,999,PUT,OPERATION,,c8y_MyOperation,status,OPERATIONSTATUS,SUCCESSFUL,c8y_Fragment.val,NUMBER,
# Usage:
999,24
```

Adding custom properties

All POST and PUT values enable you to add custom properties to the results of the templates.

A single custom property requires you to add the following three values to your template creation:

Field	Description
path	A JsonPath for the value that should be set
type	An optional data type of the value. Default: STRING

Field	Description
-------	-------------

value	The value to be set. Leaving this field empty requires the client to send the value when using the template
-------	---

Type	Description
------	-------------

STRING	The default type. No additional verification of the value
--------	---

DATE	A time stamp in the ISO 8601 format. Using date and not sending a time stamp results in the use of server time
------	--

NUMBER	An integer or number with decimal places
--------	--

INTEGER	An integer
---------	------------

UNSIGNED	An integer (only positive)
----------	----------------------------

FLAG	An empty map (for example c8y_IsDevice: {}). The client does not need to send anything for this value
------	---

SEVERITY	A severity of an alarm. Used to update the severity field of alarms
----------	---

ALARMSTATUS	A status of an alarm. Used to update the status field of alarms
-------------	---

OPERATIONSTATUS	A status of an operation. Used to update the status field of operations
-----------------	---

Examples

Template for clearing an alarm with an additional custom property

```
# Creation:
10,999,PUT,ALARM,,c8y_MyCustomALarm,status,ALARMSTATUS,CLEARED,c8y_CustomFragment.reason,STRING,
# Usage:
999,Device resolved alarm on its own
```

Template for creating a custom measurement

```
# Creation:
10,999,POST,MEASUREMENT,,c8y_CustomMeasurement,,c8y_CustomMeasurement.custom.value,NUMBER,,c8y_CustomMeasurement.custom.unit,STRING,X
# Usage:
999,30.6
```

Template for updating a property in the device

```
# Creation:
10,999,PUT,INVENTORY,,false,c8y_Serial,c8y_MyCustomValue,STRING,
# Usage:
999,myDeviceImei,updatedValue
```

Response templates

The SmartREST 2.0 response templates use the same structure as in SmartREST 1.0.

Field	Data type	Mandatory	Description
messageId	String	Y	Unique ID to reference the template within the collection
base	String	N	A JsonPath prefix that all patterns will use
condition	String	N	A JsonPath that must exist in the object to use the pattern
pattern	List<String>	Y	A list of JsonPath that will be extracted from the object and returned to the device

Response templates will be used for every operation and for any request template that defines the response field with true. In each case, the server will try every registered response template, so there might be multiple response lines for a single operation or request.

SmartREST 2.0 will always return a response template if the condition is true (or no condition was defined). Patterns that did not resolve will be returned as empty string.

You should make use of the condition field to control when response templates should be returned.

In the examples below, you can see how to query data and parse custom operations.

Querying data from the device object

Device object:

```
{
  "id": "12345",
  "type": "myMqttDevice",
  "c8y_IsDevice": {},
  "c8y_Configuration": "val1=1\nval2=2"
}
```

Template creation:

```
10,999,GET,INVENTORY,,true
11,888,,c8y_IsDevice,type,c8y_Test,c8y_Configuration
```

Client publishes:

```
999,12345
```

Client receives:

```
888,myMqttDevice,, "val1=1\nval2=2"
```

Parsing custom operations

Operation object:

```
{
  "id": "12345",
  "deviceId": "67890",
  "agentId": "67890",
  "status": "PENDING",
  "c8y_CustomConfiguration": {
    "val1": "1",
    "val2": "2",
    "customValues": ["a", "b", "c"]
  },
  "description": "Send custom configuration"
}
```

Template creation:

```
11,111,c8y_CustomConfiguration,deviceId,val1,val2,customValues[*]
11,222,,deviceId,c8y_CustomConfiguration.val1,c8y_CustomConfiguration.val2
11,333,,deviceId,val1,val2,customValues[*]
11,444,c8y_CustomConfiguration,c8y_CustomConfiguration.val3,val1,val2,customValues[*]
```

Client receives (assuming the ClientId is "myMqttTestDevice"):

```
111,myMqttTestDevice,1,2,a,b,c
222,myMqttTestDevice,1,2
333,myMqttTestDevice,,,
```

The template 444 is not returned as the condition does not match the operation.

Querying data from the device object containing key with multiple objects

Device object:

```
{
  "id": "12345",
  "name": "test",
  "type": "c8y_MQTTdevice",
  "c8y_IsDevice": {},
  "myList": [
    {
      "pid": 12345,
      "type": "test"
    },
    {
      "pid": 123456,
      "type": "test2"
    }
  ]
}
```

Template creation:

```
10,999,GET,INVENTORY,,true
11,888,, "$.myList[*].type"
```

Client publishes:

```
999,12345
```

Client receives:

```
888,test,test2
```

USING A DEFAULT COLLECTION

Having the X-ID as part of the topic gives you the freedom to easily use multiple template collections, but adds additional bytes for every message. If anyway the device uses mostly (or completely) a single collection, it makes sense to specify this collection as your default collection.

With a default collection specified, the client can use special topics which do not require the X-ID, and instead the server will use the X-ID previously specified. The topics are `s/ud` for publishing and `s/dd` for subscribing.

You can specify one X-ID within your MQTT ClientId (see [MQTT implementation](#)).

Your MQTT ClientId could look like this:

```
d:myDeviceSerial:myDefaultTemplateXID
```

INFO

If you use a default X-ID, you must include in the **ClientId** the **d:** at the beginning to specify that the client is a device.

It is not required that the default template exists at the time of establishing the MQTT connection (it will be verified once the client uses it).

SMARTREST 1.0

❗ IMPORTANT

SmartREST 1.0 has been superseded by SmartREST 2.0.

SmartREST 1.0 will be maintained by Cumulocity but no longer actively developed. We highly recommend you to use [SmartREST 2.0](#) for new device integrations.

This section describes how you can use your existing SmartREST 1.0 templates with MQTT.

Note that SmartREST 1.0 was designed for HTTP request/response and does not support the ID-less communication with MQTT. It only uses the MQTT connection to send exactly the same request as you would send using HTTP. Therefore, it comes with some limitations as MQTT is not request/response.

The support for SmartREST 1.0 was added to ease transition if you have an existing implementation using it.

For general information on SmartREST 1.0, refer to [Using the REST interface](#).

INTRODUCTION

MQTT CLIENTID

Although you must send the IDs in the body of each message with SmartREST 1.0, it is still important to connect with the correct MQTT ClientId.

The MQTT ClientId must match the externalId with type `c8y_Serial` of your device. It is used to assign the correct operations and responses.

SENDING AND RECEIVING SMARTREST 1.0

In general, the following holds for SmartREST requests and responses via MQTT:

- All request rows will be sent as single MQTT messages. A single request message always yields a single response message instead of being split up to several response messages.
- All SmartREST response templates will be applied to the JSON response of a single request.
- Every matching response template will yield one row in the response.
- Response lines are separated by `\n`.

[Sending SmartREST 1.0](#)

To send data to the server you can publish the same content as you would POST to the SmartREST endpoint `/s`.

The X-ID header is part of the topic the client must publish on.

```
s/ul/<X-ID>
```

An X-ID acts as a protocol identifier and identifies the template collection. An X-ID should be immutable: it always identifies exactly the same template collection. If the template collection changes, the X-ID must also change. An X-ID should also be globally unique: select an X-ID that is not used by anyone else. To make sure, we recommend you to use reverse domain names, for example:

```
com.acme.gw801-v1
com.acme.gw801-v2
```

We also recommend you to add the protocol version as a postfix in the X-ID.

Processing mode

Since the Cumulocity SmartREST protocol supports TRANSIENT processing mode for avoiding storage of sent data in the database, publishing on MQTT `t/` topic instead of `s/` topic will only pass the data to real-time processing.

```
t/ul/<X-ID>
```

The Cumulocity SmartREST protocol also supports QUIESCENT processing mode for avoiding real-time notifications by publishing on MQTT `q/` topic instead of `s/` topic. Currently, the QUIESCENT processing mode is applicable for measurements and events only.

```
q/ul/<X-ID>
```

The Cumulocity SmartREST protocol also supports CEP processing mode to ensure that data is only sent to the real-time event processing engine with real-time notifications, disabled by publishing on MQTT `c/` topic instead of `s/` topic. Currently, the CEP processing mode is applicable for measurements and events only.

```
c/ul/<X-ID>
```

Receiving SmartREST 1.0

If a template triggers a response template, the returning message will be published by the server on the following topic.

```
s/dll/<X-ID>
```

This topic can be subscribed by the client.

RECEIVING OPERATIONS

SmartREST 1.0 via HTTP offers the `/notification/operations` endpoint to listen to realtime operations. You can receive the same content on the following MQTT topic.

```
s/ol/<X-ID>
```

INFO

To get notifications running, the platform device must have an external ID set which matches the MQTT client ID, otherwise it will not receive notifications.

LIMITATIONS

MQTT currently does not support request/response. Therefore, if you send a request on the publish topic and receive a response on the subscribe topic, the client cannot securely match that they belong together.

You can counter this limitation by designing the templates in a way that you never need to know what request triggered the response, and the client automatically knows it by the **messageId**.

THE PROTOCOL

SmartREST is built upon the well-established HTTP protocol making it work everywhere since most popular platforms provide an HTTP client through which SmartREST can be accessed. SmartREST communicates exclusively through the `/s` resource using the HTTP `POST` method for bidirectional communication. The payload data format in CSV (comma-separated values).

The following example shows the communication between a client and the SmartREST endpoint. Note the `Authorization` header and the custom `X-Id` header in the request which specifies the SmartREST template to use for this request.

