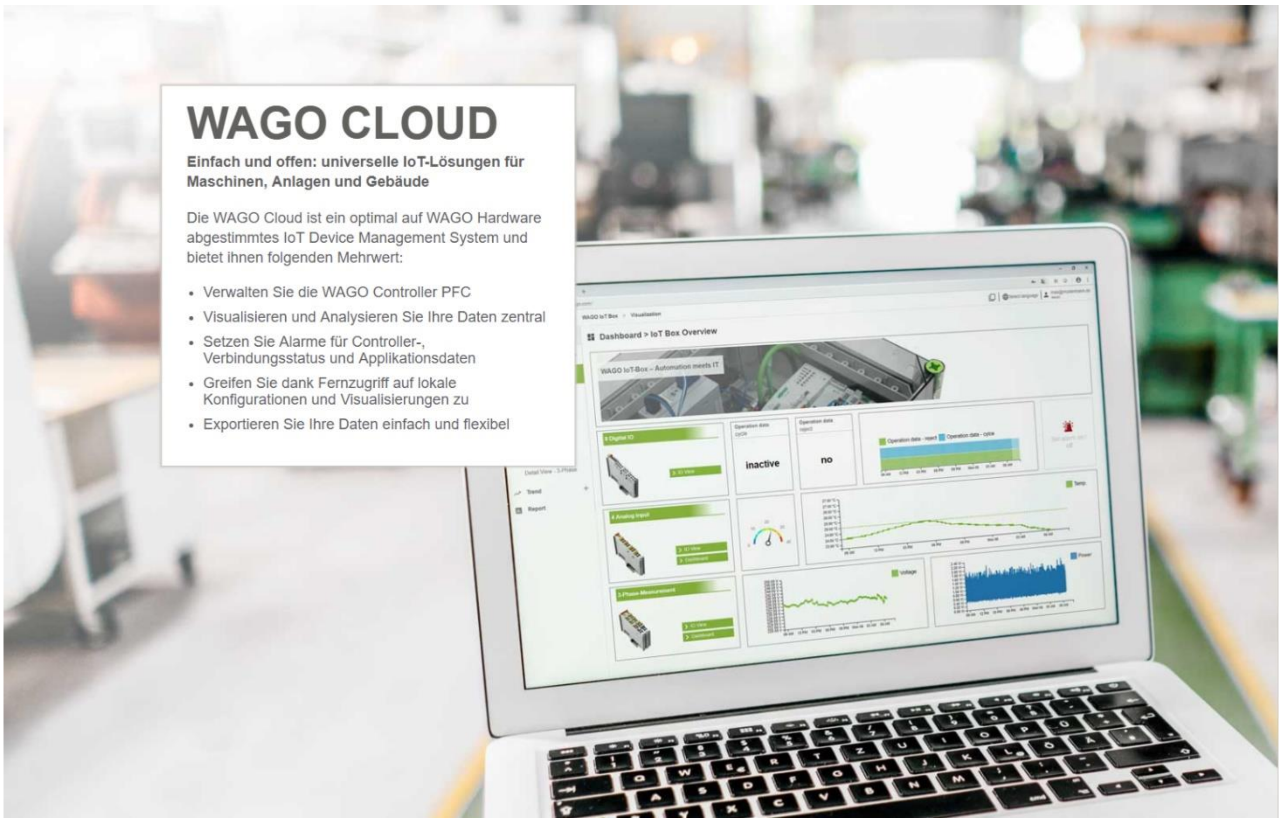
WAGO Cloud - REST-API Documentation


WAGO CLOUD
Einfach und offen: universelle IoT-Lösungen für Maschinen, Anlagen und Gebäude
Die WAGO Cloud ist ein optimal auf WAGO Hardware abgestimmtes IoT Device Management System und bietet ihnen folgenden Mehrwert:
Verwalten Sie die WAGO Controller PFC
Visualisieren und Analysieren Sie Ihre Daten zentral
Setzen Sie Alarme für Controller-, Verbindungsstatus und Applikationsdaten
Greifen Sie dank Fernzugriff auf lokale Konfigurationen und Visualisierungen zu
Exportieren Sie Ihre Daten einfach und flexibel
Dashboard > IoT Box Overview
inactive
no

Version: 1.7
2020-12-09
File name: WAGO Cloud - REST-API Documentation.docx

Table of contents

1 Overview

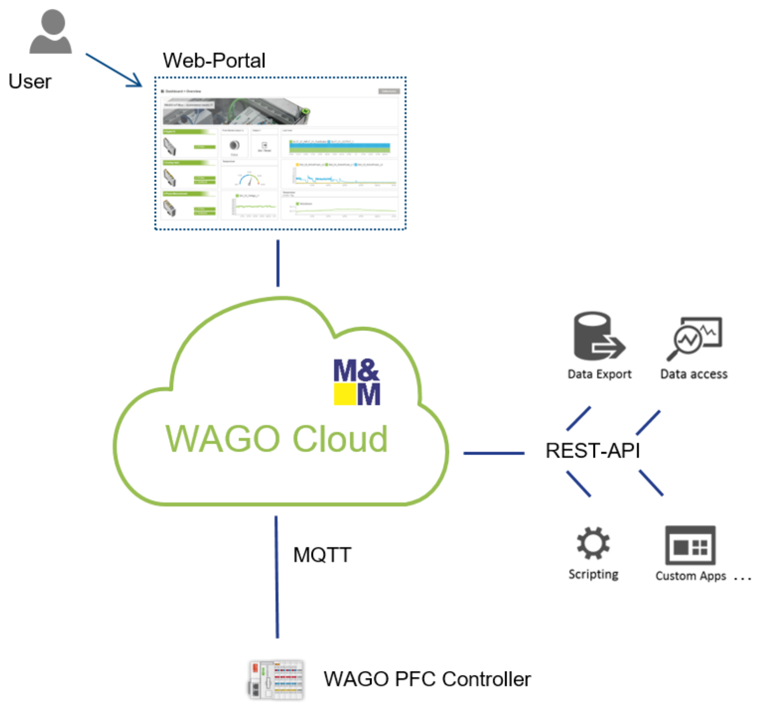
Besides managing the devices in the WAGO Cloud Web-Portal, it is also possible to manage the WAGO Cloud and the data via the REST-API. The REST-API provides the following use cases and functions:

Use cases of the REST-API:

- Scripting
  e.g. automatic device registration
- Custom applications
- Data access on historical data

REST-API Functions:

- Device management and access to device data
- Subscription and workspace management
- Alarm configuration and access to alarm data

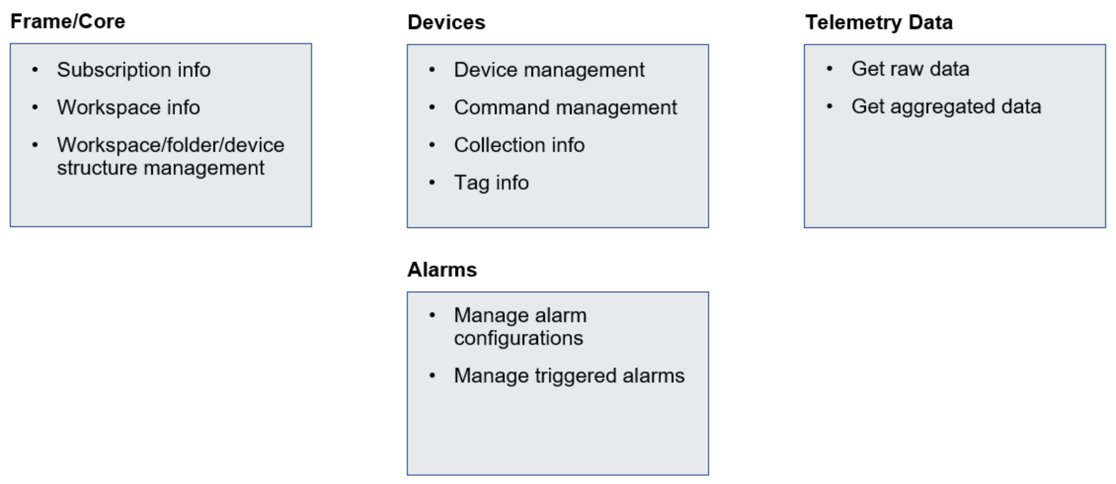


This document describes the architecture from the new version of the REST-API and gives some basic example how to register a device and receive data from the device. If you are working with an older version of the REST API v1 (deprecated) or v2 (deprecated), please use the REST-API documentation from 2018-10-01, Version 2.2.

2 Supported Components by REST-API

2.1 Overview

The following picture shows the components, which are supported by the REST-API. Each term used in the picture is described in the table followed by the picture.