```
POST /s HTTP/1.0
Authorization: Basic ...
X-Id: ...
Transfer-Encoding: chunked

100,1234456
```

Each SmartREST request is represented by one row having a unique unsigned integer as its first value determining the action and subsequent values for parameters.

The SmartREST endpoint yields the following response. Note that the HTTP response code is always `200` unless the SmartREST endpoint is not available.

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked

200,1,123456,Request result
```

Each row yielded by the SmartREST endpoint represents a set of extracted values from the result of a SmartREST request containing a unique unsigned integer, the SmartREST request line number and the extracted data values, respectively.

INFO

If the response from Cumulocity REST API was empty (for example like after deleting a Managed Object) then the response from SmartREST would be empty as well, regardless of registered response templates.

INFO

Inventory access via SmartREST is limited to inventory objects which are global or for which the client is the owner. Role assignments are not evaluated.

DATA FORMAT

The CSV (comma-separated values) format is used for communication with the SmartREST endpoint. The following rules must be followed to ensure a frictionless communication.

- Every linebreak must be encoded by the character sequence `\n`.
- Values are always separated by a comma (`,`).
- If a value contains double-quotes (`"`), commas (`,`), leading or trailing whitespaces, line-breaks (`\n`), carriage returns (`\r`) or tab stops, it must be surrounded by quotes (`"`). Contained double-quotes (`"`) must be escaped by prepending another double-quote (`""`).

The same escaping rules apply to messages that will be sent from the server to the client.

Examples

The following examples illustrate the rules stated above:

```
100,Hello world!
101," I have leading whitespace!"
102,"I have trailing whitespace!"
103,"I contain a line
break!"
104,"I have ""quotes""!"
105,I also have 'quotes'!
```

PROCESSING MODE

Similar to [Cumulocity REST implementation](#) every communication in SmartREST which can lead to data update (i. e., POST, PUT, DELETE) supports four processing modes, PERSISTENT, TRANSIENT, QUIESCENT or CEP.

If the data sent to the SmartREST endpoint must be both stored in Cumulocity database and be transferred to real-time processing, then PERSISTENT mode should be set. It is also enabled by default and does not require additional configuration.

In case when it is only needed to communicate data to real-time processing, the TRANSIENT processing mode should be specified by adding it to the header of HTTP request:

```
POST /s HTTP/1.0
Authorization: Basic ...
X-Id: ...
X-Cumulocity-Processing-Mode: TRANSIENT
Transfer-Encoding: chunked

100,1234456
```

With TRANSIENT mode in the header the body data is not persisted in Cumulocity database.

During real-time processing, CEP scripts can be used to define if updates should be stored in the database or not.

The QUIESCENT processing mode should be specified if data sent to the SmartREST endpoint must be both stored in the Cumulocity database and be transferred to real-time processing but real-time notifications must be disabled. Currently, the QUIESCENT processing mode is applicable for measurements and events only.

The CEP processing mode should be specified if data sent to the SmartREST endpoint must only be transiently sent to real-time processing engine with real-time notifications disabled. Currently, the CEP processing mode is applicable for measurements and events only.

INFO

Events are always delivered to CEP/Apama for all processing modes. This is independent from real-time notifications.

TEMPLATES

SmartREST templates are a collection of request and response templates used for the conversion of CSV data and Cumulocity REST API calls. Additionally, SmartREST templates contain a template identifier which is compared to the custom `X-Id` header field to identify the SmartREST template used for processing.

Each request and response template has a unique numeric identifier called the message identifier which is referenced by the first value of each SmartREST request or response row. To avoid collision with one of the default message identifiers, developers are advised to select message identifiers starting at `100`.

REQUEST TEMPLATES

A request template contains all necessary information to convert a SmartREST request into a corresponding REST API call which is then sent to the platform.

A request template contains the following information:

- A unique unsigned integer as a message identifier
- The request method, for example, `GET` or `POST`.
- The resource URI, for instance `/inventory/managedObjects`
- The `Content-Type` and `Accept` header values of the sent and received data
- A placeholder such as `%%`
- The expected request parameters such as `STRING` s, `NUMBER` s, `UNSIGNED` integers and `DATE` s
- The template string with placeholders for each parameter

RESPONSE TEMPLATES

A response template contains the necessary information to extract data values from a platform REST API call response which are then sent back to the client in the CSV data format.

The following information is contained within a response template:

- A unique unsigned integer as a message identifier
- A JSON path referencing a base object or object list to extract data from, for example, `$` or `$.managedObjects`. If the JSON path points to a list of objects, one row of extracted data for each object in the list is yielded.
- A JSON path which must exist within the base object or base object list in order to extract values, for example, `$.id`. The value is not added to the response.
- A variable number of JSON paths for each value to extract, for example, `$.id`, `$.name` or `$.type`. Values are added to the response in the order they were defined in the template.

REGISTRATION PROCESS

This reference guide solely focusses on the registration of SmartREST templates using the SmartREST `/s` endpoint. Alternatively, templates can also be registered using the platform inventory API.

Before a SmartREST template can be registered, its existence must be checked. If the template already exists, a registration is not necessary and yields an error message.

The existence of a SmartREST template can be checked by making an empty request:

```
POST /s HTTP/1.0
Authorization: Basic ...
X-Id: ...
Transfer-Encoding: chunked
```

If the template exists, the following response is yielded where the message identifier `20` indicates that the template exists in the inventory and the parameter `123456` indicates the managed object GId of the template:

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked

20,12345
```

If the template does not exist, a response containing an error message is yielded:

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked

40,"No template for this X-ID."
```

If the template does not exist, a template registration request can be issued using the previously checked `X-Id`.

Templates can be registered with one single request containing SmartREST template in the form of CSV data. The difference between a template registration request and a normal SmartREST request is that rows are not processed individually during template registration.

```
POST /s HTTP/1.0
Authorization: Basic ...
X-Id: ...
Transfer-Encoding: chunked

10,100,POST,/inventory/managedObjects,application/vnd.com.nsn.cumulocity.managedObject+json,application/vnd.com.nsn.cumulocity.managedObject+json,,["name":"Test Device","type":"com_example_TestDevice","c8y_IsDevice":{}]"
11,201,"$c8y_IsDevice","$.id"
```

If the template registration is successful, a response like below will be returned.

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked

20,12345
```

SYNTAX

Each request and response template is contained within one row of the template data. Request templates are indicated by the message identifier `10` and response templates by the identifier `11`. Should one of those message identifiers occur in a SmartREST request, the entire request is treated as a template. Thus any other message identifier besides `10` and `11` will yield an error.

Request templates

Request templates have the following syntax:

```
10,<ID>,<METHOD>,<URI>,<CONTENT>,<ACCEPT>,<PLACEHOLDER>,<PARAMS>,<TEMPLATE>
```

Where:

- `<ID>` is the message identifier of the request template.
- `<METHOD>` is the HTTP method used for the request. `GET`, `POST`, `PUT` and `DELETE` are supported.
- `<URI>` is the resource identifier.
- `<CONTENT>` is the `Content-Type` header field value.
- `<ACCEPT>` is the `Accept` header field value. This is mostly equal to `<CONTENT>`.
- `<PLACEHOLDER>` is the placeholder string which is replaced by parameters in the request URI and template string.
- `<PARAMS>` is a space-separated list of parameter types which are required for the template. The number of occurrences of the placeholder string in the request URI and template string must be equal to the number of parameters specified.
 - `STRING` parameters will only yield an error if no value is specified.
 - `UNSIGNED` parameters will yield an error if the supplied parameter is not an unsigned integer.
 - `INTEGER` parameters will yield an error if the supplied parameter is not a signed integer.
 - `NUMBER` parameters will yield an error if the supplied parameter is not a floating-point number.
 - `DATE` parameters will yield an error if the parameter cannot be parsed as a date.
 - `NOW` parameters will never yield an error. No request parameter is required.
- `<TEMPLATE>` is the actual template string which gets sent as payload to the platform after the placeholders have been replaced with the parameter values.

Here is a set of example requests:

```

10,100,POST,/alarm/alarms,application/vnd.com.nsn.cumulocity.alarm+json,application/vnd.com.nsn.cumulocity.alarm+json,&&,UNSIGNED NOW,"
{"source":{"id":"","&&"},"type":"c8y_MyAlarmFromSmartREST","text":"This alarm was created by using
SmartREST","severity":"MAJOR","status":"ACTIVE","time":"","&&"}
10,200,POST,/measurement/measurements,application/vnd.com.nsn.cumulocity.measurement+json,application/vnd.com.nsn.cumulocity.measurement
+json,&&,UNSIGNED UNSIGNED NOW UNSIGNED,{"c8y_SmartMeasurement":{"temp1":{"value":"","unit":"C"},"temp2":
{"value":"","unit":"F"},"time":"","&&","source":{"id":"","&&"},"type":"c8y_SmartMeasurement"}
10,300,PUT,/inventory/managedObjects/&&,application/vnd.com.nsn.cumulocity.managedObject+json,&&,"c8y_Hardware":
{"model":"","&&","revision":"","&&"}
10,301,PUT,/alarm/alarms/&&,application/vnd.com.nsn.cumulocity.alarm+json,&&,"status":"CLEARED"}
10,302,PUT,/devicecontrol/operations/&&,application/vnd.com.nsn.cumulocity.operation+json,&&,"status":"SUCCESSFUL"}
10,600,GET,/identity/externalIds/c8y_Serial/&&,,&&,
10,601,GET,/devicecontrol/operations?deviceId=##&status=PENDING,,,##,,
10,602,GET,/inventory/managedObjects/&&,,&&,

```

The example requests are also included in our Postman collection. See [Using Postman](#) to learn how to import the collection into Postman. In the Postman collection, the set of requests is located in **SmartREST > Register Request Templates**.

Example

Create a device:

```

10,100,POST,/inventory/managedObjects,application/vnd.com.nsn.cumulocity.managedObject+json,application/vnd.com.nsn.cumulocity.managedObje
ct+json,%%,STRING,{"name":"","&&","type":"com_example_TestDevice","c8y_IsDevice":{}}

```

Explanation:

- **10** describes a request template.
- **100** is the message identifier of the request template.
- **POST** is the HTTP method used.
- **/inventory/managedObjects** is the resource identifier.
- **application/vnd.com.nsn.cumulocity.managedObject+json** is the content type.
- **application/vnd.com.nsn.cumulocity.managedObject+json** is the accept content type.
- **%%** is the placeholder string.
- **STRING** specifies that the request accepts one parameter which must be a string.

Update operation to EXECUTING:

```

10,101,PUT,/devicecontrol/operations/%%, application/vnd.com.nsn.cumulocity.operation+json,
application/vnd.com.nsn.cumulocity.operation+json,%%,INTEGER,{"status":"EXECUTING"}

```

Response templates

Response templates have the following syntax:

```

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

```

Where:

- **<ID>** is the message identifier of the response template.
- **<BASE>** is the base JSON path pointing to an object or object list from which the values are extracted.
- **<COND>** is a conditional JSON path which gets checked for existence. Only if the path exists, values are extracted.
- **<VALUE>** is a JSON path pointing to a value to extract within the base object or object in the base object list. An unlimited number of **<VALUE>** s can be specified.

Example

```

11,201,,"c8y_IsDevice","$.id"

```

Explanation:

- **11** describes a response template.

- `201` is the message identifier of the response template.
- The base object JSON path is empty, thus `$` is assumed.
- `$.c8y_IsDevice` specifies that values are only extracted if the object has a fragment called `c8y_IsDevice`.
- `$.id` is the value extracted, namely the device ID.

USING SMARTREST WITH MULTIPLE X-IDS

SmartREST supports sending of messages for different X-Ids within the same request. In this case the X-Id header mustn't be used but instead the body will contain additional information about which lines belong to which X-Id.

SENDING MESSAGES

To indicate the X-Id in the body it is possible to include the following line

```
15,myxid
```

All following lines will be handled with the given X-Id until you enter the next X-Id line.

```
15,myxid1
...
...
15,myxid2
...
```

RECEIVING MESSAGES

When sending with multiple X-Ids the response also can contain responses from multiple X-Ids. The response will contain an additional line that will indicate which X-Id the following lines are from. The second value in this line indicates how many lines are following from this X-Id.

```
87,2,myxid1
...
...
87,1,myxid2
...
```

CHECKING IF TEMPLATES ARE REGISTERED

You can check if templates are already existing by just include X-Id lines in the body.

```
15,myxid1
15,myxid2
15,myxid3
15,myxid4
```

You will get the same response like described in the registration process but for every line.

```
20,12345
20,12346
40,"No template for this X-ID."
20,12347
```

REGISTERING TEMPLATES

Template registration also supports the use of the X-Id in the body. Therefore you can create multiple in a single request.

```
15,myxid1
10,100,POST,/inventory/managedObjects,application/vnd.com.nsn.cumulocity.managedObject+json,application/vnd.com.nsn.cumulocity.managedObject+json,,,{""name"":""Test Device"", ""type"":""com_example_TestDevice"", ""c8y_IsDevice"":{}}
11,201,, "$c8y_IsDevice", "$.id"
15,myxid2
10,100,POST,/inventory/managedObjects,application/vnd.com.nsn.cumulocity.managedObject+json,application/vnd.com.nsn.cumulocity.managedObject+json,,,{""name"":""Test Device"", ""type"":""com_example_TestDevice"", ""c8y_IsDevice"":{}}
11,201,, "$c8y_IsDevice", "$.id"
```

SMARTREST REAL-TIME NOTIFICATIONS

All available real-time notification endpoints and channels of the Cumulocity platform are also available in a SmartREST syntax. See the [Real-time notification API](#) to understand the general functionality of the [Bayeux protocol](#) and to get an overview of our available endpoints and channels for real-time notifications.

USING REAL-TIME NOTIFICATIONS WITH SMARTREST

To tell the Cumulocity platform that the real-time notifications should use SmartREST all requests send to the URL must contain the **X-Id** header.

Message identifiers

Message identifier	Message parameters	Description
80	<i>None</i>	Initial handshake that will return a unique bayeux clientId.
81	clientId,channel	Subscribe for the given channel.
82	clientId,channel	Unsubscribe for the given channel.
83	clientId	Establish connection for receiving the notifications (long-polling).
84	clientId	Disconnect the client from the server.

Handshake

Example request:

```
80
```

Example response:

```
Un1q31d3nt1f13r
```

Subscribe

Example request:

```
81,Un1q31d3nt1f13r,/mychannel
```

Example response:

Unless there is an error there is no specific response for the subscribe

Unsubscribe

Example request:

```
82,Un1q31d3nt1f13r,/mychannel
```

Example response:

Unless there is an error there is no specific response for the unsubscribe

Connect

Example request:

```
83,Un1q31d3nt1f13r
```

Example response:

The response is formed by the response templates registered via SmartREST for the **X-Id**. Every received notification via real-time will be parsed with the available templates and every matching template will be returned as response for the connect request.

Keep-Alive:

The Cumulocity platform will send every 10 minutes a space character through an open long-polling connection to detect connection loss. A response for a connect that has been open for a longer time could contain leading space characters in the first line of the response.

Disconnect

Example request:

```
84,Un1q31d3nt1f13r
```

Example response:

Unless there is an error there is no specific response for the disconnect

The advice response

The bayeux protocol has a special fragment to tell the client about the recommended settings for timeout of a connection, interval between connect requests and the policy for the follow up after a response for a connect. The advice will be communicated via SmartREST also as a separate line in the response and can be contained in any response of the above requests.

Response Structure:

```
86,<timeout>,<interval>,<reconnect policy>
```

Timeout and interval will be numbers defining the time in milliseconds. The reconnect policy can be one of three values:

- none: do not reconnect after the response from a connect.
- retry: do reconnect after the response from a connect.
- handshake: start with a new handshake (for example because the clientId is invalid / server has closed session).

An advice response line does not need to have every value filled

Example:

```
86,,10000,retry
```

SUBSCRIBING WITH MULTIPLE TEMPLATES

If your device uses multiple templates (for example, child devices have a different template than the parent) it is possible to add these templates to your subscribe request. The server will then use all templates (from header and subscribe statement) to parse the responses.

Example request:

```
POST /notification/operations HTTP/1.0
Authorization: Basic ...
X-Id: mytemplate1

81,Un1q31d3nt1f13r,/mychannel,mytemplate2,mytemplate3
```

In this case multiple templates are used and the response will contain an additional line that indicates which lines are parsed with which templates:

```
87,{number of parsed rows},{X-Id used to parse the following rows}
```

Example response:

```
HTTP/1.0 200 OK

87,2,mytemplate1
100,myvalue
101,myvalue2
87,1,mytemplate3
100,myvalue3
```

BUILT-IN MESSAGES

SmartREST has a variety of built-in messages.

REQUEST MESSAGES

Message identifier	Message parameters	Description
10	Template message identifier Method Resource identifier Content MIME type Accept MIME type Placeholder Request parameters Template string	Represents a request template. If this message occurs in the body, the whole body is treated as a <i>SmartREST</i> template and thus, all messages besides 10 and 11 will yield an error.
11	Template message identifier Base JSON path Conditional JSON path Value JSON paths	Represents a response template. If this message occurs in the body, the whole body is treated as a <i>SmartREST</i> template and thus, all messages besides 10 and 11 will yield an error.
15	X-Id	Defines which X-Id to use for the following lines. You must not use the X-Id header when using this line.

Message identifier	Message parameters	Description
61	Device MO GId	Poll device credentials during device bootstrapping process. No X-Id header must be present and the device bootstrap authorization must be used.
80	<i>None</i>	Initial handshake that will return a unique bayeux clientId. SmartREST real-time notifications.
81	clientId,channel	Subscribe for the given channel. SmartREST real-time notifications.
82	clientId,channel	Unsubscribe for the given channel. SmartREST real-time notifications.
83	clientId	Establish connection for receiving the notifications (long-polling). SmartREST real-time notifications.
84	clientId	Disconnect the client from the server. SmartREST real-time notifications.

RESPONSE MESSAGES

Message identifier	Message parameters	Description
20	<i>SmartREST</i> Template MO GId	Echo response message. Template was found or has been created and everything is OK.
40	<i>None</i>	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	Line number <i>HTTP</i> response code	Server error. This message occurs when an error happened between the <i>SmartREST</i> proxy and the platform.
70	Line number Unique device identifier Tenant ID Username Password	Device bootstrap polling response with credentials.
86	timeout,interval,reconnect policy	Settings advice for the client using SmartREST real-time notifications.
87	amount of lines, X-Id	Indicates which X-Id was used to create the amount of following response lines.

Error messages

Message identifier	Error message
41	Cannot create templates for already existing template object

Message identifier	Error message
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}

HANDLING OF IDS

CONCEPT OF ID-LESS COMMUNICATION

The MQTT implementation of Cumulocity is specifically designed for device communication. Therefore, it tries to remove as much unnecessary logic from the client side as possible.

Using the REST (or SmartREST) protocol requires to know the ID of every object, alarm and operation you want to update. Hence, a client must keep a state of these IDs. For example, if it creates an alarm, it must know the ID of the alarm so it can clear it afterwards.

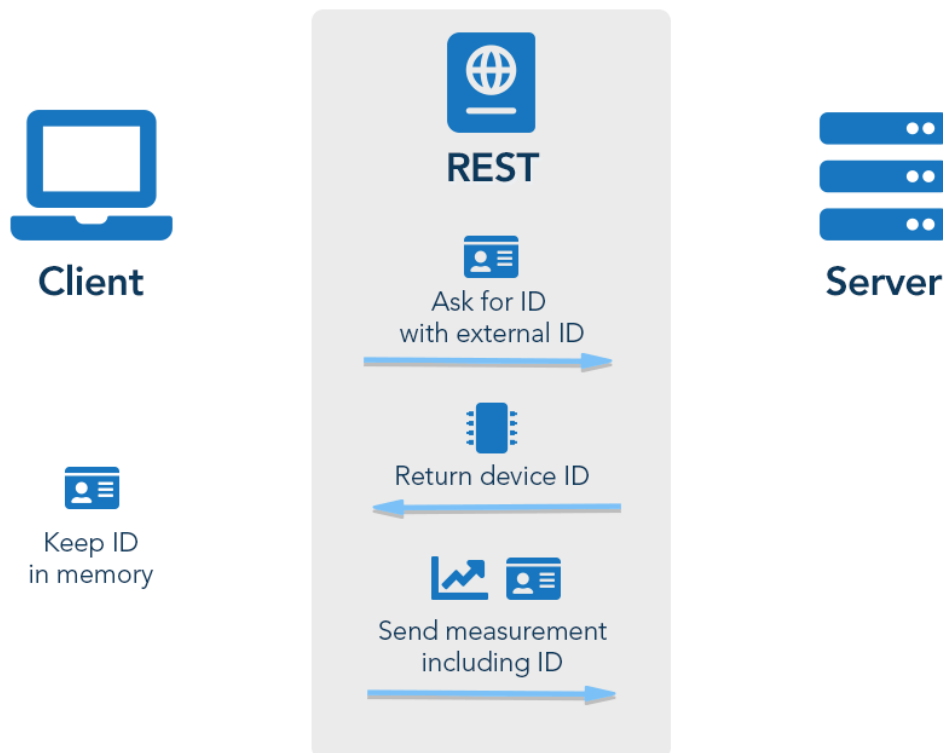
With the MQTT implementation we want to reduce the logic required on the device to do such actions and move the logic to the server.

Example 1: ID of the device

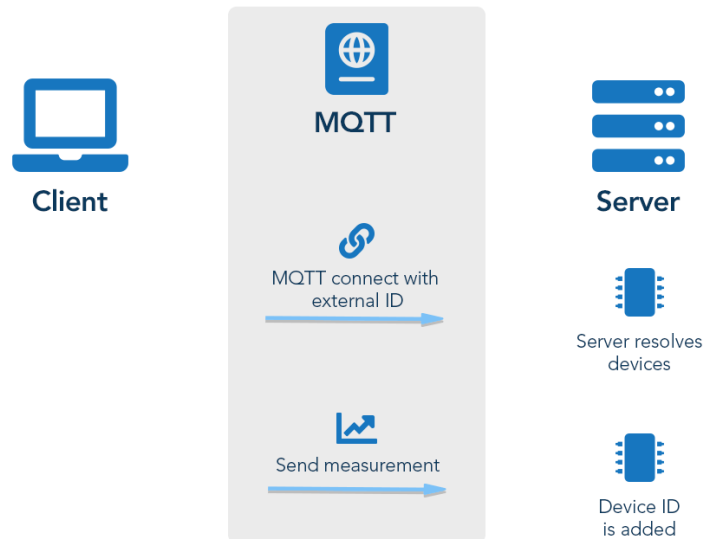
You must know the ID of the device object to update data in it via REST. Also, this ID is required for every other data that needs to be associated with the device, for example, the source in a measurement, alarm or event.

To remove the necessity of persisting the ID on the device, Cumulocity offers the identity API where you can link external IDs (for example a serial number) to the object, so you can query the ID any time.

A typical device start looks like this:



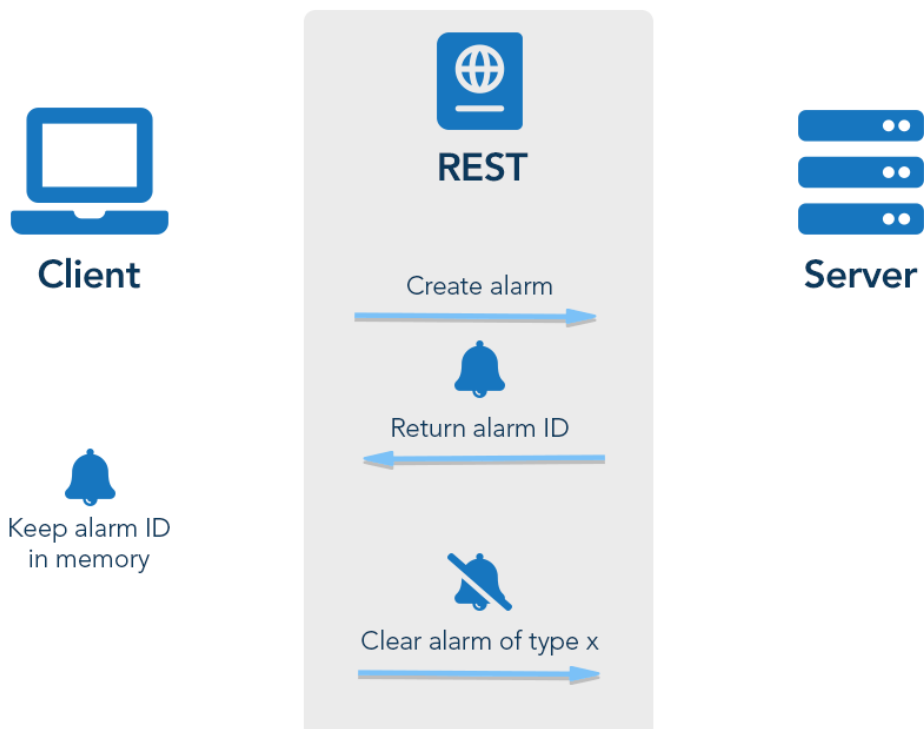
With MQTT, we automatically use the identity API with the MQTT clientId. This removes the necessity to tell the ID to the device, and because the client sends also the other data on this connection, we can associate every measurement, alarm, event, and so on with the correct device.



Example 2: ID of alarms

When a client creates an alarm using the REST API, it must ensure that it gets the ID of the alarm that was generated by Cumulocity in return.

The client will need this ID to later update the alarm, for example, to status CLEARED, if the alarm is not active anymore.



In Cumulocity, a device can only have a single alarm per type in status ACTIVE. If it creates another alarm with the same type, it will get de-duplicated. Therefore, the MQTT implementation uses the type of an alarm as identifier. The client must only send which type of alarm has been resolved, and the server will find the correct alarm object.



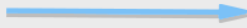
Client



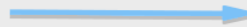
MQTT



Create alarm of type x



Clear alarm of type x



Server



Find active alarm
of type x for device

JSON VIA MQTT

INFO

JSON via MQTT is meant as an addition to a device that is connected via SmartREST. This is not a standalone interface.

This section describes the JSON payload format that can be used with the Cumulocity MQTT implementation.

Compared to SmartREST 2.0 – which only works with fixed templates – JSON's support for MQTT was designed to combine the payload flexibility of our REST API with the low protocol overhead of MQTT.

The SmartREST way should still be the preferred way if it is important to reduce your payload to the minimum (mobile traffic, low capability device).

TOPIC STRUCTURE

The topic structure in JSON MQTT is quite similar to the REST endpoints. The main difference is in the additional action part which is included in the topic.

To publish messages:

```
<api>/<resource>/<action>/<resource_id>
```

To publish messages in transient mode:

```
t/<api>/<resource>/<action>/<resource_id>
```

To publish messages in quiescent mode:

```
q/<api>/<resource>/<action>/<resource_id>
```

To publish messages in CEP mode:

```
c/<api>/<resource>/<action>/<resource_id>
```

INFO

<resource_id> is not required for every **<action>**. See the examples below.

Refer to [Processing mode](#) for more information about transient, quiescent and CEP data processing.

Topic actions

The action in the topic corresponds to the HTTP methods combined with the content-type header.

The following actions are available:

- create - corresponds to HTTP POST
- createBulk - corresponds to HTTP POST with the content-type header value set to collection media type, for example

`application/vnd.com.nsn.cumulocity.measurementCollection+json; charset=UTF-8; ver=0.9`

- update - corresponds to HTTP PUT
- delete - corresponds to HTTP DELETE

SUPPORTED ENDPOINT

The current JSON MQTT implementation does not cover all SmartREST 2.0 operations, so for example the whole [device bootstrap process](#) must be done using SmartREST 2.0.

The following endpoints and actions are supported:

Endpoint	create	createBulk	update	delete
event/events	x	x	x	x
alarm/alarms	x	x	x	
measurement/measurements	x	x		x
inventory/managedObjects	x		x	
inventory/managedObjects/{id}/childDevices	x			

If the operation is not supported, a proper error message will be sent to the `error` topic.

For all of the above endpoints, you can use the same payload like in the REST API. The only difference is in the "source" field - in REST this field is mandatory while for JSON MQTT there is no need to set the device ID here. The source device ID will automatically be resolved based on the MQTT client ID. This value will always be used no matter if something is already defined there.

EXAMPLES

Create new event

Publish a message on topic `event/events/create` with payload:

```
{
  "type": "TestEvent",
  "text": "sensor was triggered",
  "time": "2014-03-03T12:03:27.845Z"
}
```

Create many events

Publish a message on topic `event/events/createBulk` with payload:

```
{
  "events": [
    {
      "type": "TestEvent1",
      "text": "sensor was triggered",
      "time": "2014-03-03T12:03:27.845Z"
    },
    {
      "type": "TestEvent2",
      "text": "sensor was triggered",
      "time": "2014-03-04T12:03:27.845Z"
    }
  ]
}
```

Update event

Publish a message on topic `/event/events/update/<event_id>` with payload:

```
{
  "text": "new text"
}
```

Delete event

Publish a message on topic `/event/events/delete/<event_id>` with empty payload.

Create a measurement data point

Publish a message on topic `measurement/measurements/create` with payload:

```
{
  "type": "c8y_TemperatureMeasurement",
  "time": "2021-09-06T17:35:14.000+02:00",
  "c8y_TemperatureMeasurement": {
    "T": {
      "value": 20,
      "unit": "C"
    }
  }
}
```

ERROR HANDLING

Use the `error` topic to subscribe for errors related to the JSON MQTT implementation. In case of invalid payload, wrong topic or any other exception, a notification will be published on this topic. The payload is in JSON format. Besides a standard error message, it also contains a message ID which helps the client in finding out which exact message was failing.

Example payload:

```
{
  "error": "undefined/validationError",
  "message": "Following mandatory fields should be included: severity,text,time",
  "messageId": 3
}
```

RECEIVING OPERATIONS

A notification client can subscribe to the `devicecontrol/notifications` topic to receive notifications of newly created operations. Initially upon subscription, all operations which are not yet forwarded will be published.

Additionally, it contains an [External ID](#), so the client can identify for which child the operation is executed.

Example notification:

```
{
  "agentId": "1",
  "creationTime": "2018-05-17T07:33:15.555Z",
  "delivery": {
    "log": [

    ],
    "status": "PENDING",
    "time": "2018-05-17T07:33:15.575Z"
  },
  "deviceId": "2",
  "id": "123",
  "status": "PENDING",
  "c8y_Command": {
    "text": "Do something"
  },
  "description": "Execute shell command",
  "externalSource": {
    "externalId": "3",
    "type": "c8y_Serial"
  }
}
```

MQTT STATIC TEMPLATES

To ease device integration Cumulocity already supports a number of static templates that can be used by any client without the need to create your own templates. These templates focus on the most commonly used messages for device management purposes.

To use the templates listed below, you must publish the messages to the topic `s/us` (`t/us` for transient processing of published content, `q/us` for quiescent processing of published content or `c/us` for CEP processing of published content. Refer to [Processing mode](#) for further information.

You must subscribe to the topic `s/ds` to receive operations with the static templates.

Templates quick reference

Click the commands below to see more information on the respective template. If a parameter is in square brackets, it is optional.

Inventory templates

- [100,createdDeviceName,deviceType](#)
- [101,createdChildId,childName,childType](#)
- [102,serviceExternalId,serviceType,serviceName,serviceStatus](#)
- [104,serviceStatus](#)
- [105 \(Get children, reply: 106,child1,child2,...\)](#)
- [107,fragmentToBeUninstalled1,fragment2,...](#)
- [110,serialNumber,hardwareModel,revision](#)
- [111,IMEI,ICCID,IMSI,MCC,MNC,LAC,cellId](#)
- [112,latitude,longitude,altitude,accuracy](#)
- [113,"configProp1=val1\nprop2=val2\n..."](#)
- [114,supportedOperation1,operation2,...](#)
- [115,currentFirmwareName,version,url](#)
- [116,currentSoftwareName1,version1,url1,name2,...](#)
- [117,requiredInterval](#)
- [118,supportedLog1,log2,...](#)
- [119,supportedConfiguration1,config2,...](#)
- [120,configType,url,filename\[,time\]](#)
- [121,profileExecuted,profileID](#)
- [122,agentName,agentVersion,agentURL,agentMaintainer](#)
- [123 \(Retrieve the internal ID, reply: "124,id"\)](#)
- [125 \(send heartbeat\)](#)
- [140,setAdvancedSWName1,AdvancedSWVersion1,AdvancedSWType1,AdvancedSWurl1,sw2,ver2,type2,url2,...](#)
- [141,appendAdvancedSWName1,AdvancedSWVersion1,AdvancedSWType1,AdvancedSWurl1,sw2,ver2,type2,url2,...](#)
- [142,deleteAdvancedSWName1,AdvancedSWVersion1,sw2,ver2,...](#)
- [143,supportedSoftwareType1,supportedSoftwareType2,...](#)
- [150,cloudRemoteAccessProtocol1,cloudRemoteAccessProtocol2,...](#)

Measurement templates

- [200,fragment,series,value\[,unit,time\]](#)
- [201,type\[,time\],fragment1,series1,value1\[,unit1\],...](#)
- [210,rsi,ber\[,time\]](#)
- [211,temperature\[,time\]](#)
- [212,battery\[,time\]](#)

Alarm templates

- [301,criticalAlarmType\[,text\]\[,time\]](#)
- [302,majorAlarmType\[,text\]\[,time\]](#)
- [303,minorAlarmType\[,text\]\[,time\]](#)
- [304,warningAlarmType\[,text\]\[,time\]](#)
- [305,alarmType,newSeverity](#)
- [306,alarmTypeToBeCleared](#)
- [307,alarmType,fragmentToBeRemoved1,fragment2,...](#)

Event templates

- [400,eventType,text\[,time\]](#)
- [401,latitude,longitude,altitude,accuracy\[,time\]](#)
- [402,latitude,longitude,altitude,accuracy\[,time\] \(incl. inv. update\)](#)
- [407,eventType,fragmentToBeRemoved1,fragment2,...](#)

Operation templates

- 500 (get pending)
- 501,typeToSetToExecuting
- 502,typeToSetToFailed,failureReason
- 503,typeToSetToSuccessful,parameters
- 504,operationId
- 505,operationId,failureReason
- 506,operationId,parameters
- 507,typeToSetToExecuting,failureReason
- 510,serial (restart)
- 511,serial,commandToExecute
- 513,serial,configurationText
- 515,serial,firmwareToBeInstalled,version,url
- 516,serial,softwareToBeInstalled1,version1,url1,sw2,ver2,url2,...
- 517,serial,measurementToBeSent
- 518,serial,relayStatusToBeSet [OPEN/CLOSED]
- 519,serial,relay1Status,relay2Status,...
- 520,serial (upload your current configuration)
- 521,serial,url (download configuration)
- 522,serial,logFileToBeSend,start,stop,searchText,maxLines
- 523,serial,communicationMode (SMS/IP)
- 524,serial,url,configType
- 525,serial,currentFirmwareName,version,url,dependency
- 526,serial,configType
- 527,serial,firmwareMarker,name,version,url,isPatch,dependency,softwareMarker,name,version,url,action,configurationMarker,url,type
- 528,serial,softwareToBeUpdated1,version1,url1,action1,sw2,ver2,url2,action2,...
- 529,serial,softwareToBeUpdated1,version1,type1,url1,action1,sw2,ver2,type2,url2,action2,...
- 530,serial,hostname,port,connectionKey
- 531,serial,firmwareMarker,name,version,url,isPatch,dependency,softwareMarker,name,version,type,url,action,configurationMarker,url,type
- 532,serial,PARAM_DEF_0,PARAM_DEF_1,...,PARAM_DEF_N

Device parameter templates

- 408,[text],[time],[change detect],path1,type1,value1[,path2,type2,value2,...]

Platform capabilities templates

- 600 (Retrieve the platform version, reply: "601, platformVersion")
- 602 (Retrieve a list of valid SmartREST template IDs, reply: "603,comma-separated-list-of-SmartREST-template-IDs")

The client can receive the following templates when subscribing to s/ds .

Inventory templates

- 106,child1,child2,...
- 124,id

Operation templates

- 510,serial (restart)
- 511,serial,commandToExecute
- 513,serial,configurationText
- 515,serial,firmwareToBeInstalled,version,url
- 516,serial,softwareToBeInstalled1,version1,url1,sw2,ver2,url2,...
- 517,serial,measurementToBeSent
- 518,serial,relayStatusToBeSet [OPEN/CLOSED]
- 519,serial,relay1Status,relay2Status,...
- 520,serial (upload your current configuration)
- 521,serial,url (download configuration)
- 523,serial,communicationMode (SMS/IP)
- 524,serial,url,configType
- 525,serial,currentFirmwareName,version,url,dependency
- 526,serial,configType
- 527,serial,firmwareMarker,name,version,url,isPatch,dependency,softwareMarker,name,version,url,action,configurationMarker,url,type
- 528,serial,softwareToBeUpdated1,version1,url1,action1,sw2,ver2,url2,action2,...
- 529,serial,softwareToBeUpdated1,version1,type1,url1,action1,sw2,ver2,type2,url2,action2,...
- 530,serial,hostname,port,connectionKey
- 532,serial,parameterKey1,type1,value1,...

Device parameter templates

- 601, platformVersion

AUTOMATIC DEVICE CREATION

The topic for static templates supports an automatic creation of devices. Whenever there is no child associated with your MQTT ClientID and you send data, Cumulocity will automatically create a device for the MQTT ClientID. If you want to create the device on your own, your first message must be the device creation. In this case Cumulocity will create the device from the template.

The automatic creation of devices is also supported for 1st level child devices. For child devices on a deeper level, you must use the template for creating a child device by sending it to the topic of the child device under the one you want to place the new child.

HANDLING NON-MANDATORY PARAMETERS

If a parameter is not declared as mandatory, the client can send an empty string in that place.