2.2 Definition of Terms

Term	Definition
Frame/Core	The general functionality in the WAGO Cloud is clustered in the frame/core component, e.g. subscription and workspace management. The REST-API allows to get subscription and workspace information. The management of the workspace structure is also possible. It is also possible to create a subscription that informs about an event by sending callbacks to a specified client.
Device	Devices can be created in the WAGO Cloud. Devices, together with folders, are contained in the workspace structure. Data can be sent from the physical device. The sent data use collections and tags for the transmission of device data. Commands can be executed on the device. The REST-API allows to get device info and create or delete devices. Commands can be read and triggered on the device. Collections and tags can be also created, updated and deleted. Configuration of device is also possible. Developer can use API to configure device instead of sending a tag configuration message.
Telemetry data	The physical device can send telemetry data to the cloud. The telemetry data can be visualized in the WAGO Cloud, e.g. in a dashboard or in a trend.

Term	Definition
	The REST-API allows to export the raw data from the device and the aggregated data from the device. You can use the data in any other 3rd party tool.
Alarms	The WAGO Cloud contains an alarm management app. Alarm configuration can be created with several alarm rules. While sending telemetry data from devices, the alarm configurations will be checked and if a rule is met an alarm in the WAGO Cloud will be created. The REST-API supports the management of the alarm configurations and the management of triggered alarms.

The access to the WAGO Cloud components is always restricted by the WAGO Cloud user management. An API key (primary or secondary) is necessary Access to WAGO Cloud is always via a user.

2.3 Supported Operations by REST-API

App / Category	Method / Operation	Description
Subscriptions	GET/api/core/subscriptions	Gets all subscription information the user does have access to.
	GET/api/core/subscriptions/{id}	Gets the subscription information for the specified id.
	GET/api/core/subscriptions/{id}/workspaces	Gets a list of all workspaces for the subscription with the given id.
	PUT/api/core/subscriptions/{id}/apps	Register or update the registration of an existing application.
	GET/api/core/subscriptions/{id}/apps	Gets the list of all registered apps that were registered for the subscription with the given id.
	DELETE/api/core/subscriptions/{id}/apps/{name}	Deregisters the app with the given name from the subscription with the given id.
Workspaces	GET/api/core/workspaces/{id}/structure	Gets the workspace structure for the workspace with the given id.
	POST/api/core/workspaces/{id}/structure	Creates a folder in the workspace structure.
	GET/api/core/workspaces/{id}/structure/search	Searches for objects (devices/folder) within the workspace with the given id.

App / Category	Method / Operation	Description
	GET/api/core/workspaces/{id}/structure/{nodeId}	Searches for the structure node with the given node id in the workspace with the given workspace id.
	GET/api/core/workspaces/{id}/structure/{nodeId}/additional-properties	Gets the additional properties for a specific node folder or device, collection or tag.
	PUT/api/core/workspaces/{id}/structure/{nodeId}/additional-properties	Creates or updates the additional properties for a specific node folder or device, collection or tag.
	GET/api/core/workspaces/{id}/strcture/{nodeId}/aggregated-tags	Gets the configured structural tag aggregates for a folder, device or workspace
	DELETE/api/core/workspaces/{folderId}/folder	Deletes the specified folder and its sub-folders.
	DELETE/api/core/workspaces/{id}/structure/{nodeId}/additional-properties/{propertyName}	Deletes the additional properties for a specific node folder or device, collection or tag with the given property name.
	GET/api/core/workspaces/{id}/feature-permissions	Gets the feature permissions the calling user does have within the workspace with the given identifier.
Events	POST/api/core/eventsubscriptions	Creates a subscription that informs about an event by sending callbacks to a specified client.
	DELETE/api/core/eventsubscriptions/{id}	Deletes a previously created event subscription.
	GET/api/core/eventsubscriptions	Lists all currently active event subscriptions.
Devices	GET/api/deviceapp/devices	Gets all devices for the given workspace the user does have access to.
	POST/api/deviceapp/devices	Creates a device in the workspace with the given id.
	GET/api/deviceapp/devices/{id}	Returns the device with the given id.
	PUT/api/deviceapp/devices/{id}	Updates properties of a device with the given id.
	DELETE/api/deviceapp/devices/{id}	Deletes the device with given id.