```
100,,myType
```

Tailing commas is not required. The two lines below result in the same message.

```
100,,
100
```

PUBLISH TEMPLATES

The following templates can be used to publish data on the topics `s/us` as well as `t/us`. Refer to [Processing mode](#) for more information about the `t/` topic for transient data processing.

Inventory templates (1xx)

Device creation (100)

Create a new device for the serial number in the inventory if not yet existing. An externalId for the device with type `c8y_Serial` and the device identifier of the MQTT clientId as value will be created.

Position	Parameter	Mandatory	Type	Default value
1	device name	NO	String	MQTT Device <serialNumber>
2	device type	NO	String	c8y_MQTTDevice

Example

```
100,myDevice,myType
```

Child device creation (101)

Create a new child device for the current device. The newly created object will be added as child device. Additionally, an externalId for the child will be created with type `c8y_Serial` and the value a combination of the serial of the root device and the unique child ID.

Position	Parameter	Mandatory	Type	Default value
1	unique child ID	YES	String	
2	device name	NO	String	MQTT Device <serialNumber>
3	device type	NO	String	c8y_MQTTChildDevice
4	child device marker	NO	Boolean	true

Example

```
101,uniqueChildId,myChildDevice,myChildType,true
```

Service creation (102)

Create a new software service for given device.

Position	Parameter	Mandatory	Type
1	service unique external id	YES	String
2	service type	YES	String
3	service name	YES	String
4	service status	YES	String

Example

```
102,myDatabaseDevice,systemd,DatabaseService,up
```

Service status update (104)

Set a status for given software service.

Position	Parameter	Mandatory	Type
1	service status	YES	String

Example

```
104,up
```

Get child devices (105)

Trigger the sending of child devices of the device.

Example

```
105
```

Clear device's fragment (107)

Remove one or more fragments from a device.

Position	Parameter	Mandatory	Type
1...	fragmentName	YES	String

Example

```
107,c8y_Position,c8y_Configuration
```

Configure Hardware (110)

Update the hardware properties of the device.

Position	Parameter	Mandatory	Type
1	serialNumber	NO	String
2	model	NO	String
3	revision	NO	String

Example

```
110,1234567890,myModel,1.2.3
```

Configure Mobile (111)

Update the mobile properties of the device.

Position	Parameter	Mandatory	Type
1	imei	NO	String
2	iccid	NO	String
3	imsi	NO	String
4	mcc	NO	String
5	mnc	NO	String
6	lac	NO	String
7	cellId	NO	String

Example

```
111,1234567890,,54353
```

Configure Position (112)

Update the position properties of the device.

Position	Parameter	Mandatory	Type
1	latitude	NO	Number
2	longitude	NO	Number
3	altitude	NO	Number
4	accuracy	NO	Integer

Example

```
112,50.323423,6.423423
```

Set Configuration (113)

Update the configuration properties of the device.

Position	Parameter	Mandatory	Type
1	configuration	NO	String

Example

```
113,"val1=1\nval2=2"
```

Set supported operations (114)

Set the supported operations of the device.

INFO

For details on all options and parameters used to specify the support of particular functionalities, refer to the [Fragment library](#).

Position	Parameter	Mandatory	Type
1...	List of supported operations	NO	String

Example

```
114,c8y_Restart,c8y_Configuration,c8y_SoftwareList
```

INFO

If you want to remove an item from the supported operations list, send a new 114 request with the updated list, for example, `114, c8y_Restart,c8y_Configuration` in order to remove `c8y_SoftwareList` after the request from the example above.

Set firmware (115)

Set the firmware installed on the device.

Position	Parameter	Mandatory	Type
1	name	NO	String
2	version	NO	String
3	url	NO	String

Example

```
115,firmwareName,firmwareVersion,firmwareUrl
```

Set software list (116)

Set the list of software installed on the device.

Position	Parameter	Mandatory	Type
1...	List of 3 values per software	NO	(n/a)
1.1	name	NO	String
1.2	version	NO	String
1.3	url	NO	String

Example

```
116,software1,version1,url1,software2,,url2,software3,version3
```

Set required availability (117)

Set the required interval for availability monitoring as an integer value representing minutes. For more information, see *c8y_RequiredAvailability* in [Device availability](#). This will only set the value if it does not exist. Values entered, for example, through the UI, are not overwritten.

Position	Parameter	Mandatory	Type
1	Required interval	NO	Integer

Example

```
117,60
```

Set supported logs (118)

Set the supported logs of the device.

Position	Parameter	Mandatory	Type
1...	List of supported logs	NO	String

Example

```
118,ntcagent,dmesg,logread
```

Set supported configurations (119)

Set the supported configurations of the device.

Position	Parameter	Mandatory	Type
1...	List of supported configurations	NO	String

Example

```
119,modbus,system
```

Set currently installed configuration (120)

Set currently installed configuration of the device.

Position	Parameter	Mandatory	Type	Default value
1	Configuration type	YES	String	
2	Configuration file download URL	YES	String	
3	File name	NO	String	Configuration type
4	Date and time when the configuration was applied	NO	Date	Current date and time

Example

```
120,myType,http://www.my.url,config.bin,2020-07-22T17:03:14.000+02:00
```

Set device profile that is being applied (121)

Set device profile that is being applied to the device.

Position	Parameter	Mandatory	Type	Default value
1	Profile executed	YES	String	
2	Profile ID	NO	String	Profile ID from the oldest EXECUTING device profile operation

Example

```
121,true,8473
```

Set device agent information (122)

Allows a device to provide information about the agent running on it.

Position	Parameter	Mandatory	Type	Default value
1	Name of the agent	YES	String	
2	Version of the agent	YES	String	
3	The agent URL	NO	String	
4	Maintainer of the agent	YES	String	

Example

```
122,thin-edge.io,0.6,https://thin-edge.io,Cumulocity
```

Send heartbeat (125)

Sends a heartbeat from the device to update its availability status.

Example

```
125
```

Set advanced software list (140)

Sets the list of advanced software installed on the device. Any existing list will be overwritten.

Position	Parameter	Mandatory	Type	Default value
1	Name of the software	YES	String	
2	Version of the software	YES	String	
3	Type of the software	NO	String	
4	URL of the software	NO	String	

Example

```
140,docker,3.2.1,systemd,https://www.docker.com/,nginx,1.6,container,https://www.nginx.com/
```

Get the device managed object ID (123)

Retrieve the ID of the device managed object.

Example

```
123
```

Append advanced software items (141)

Appends advanced software items to the list that exists for the device.

Position	Parameter	Mandatory	Type	Default value
1	Name of the software	YES	String	
2	Version of the software	YES	String	
3	Type of the software	NO	String	
4	URL of the software	NO	String	

Example

```
141,docker,3.2.1,systemd,https://www.docker.com/,nginx,1.6,container,https://www.nginx.com/
```

Remove advanced software items (142)

Removes advanced software items from the list that exists for the device.

Position	Parameter	Mandatory	Type	Default value
1	Name of the software	YES	String	
2	Version of the software	YES	String	

Example

```
142,docker,3.2.1,nginx,1.6
```

Set supported software types (143)

Sets the supported software types of the device. Ignores empty elements. An empty list removes the `c8y_SupportedSoftwareTypes` fragment entirely.

Position	Parameter	Mandatory	Type
1...	List of supported software types	NO	String

Example

```
143,yum,docker
```

Set supported remote access protocols (150)

Sets the list of Cloud Remote Access protocols supported by the device. Possible values are **SSH** , **TELNET** , **VNC** and **PASSTHROUGH** . Empty elements are ignored. An empty list removes the **c8y_RemoteAccessSupportedProtocols** fragment entirely.

Position	Parameter	Mandatory	Type
1...	List of supported protocols	NO	String

Examples

```
150,ssh,vnc
```

Measurement templates (2xx)**Create custom measurement (200)**

Create a measurement with a given fragment and series.

Position	Parameter	Mandatory	Type	Default value
1	fragment	YES	String	
2	series	YES	String	
3	value	YES	Number	
4	unit	NO	String	
5	time	NO	Date	Current server time

Example

```
200,c8y_Temperature,T,25
```

Create a custom measurement with multiple fragments and series (201)

Create a measurement with multiple fragments and series.

Position	Parameter	Mandatory	Type	Default value
1	type	YES	String	
2	time	NO	Date	
3	List of 4 values per fragment-series combination	YES	(n/a)	
3.1	fragment	YES	String	
3.2	series	YES	String	

Position	Parameter	Mandatory	Type	Default value
3.3	value	YES	Number	
3.4	unit	NO	String	

Example