App / Category	Method / Operation	Description
	GET/api/deviceapp/devices/{id}/commands	Gets all commands supported by the device with the given id.
	POST/api/deviceapp/devices/{id}/commands/{command}	Sends the specified command request to the device with the given id.
	GET/api/deviceapp/devices/{id}/collections	Gets all collections for the device with the given id.
	PUT/api/deviceapp/devices/{id}/collections	Creates/Updates the collections for the device with the given id.
	GET/api/deviceapp/devices/{id}/collections/{key}/tags	Gets the tags for the specified collection for the device with the given id.
	PUT/api/deviceapp/devices/{id}/collections/{key}/tags	Creates/Updates the tags for the device collection.
Telemetry Data	POST/api/telemetry/telemetrydata/raw	Gets the raw data values for all given tag descriptions.
	POST/api/telemetry/telemetrydata/aggregates/datapoints	Gets the aggregated data values for the given tag descriptions. The result will be packed into the given number of even sized datapoints.
	POST/api/telemetry/telemetrydata/aggregates/timeinterval	Gets the aggregated data values for the given tag descriptions. The result will be aggregated into time buckets of the given time interval.
	POST/api/telemetry/telemetrydata/aggregates/timewindow	Get aggregates with windowing.
	POST/api/telemetry/telemetrydata/latest	Gets for given tags the latest telemetry data values.
	POST/api/telemetry/telemetrydata/at	Gets the data values for the given tags at the specified point in time.
	POST/api/telemetry/telemetrydata/structuralaggregates/latest	Gets structural aggregation based on the last values.
	POST/api/telemetry/telemetrydata/structuralaggregates/timewindow	Get structural aggregates with windowing
Alarm Configuration V1.0	GET/api/alarmapp/alarmconfigurations	Gets the existing alarm configurations for the given workspace or device respectively.

App / Category	Method / Operation	Description
	POST/api/alarmapp /alarmconfigurations /plcStatusBased	Create a plc status based alarm configuration.
	PUT/api/alarmapp /alarmconfigurations /plcStatusBased/{id}	Updates a plc status based alarm configuration.
	POST/api/alarmapp /alarmconfigurations /valueBased	Create a value based alarm configuration.
	PUT/api/alarmapp /alarmconfigurations /valueBased/{id}	Updates a value based alarm configuration.
	POST/api/alarmapp /alarmconfigurations /connectionStatusBased	Create a connection status based alarm configuration.
	PUT/api/alarmapp /alarmconfigurations /connectionStatusBased/{id}	Updates a connection status based alarm configuration.
	DELETE/api/alarmapp /alarmconfigurations/{id}	Deletes the alarm configuration with the given id.
Alarms V1.0	GET/api/alarmapp/alarms	Gets a list of all raised alarms for the given workspace or device respectively.
	DELETE/api/alarmapp/alarms	Deletes all the alarms for the given workspace or device respectively.
	DELETE/api/alarmapp/alarms /{id}	Deletes the alarm with the given id.
	PATCH/api/alarmapp/alarms/{id}	Change the status of the alarm with the given id.
Alarm Configuration V2.0	GET/api/alarmapp /alarmconfigurations	Gets the configured alarm configurations for either the workspace or the device respectively.
	POST/api/alarmapp /alarmconfigurations /valueBased	Creates a value based alarm configuration.
	PUT/api/alarmapp /alarmconfigurations /valueBased/{id}	Updates the alarm configuration identified with the given id with the given configuration data.

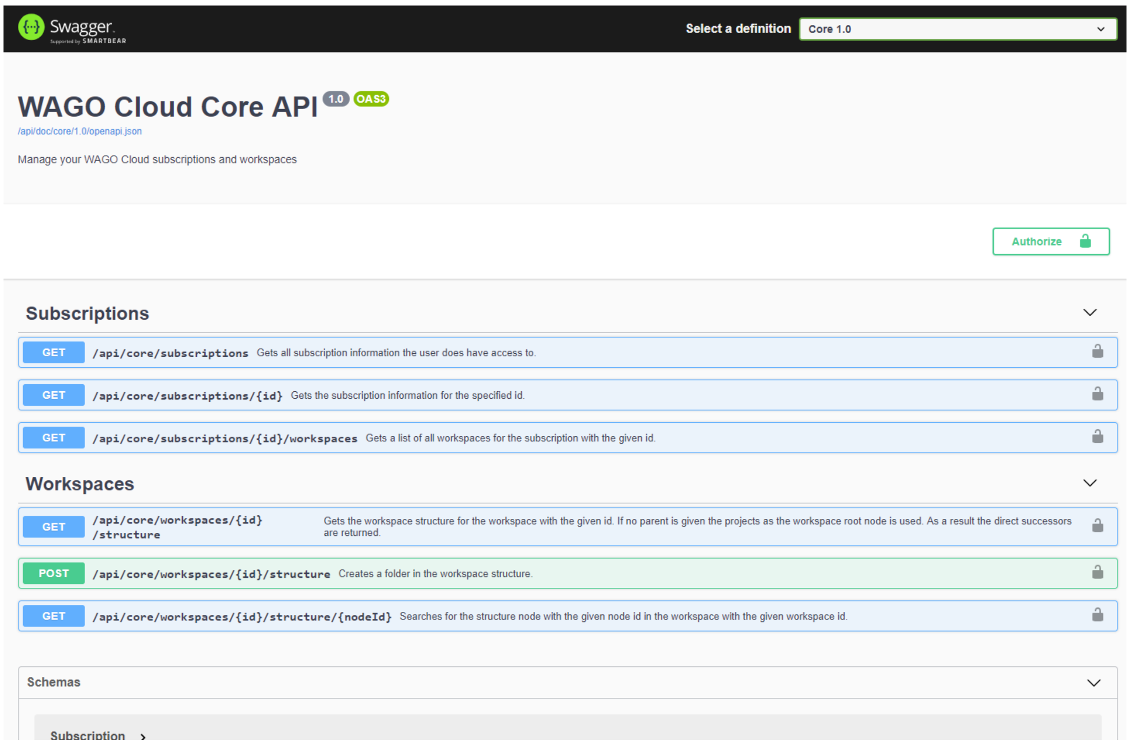
App / Category	Method / Operation	Description
	POST/api/alarmapp /alarmconfigurations /timeIntervalBased	Creates a time interval based alarm configuration.
	PUT/api/alarmapp /alarmconfigurations /timeIntervalBased/{id}	Updates the alarm configuration identified with the given id with the given configuration data.
	POST/api/alarmapp /alarmconfigurations /timeoutBased	Creates an alarm configuration of the type timeout based within the workspace with the given identifier.
	PUT/api/alarmapp /alarmconfigurations /timeoutBased/{id}	Updates the alarm configuration with the given identifier.
Alarm Status V2.0	GET/api/alarmapp/alarmstatus	Gets the list of alarm statuses created for the given subscription
	PUT/api/alarmapp/alarmstatus	Creates/Updates alarm statuses for the given subscription

3 REST-API Usage via Swagger

3.1 Overview

Swagger is a tool to manage REST-APIs. The WAGO Cloud REST-API operations can be used via the Swagger UI, the detailed documentation of the REST-API is also part of the Swagger UI. You can open the WAGO Cloud REST-API via the following link: https://cloud.wago.com/api/doc/index.html#/

The Swagger UI Timeout is set to 60 Minutes. After that a new login is required.



WAGO Cloud Core API

/api/doc/core/1.0/openapi.json

Manage your WAGO Cloud subscriptions and workspaces

Subscriptions

GET `/api/core/subscriptions` Gets all subscription information the user does have access to.

GET `/api/core/subscriptions/{id}` Gets the subscription information for the specified id.

GET `/api/core/subscriptions/{id}/workspaces` Gets a list of all workspaces for the subscription with the given id.

Workspaces

GET `/api/core/workspaces/{id}/structure` Gets the workspace structure for the workspace with the given id. If no parent is given the projects as the workspace root node is used. As a result the direct successors are returned.

POST `/api/core/workspaces/{id}/structure` Creates a folder in the workspace structure.

GET `/api/core/workspaces/{id}/structure/{nodeId}` Searches for the structure node with the given node id in the workspace with the given workspace id.

The new REST-API was introduced with WAGO Cloud Release 2.2. The following components are supported:

Release 2.2

- Core (Frame)
- Devices
- Telemetry
- Alarm

The REST-API operations before release 2.2 should no longer be used and they are marked as deprecated:

- V1 (deprecated) - WAGO Cloud API
- V2 (deprecated) - WAGO Cloud API

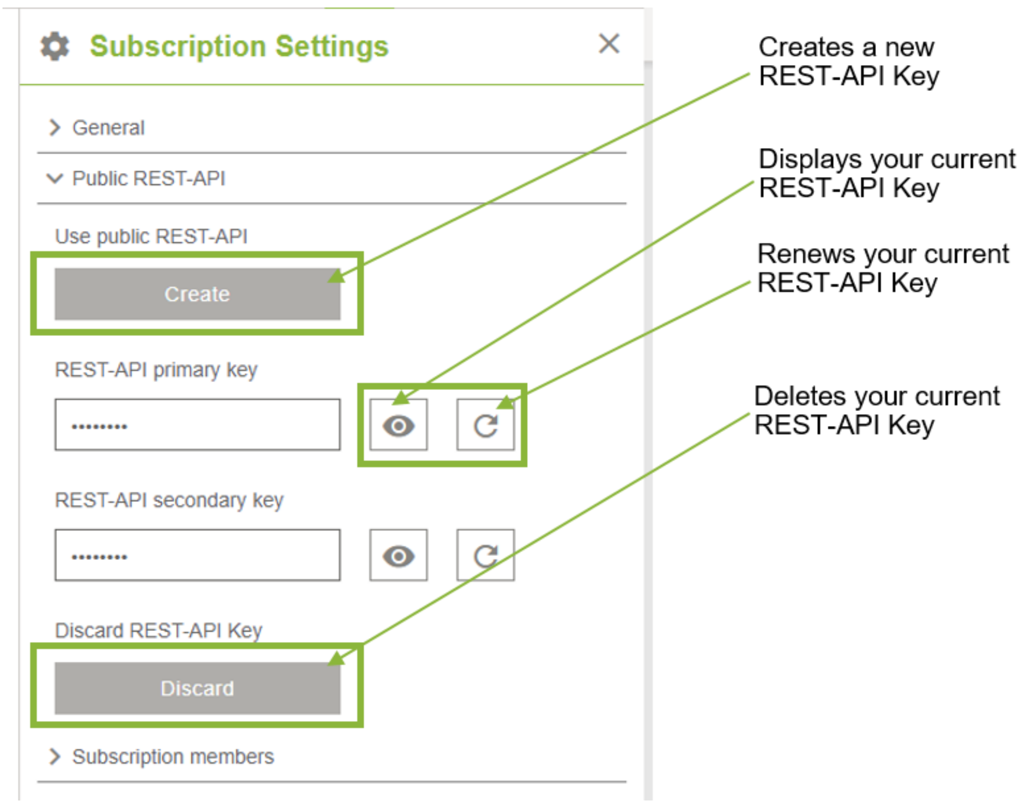
Below sections describe the usage of a REST-API key, the authorization and the usage of REST-API operations for the WAGO Cloud.

3.2 REST-API Key

A new REST-API key is required to be able to use the new version of the REST-API. This key uniquely assigns the REST-API to a subscription. The following steps explain shortly how the key is generated and used.

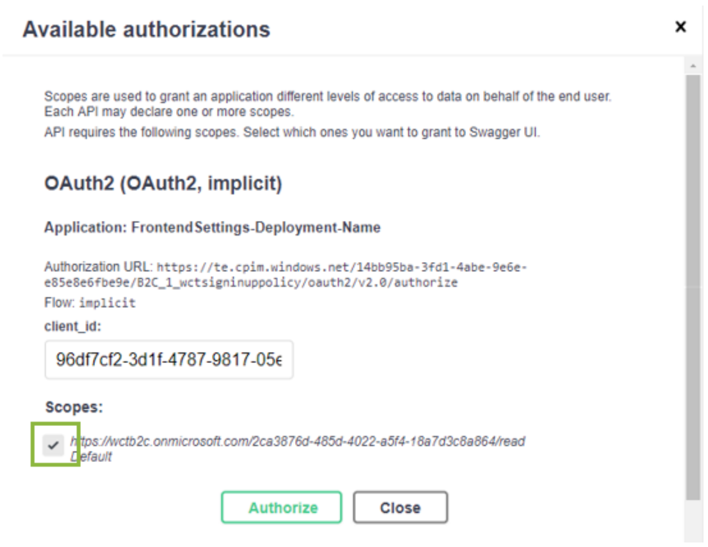
3.2.1 Create an API Key

The first thing you need to do is to login to the WAGO Cloud and open the subscription settings. Open the tab *"REST-API"* and click on "*Create*"*.* A new key is generated for this subscription. Make sure that you are Subscription admin, because only then you can create, delete or renew a key. As a normal user you can only see the created key. The secondary key can be used if the primary key is currently not available.



3.3 Authentication

Open the REST-API Swagger UI and click on "*Authorize". *Select the checkbox of the Scope and click again on "*Authorize*". You will be redirected to a Microsoft page where you will need to login with your WAGO Cloud credentials. **NOTE**: Do not change the *"client_id*" in below screenshot.



Available authorizations

Scopes are used to grant an application different levels of access to data on behalf of the end user. Each API may declare one or more scopes.
API requires the following scopes. Select which ones you want to grant to Swagger UI.

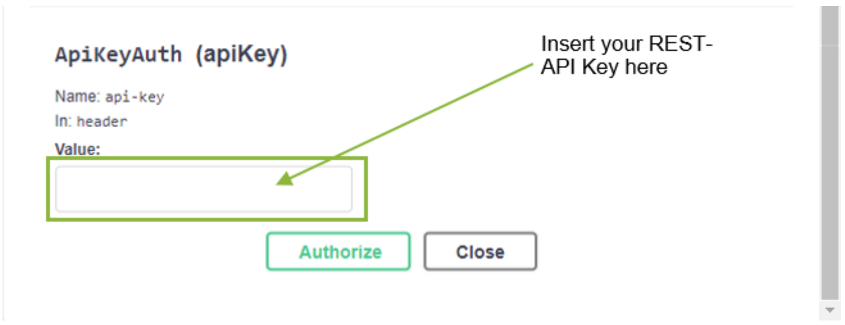
OAuth2 (OAuth2, implicit)