```
201,KamstrupA220Reading,2022-03-19T12:03:27.845Z,c8y_SinglePhaseEnergyMeasurement,A+:1,1234,kWh,c8y_SinglePhaseEnergyMeasurement,A-:1,2345,kWh,c8y_ThreePhaseEnergyMeasurement,A+:1,123,kWh,c8y_ThreePhaseEnergyMeasurement,A+:2,234,kWh,c8y_ThreePhaseEnergyMeasurement,A+:3,345,kWh
```

Create signal strength measurement (210)

Create a measurement of type `c8y_SignalStrength`.

Position	Parameter	Mandatory	Type	Default value
1	rsi value	YES, if 2 not set	Number	
2	ber value	YES, if 1 not set	Number	
3	time	NO	Date	Current server time

Example

```
210,-90,23,2016-06-22T17:03:14.000+02:00
```

Create temperature measurement (211)

Create a measurement of type `c8y_TemperatureMeasurement`.

Position	Parameter	Mandatory	Type	Default value
1	temperature value	YES	Number	
2	time	NO	Date	Current server time

Example

```
211,25,2016-06-22T17:03:14.000+02:00
```

Create battery measurement (212)

Create a measurement of type `c8y_Battery`.

Position	Parameter	Mandatory	Type	Default value
1	battery value	YES	Number	
2	time	NO	Date	Current server time

Example

```
212,95,2016-06-22T17:03:14.000+02:00
```

Alarm templates (3xx)

[Create CRITICAL alarm \(301\)](#)

Create a CRITICAL alarm.

Position	Parameter	Mandatory	Type	Default value
1	type	YES	String	
2	text	NO	String	Alarm of type alarmType raised
3	time	NO	Date	Current server time

Example

```
301,c8y_TemperatureAlarm
```

[Create MAJOR alarm \(302\)](#)

Create a MAJOR alarm.

Position	Parameter	Mandatory	Type	Default value
1	type	YES	String	
2	text	NO	String	Alarm of type alarmType raised
3	time	NO	Date	Current server time

Example

```
302,c8y_TemperatureAlarm,"This is an alarm"
```

[Create MINOR alarm \(303\)](#)

Create a MINOR alarm.

Position	Parameter	Mandatory	Type	Default value
1	type	YES	String	
2	text	NO	String	Alarm of type alarmType raised
3	time	NO	Date	Current server time

Example

```
303,c8y_TemperatureAlarm
```

[Create WARNING alarm \(304\)](#)

Create a WARNING alarm.

Position	Parameter	Mandatory	Type	Default value
1	type	YES	String	
2	text	NO	String	Alarm of type alarmType raised
3	time	NO	Date	Current server time

Example

```
304,c8y_TemperatureAlarm,,2013-06-22T17:03:14.000+02:00
```

Update severity of existing alarm (305)

Change the severity of an existing alarm.

Position	Parameter	Mandatory	Type
1	type	YES	String
2	severity	YES	String

Example

```
305,c8y_TemperatureAlarm,CRITICAL
```

Clear existing alarm (306)

Clear an existing alarm.

Position	Parameter	Mandatory	Type
1	type	YES	String

Example

```
306,c8y_TemperatureAlarm
```

Clear alarm's fragment (307)

Remove one or more fragments from an alarm of a specific type.

Position	Parameter	Mandatory	Type
1	alarmType	YES	String
2...	fragmentName	YES	String

Example

```
307,c8y_TemperatureAlarm,c8y_Position,c8y_Configuration
```

Event templates (4xx)**Create basic event (400)**

Create an event of given type and text.

Position	Parameter	Mandatory	Type	Default value
1	type	YES	String	
2	text	YES	String	
3	time	NO	Date	Current server time

Example

```
400,c8y_MyEvent,"Something was triggered"
```

Create location update event (401)

Create typical location update event containing `c8y_Position`.

Position	Parameter	Mandatory	Type	Default value
1	latitude	NO	Number	
2	longitude	NO	Number	
3	altitude	NO	Number	
4	accuracy	NO	Number	
5	time	NO	Date	Current server time

Example

```
401,51.151977,6.95173,67
```

Create location update event with device update (402)

Create typical location update event containing `c8y_Position`. Additionally the device will be updated with the same `c8y_Position` fragment.

Position	Parameter	Mandatory	Type	Default value
1	latitude	NO	Number	
2	longitude	NO	Number	
3	altitude	NO	Number	
4	accuracy	NO	Number	
5	time	NO	Date	Current server time

Example

```
402,51.151977,6.95173,67
```

Clear event's fragment (407)

Remove one or more fragments from an event of a specific type.

Position	Parameter	Mandatory	Type
1	eventType	YES	String
2...	fragmentName	NO	String

Example

```
407,c8y_MyEvent,c8y_Position,c8y_Configuration
```

Create device parameter update events (408)

Create parameter update events for a device. This event always has the type `c8y_ParameterUpdate` and contains the fragments and properties specified in the payload. If `change detect` is enabled, events are only created if the given state in the event was not already known at the time.

Position	Parameter	Mandatory	Type	Default value
1	text	NO	String	Parameter 'c8y_RelayStatus' was updated
2	time	NO	Date	Current server time
3	change detect	NO	Boolean	false
4	List of 3 values per path-value	YES	(n/a)	
4.1	path	YES	String	
4.2	type	YES	String	
4.3	value	NO	String	null

Example

```
408,...,c8y_RelayStatus.left,BOOLEAN,true,c8y_RelayStatus.right,BOOLEAN,false
```

Operation templates (5xx)

Get PENDING operations (500)

Trigger the sending of all PENDING operations for the agent.

Example

```
500
```

Set operation to EXECUTING (501)

Set the oldest PENDING operation with given fragment to EXECUTING.

Position	Parameter	Mandatory	Type
1	fragment	YES	String

Example

```
501,c8y_Restart
```

Set operation to FAILED (502)

Set the oldest EXECUTING operation with given fragment to FAILED.

Position	Parameter	Mandatory	Type
1	fragment	YES	String
2	failureReason	NO	String

Example

```
502,c8y_Restart,"Could not restart"
```

[Set operation to SUCCESSFUL \(503\)](#)

Set the oldest EXECUTING operation with given fragment to SUCCESSFUL.

It enables the device to send additional parameters that trigger additional steps based on the type of operation sent as fragment (see [Section Updating operations](#)).

Position	Parameter	Mandatory	Type
1	fragment	YES	String
2...	parameters	NO	String

Example

```
503,c8y_Restart
```

[Set operation to EXECUTING \(504\)](#)

Set the operation with the given ID to EXECUTING. The operation must exist and must have the requesting device as the source.

Position	Parameter	Mandatory	Type
1	operationId	YES	String

Example

```
504,123
```

[Set operation to FAILED \(505\)](#)

Set the operation with the given ID to FAILED. The operation must exist and must have the requesting device as the source.

Position	Parameter	Mandatory	Type
1	operationId	YES	String
2	failureReason	NO	String

Example

```
505,123,"Could not restart"
```

[Set operation to SUCCESSFUL \(506\)](#)

Set the operation with given ID to SUCCESSFUL. The operation must exist and must have the requesting device as the source.

This may let the device send additional parameters that trigger further steps based on the type of the operation, also see [Updating operations](#).

Position	Parameter	Mandatory	Type
1	operationId	YES	String
2...	parameters	NO	String

Example

```
506,123
```

Set EXECUTING operations to FAILED (507)

Set EXECUTING operations with a given fragment to FAILED. If the fragment parameter is empty, all EXECUTING operations are set to FAILED.

Position	Parameter	Mandatory	Type
1	fragment	NO	String
2	failureReason	NO	String

Example

```
507,c8y_Restart,"Unexpected device restart"
```

Platform capabilities templates (6xx)**Get platform version (600)**

Retrieve the platform version. Used to evaluate if the platform supports API interfaces or what kind of behaviour to expect from requests to them.

Example

```
600
```

List of valid SmartREST templates (602)

Retrieve the list of valid SmartREST template IDs associated with a tenant.

Example

```
602
```

SUBSCRIBE TEMPLATES**Inventory templates (1xx)****Get children of device (106)**

List all children of the device.

Position	Parameter	Type
1...	child	String

Example

```
106,child1,child2,child3
```

Get the device managed object ID (124)

Retrieve the ID of the device managed object.

Position	Parameter	Type
1	id	String

Example

```
124,12345
```

Operation templates (5xx)

All operation responses have the same base structure, leading with the message ID and followed by the ID of either the root device or a child which should handle the operation.

Restart (510)

Restart a device.

Example

```
510,DeviceSerial
```

Command (511)

Run the command being sent in the operation.

Position	Parameter	Type
1	Command text	String

Example

```
511,DeviceSerial,execute this
```

Configuration (513)

Set the configuration being sent in the operation.

Position	Parameter	Type
1	configuration	String

Example

```
513,DeviceSerial,"val1=1\nval2=2"
```

Firmware (515)

Install the firmware from the url.

Position	Parameter	Type
1	firmware name	String
2	firmware version	String
3	url	String

Example

```
515,DeviceSerial,myFirmware,1.0,http://www.my.url
```

Software list (516)

Install the software sent in the operation.

Position	Parameter	Type
1...	List of 3 values per software	(n/a)
1.1	name	String
1.2	version	String
1.3	url	String

Example

```
516,DeviceSerial,softwareA,1.0,url1,softwareB,2.0,url2
```

Measurement request operation (517)

Send the measurements specified by the request name.

Position	Parameter	Type
1	request name	String

Example

```
517,DeviceSerial,LOGA
```

Relay (518)

Open or close the relay.

Position	Parameter	Type
1	Relay state	String

Example