Application: FrontendSettings-Deployment-Name

Authorization URL: `https://te.cpim.windows.net/14bb95ba-3fd1-4abe-9e6e-e85e8e6fbe9e/B2C_1_wctsigninuppolicy/oauth2/v2.0/authorize`
Flow: `implicit`
client_id: 96df7cf2-3d1f-4787-9817-05e

Scopes:
✓ *https://wctb2c.onmicrosoft.com/2ca3876d-485d-4022-a5f4-18a7d3c8a864/read*
Default

After you have successfully signed up you have to copy the REST-API key from the WAGO Cloud into the appropriate field and click again on “*Authorize*”.



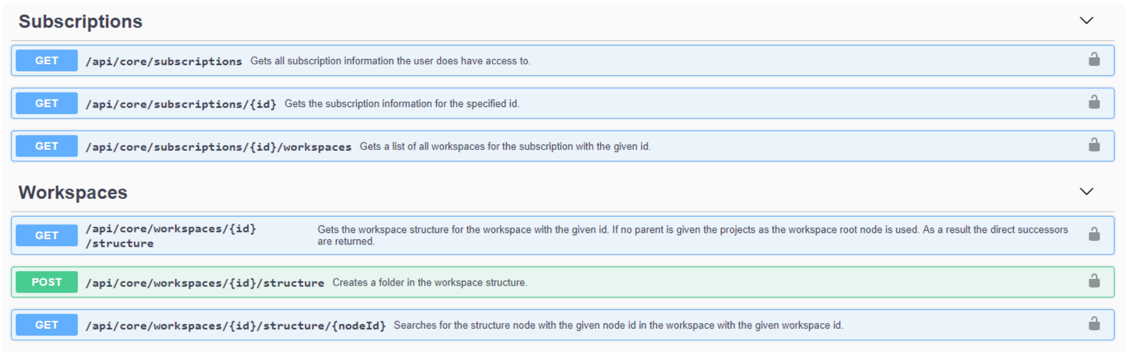
ApiKeyAuth (apiKey)

Name: `api-key`
In: `header`
Value:

You can now work with the WAGO Cloud REST-API. This is a paid feature. For detailed information see the WAGO Online Calculator.

3.4 REST-API Operations

To execute the WAGO Cloud REST-API operations, you have to choose the appropriate definition, e.g. Core, Device, Telemetry or Alarm and then select the operation.



Subscriptions

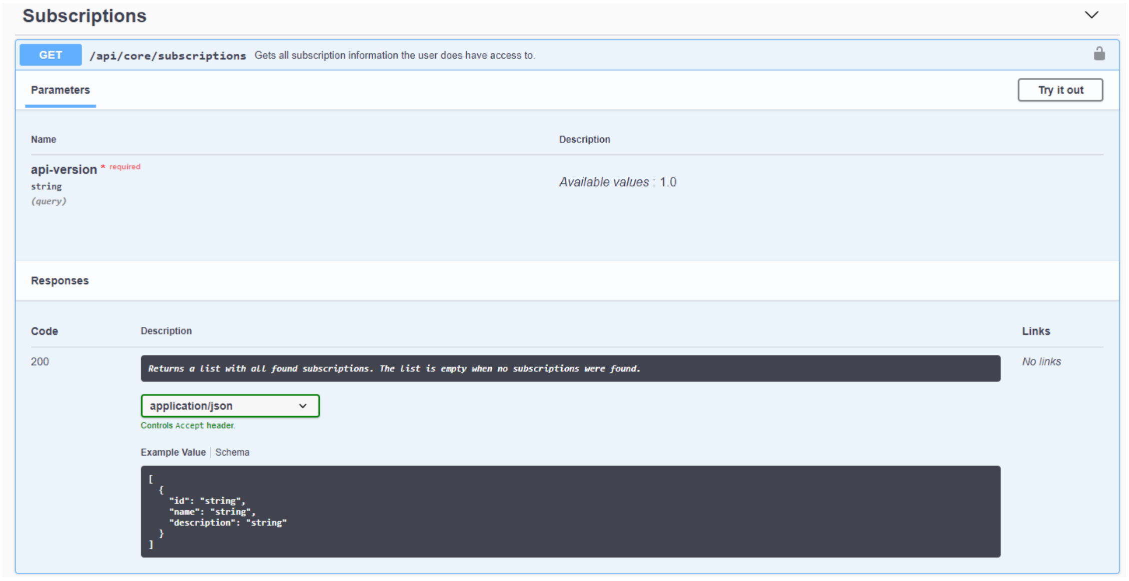
GET	`/api/core/subscriptions`	Gets all subscription information the user does have access to.
GET	`/api/core/subscriptions/{id}`	Gets the subscription information for the specified id.
GET	`/api/core/subscriptions/{id}/workspaces`	Gets a list of all workspaces for the subscription with the given id.