```
518,DeviceSerial,OPEN
```

RelayArray (519)

Open or close the relays in the array.

Position	Parameter	Type
1...	List of relay state	String

Example

```
519,DeviceSerial,OPEN,CLOSE,CLOSE,OPEN
```

Upload configuration file (520)

Upload the current configuration to Cumulocity.

Example

```
520,DeviceSerial
```


[Download configuration file \(521\)](#)

Download a configuration file from the URL.

Position	Parameter	Type
1	url	String

Example

```
521,DeviceSerial,http://www.my.url
```

[Logfile request \(522\)](#)

Upload a log file for the given parameters.

Position	Parameter	Type
1	Log file name	String
2	Start date	Date
3	End date	Date
4	Search text	String
5	Maximum lines	Integer

Example

```
522,DeviceSerial,logfileA,2013-06-22T17:03:14.000+02:00,2013-06-22T18:03:14.000+02:00,ERROR,1000
```

[Communication mode \(523\)](#)

Change the communication mode.

Position	Parameter	Type
1	mode	String

Example

```
523,DeviceSerial,SMS
```

[Download configuration file with type \(524\)](#)

Download a configuration file from the URL with type.

Position	Parameter	Type
1	URL	String
2	configuration type	String

Example

```
524,DeviceSerial,http://www.my.url,type
```

[Firmware from patch \(525\)](#)

Install the firmware from the patch.

Position	Parameter	Type
1	firmware name	String
2	firmware version	String
3	URL	String
4	dependency	String

Example

```
525,DeviceSerial,firmwareName,1.0,http://www.my.url,dependency
```

[Upload configuration file with type \(526\)](#)

Configuration is uploaded from Cumulocity to the device with type.

Position	Parameter	Type
1	configuration type	String

Example

```
526,DeviceSerial,type
```

[Set device profiles \(527\)](#)

Set the device profiles

Position	Parameter	Type
1	firmware marker	(n/a)
1...	5 values of firmware	(n/a)
1.1	firmware name	String
1.2	firmware version	String
1.3	firmware URL	String
1.4	firmware isPatch	String
1.5	firmware dependency	String
2	software marker	(n/a)
2...	List of 4 values per software	(n/a)
2.1	software name	String
2.2	software version	String
2.3	software URL	String

Position	Parameter	Type
2.4	software action	String
3	configuration marker	(n/a)
3...	List of 2 values per configuration	(n/a)
3.1	configuration URL	String
3.2	configuration type	String

Example

```
527,DeviceSerial,$FW,firmwareName,1.0,http://www.my.url,true,dependency,$SW,softwareA,1.0,http://www.my.url1,action1,softwareB,2.0,http://www.my.url2,action2,$CONF,http://www.my.url1,type1,http://www.my.url2,type2
```

[Update software \(528\)](#)

Update the software installed on the device.

Position	Parameter	Type
1...	List of 4 values per software	(n/a)
1.1	name	String
1.2	version	String
1.3	URL	String
1.4	action	String

Example

```
528,DeviceSerial,softwareA,1.0,url1,install,softwareB,2.0,url2,install
```

i INFO

The action can either be `install` or `delete`.

When the `install` action is received, the device agent ensures that the software will appear in the `c8y_SoftwareList` fragment of the device after it has completed the installation. The agent will also determine if there is a previous version of the software and replace it with the new version, resulting in an update.

When the `delete` action is received, the device agent ensures that the software will no longer appear in the `c8y_SoftwareList` fragment of the device after the software update operation has completed.

[Update advanced software \(529\)](#)

Update the software installed on the device.

Position	Parameter	Type
1...	List of 5 values per software	(n/a)

Position	Parameter	Type
1.1	name	String
1.2	version	String
1.3	type	String
1.4	URL	String
1.5	action to be performed	String

Example

```
529,DeviceSerial,softwareA,1.0,url1,install,softwareB,2.0,url2,install
```

Cloud Remote Access connect (530)

Establish tunneling by Remote Access device agent.

Position	Parameter	Type
1	hostname	String
2	port	Integer
3	connection key	String

Example

```
530,DeviceSerial,10.0.0.67,22,eb5e9d13-1caa-486b-bdda-130ca0d87df8
```

Set device profiles with software type (531)

Set the device profiles with software type

Position	Parameter	Type
1	firmware marker	(n/a)
1...	5 values of firmware	(n/a)
1.1	firmware name	String
1.2	firmware version	String
1.3	firmware URL	String
1.4	firmware isPatch	String
1.5	firmware dependency	String
2	software marker	(n/a)
2...	list of 5 values per software	(n/a)
2.1	software name	String

Position	Parameter	Type
2.2	software version	String
2.3	software type	String
2.4	software URL	String
2.5	software action	String
3	configuration marker	(n/a)
3...	list of 2 values per configuration	(n/a)
3.1	configuration URL	String
3.2	configuration type	String

Example

```
531,DeviceSerial,$FW,firmwareName,1.0,http://www.my.url,true,dependency,$SW,softwareA,1.0,http://www.my.url1,type1,action1,softwareB,2.0,http://www.my.url2,type2,action2,$CONF,http://www.my.url1,type1,http://www.my.url2,type2
```

i INFO

When a device profile operation contains a type in one of the software entries in the list, template ID 531 is triggered. The triggering of template ID 531 is independent of template ID 527. This means that whenever template ID 531 is triggered, template ID 527 will also be triggered at the same time.

Parameter update operation (532)

Notifies the device that a **c8y_ParameterUpdate** operation has been created.

Position	Parameter	Type	Description
1..n	list of parameters	n/a	
1.1	parameter name	string	i.e. MyParameter
1.2	data type	string	"s" - string, "n" - number, "b" - boolean, "" - (empty string) null
1...	list of 3-sets per parameter	n/a	
1.1	parameter name	string	i.e. MyParameter
1.2	data type	string	"s" - string, "n" - number, "b" - boolean, "" - (empty string) null
1.3	value	string	value serialized to string or empty string for null values

Example

Operation with fragments:

```
{
  "c8y_ParameterUpdate": {},
  "c8y_ParameterUpdate_MyParameter": {},
  "c8y_ParameterUpdate_DeviceMaintainer": {},
  "c8y_ParameterUpdate_Problems": {},
  "MyParameter": true,
  "DeviceMaintainer": {
    "name": "John Smithsky",
    "contact": "12312"
  },
  "Problems": null
}
```

Will result in the message:

```
532,someSerial,MyParameter,b,true,DeviceMaintainer.name,s,John Smithsky,DeviceMaintainer.contact,s,12312,Problems,,
```

The maximum number of parameters is **100**. Operations containing more parameters cannot be pushed through this template.

Platform capabilities templates (6xx)

Return platform version (601)

Returns the platform version after a request using 600.

Position	Parameter	Type
1	version	String

Example

```
601,2025.0.0
```

Return list of valid SmartREST templates (603)

Returns the list of all available SmartREST template IDs and additional capabilities on a request using 602.

Example

```
603,100,101,102,...,603
```

UPDATING OPERATIONS

When using the template to set an operation to status **SUCCESSFUL**, it supports sending additional parameters to trigger additional calls on the server. The table below shows the operations supporting this feature and what will be done with the parameters.

Fragment	Parameters	Action triggered
c8y_Command	result	Result will be added to operation
c8y_RelayArray	relay states	Device object will be updated with the states
c8y_CommunicationMode	no parameter needed	Device object will be updated with the mode
c8y_LogfileRequest	file url	File url will be added to operation
c8y_DownloadConfigFile	(optional) timestamp	Device object will be updated with the ID of the configuration dump and the timestamp (or server time)

MQTT QUICK REFERENCE

Connection

- CONNECT d:1235:myDevice_10 acme/device_1235
Connect the device with serial "1235" and default template myDevice_10 to tenant "acme" and user "device_1235".

Topics

Publish

- PUBLISH s/us - Send a static template.
- PUBLISH s/us/5678 - Send a static template as child "5678".
- PUBLISH s/ud - Send a message using the default template (myDevice_10).
- PUBLISH s/ud/5678 - Same as above, but as child "5678".
- PUBLISH s/uc/myCommon_10 - Send a message using myCommon_10 template.
- PUBLISH s/uc/myCommon_10/5678 - Same as above, but as child "5678".

Subscribe

- SUBSCRIBE s/ds - Receive static commands.
- SUBSCRIBE s/dd - Receive commands using the default template (myDevice_10).
- SUBSCRIBE s/dc/myCommon_10 - Receive commands using the myCommon_10 template.
- SUBSCRIBE s/e - Receive error messages.

Topic format

`<protocol>/<direction><type>[/<template>][/<child_id>]`

where:

- `<protocol>` can be s (persistent), t (transient), q (quiescent) and c (CEP), see [Processing mode](#) for more information.
- `<direction>` can be u (upstream from the device), d (downstream to the device) or e (error).
- `<type>` can be s (static), c (custom, device-defined), d (default), t (template) or cr (credentials).

Device registration

- CONNECT 1234 management/devicebootstrap
- SUBSCRIBE s/dcr
- PUBLISH s/ucr
- PUBLISH s/ucr
- ...
- 70,tenant,username,password

Template registration

- PUBLISH s/ut/myCommon_10
- 10,999,POST,MEASUREMENT,,c8y_MyMeasurement;;c8y_MyMeasurement.M.value,NUMBER,...
10,msgId,api,method,response,type,time,custom1.path,custom1,type,custom1.value

Templates

See the [templates quick reference](#) for an overview of the available MQTT static templates.