Workspaces

GET	`/api/core/workspaces/{id}/structure`	Gets the workspace structure for the workspace with the given id. If no parent is given the projects as the workspace root node is used. As a result the direct successors are returned.
POST	`/api/core/workspaces/{id}/structure`	Creates a folder in the workspace structure.
GET	`/api/core/workspaces/{id}/structure/{nodeId}`	Searches for the structure node with the given node id in the workspace with the given workspace id.

If you select the operation, the parameters and the responses will be displayed. The mandatory parameters are marked with "*required". For some operations, different response codes are possible.

After the successful authentication and the click on "Try it out", the request body is displayed and can be changed. The operation will be performed, if you click on the "Execute" button. The result will be displayed under "Responses".



/api/core/subscriptions Gets all subscription information the user does have access to.

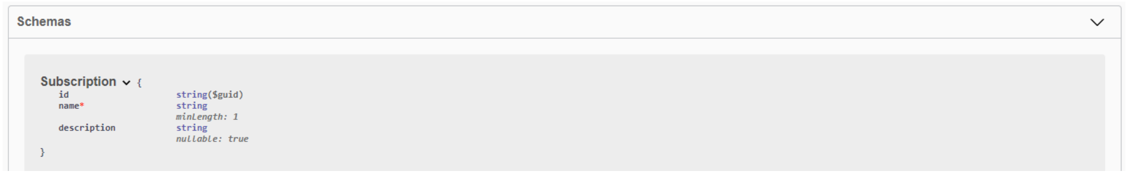
Name	Description
api-version * required string *(query)*	*Available values* : 1.0

Code	Description	Links
200	*Returns a list with all found subscriptions. The list is empty when no subscriptions were found.*	*No links*

Controls Accept header.

```
[
  {
    "id": "string",
    "name": "string",
    "description": "string"
  }
]
```

Below the operations, the schemas are displayed. The schemas describe the possible data types with their attributes, e.g. subscription or device. Enumerations can also be shown in the schema definition.



```
Subscription {
   id                string($guid)
   name*             string
                     minLength: 1
   description       string
                     nullable: true
}
```

4 REST-API Scenarios via HTTP requests

The following chapters contain step-by-step manuals of how to get started with the WAGO Cloud REST-API.

These chapters will show you:

- How to get subscription and workspace information
- How to create a device in your workspace
- How to get all commands supported by your device
- How to send a command to your device
- How to get a workspace structure
- How to get telemetry data

4.1 Get Subscription and Workspace Information

For executing all APIs, the precondition would be you have already get authorized, which mentioned in chapter 3.3.

To get the information for a subscription, you need to know the subscription Id, which is created after you registered your account. To get this information use the HTTP GET /api/core/subscriptions/{id} method.

The required parameter is subscription Id, which is a GUID string:

```
["<mySubscriptionId>"]
```

If the request is successful, you will get the HTTP status code 200. The response body would be like this:

```
{
  "id": "ecfb9777-d46d-4ccf-bffa-3ff5961d38bf",
  "name": "Sample subscription",
  "description": " Sample subscription"
}
```

4.2 Create a Device

To create a device, you need to invoke the HTTP POST /api/deviceapp/devices method. For the parameter, your need give the workspace Id:

```
["<workspaceId>"]
```

The request body should contains the folder Id and device name you defined:

```
{
   "folder": "<yourFolderId>",
   "name": "<yourDeviceName>"
}
```

If the request is successful, you will get the HTTP status code 201. The response body would be like this:

```
{
  "name": "Device_X",
  "key": "30etXnPPiKjjOlzrdoRpkqxXQew7Em6qZ+zDQBn3yL8="
}
```

4.3 Get all Commands Supported for a Device

To get all commands supported for a device use the HTTP Get /api/deviceapp/devices/{id}/commands method. For the parameter you need to provide the device id.

```
["<deviceId>"]
```

If the request is successful, you will get the HTTP status code 200. The response body would be like this:

```
[
 {
  "id": 502,
  "name": "StartDataSubmission",
  "commandRequest": [],
  "commandResponse": [
   {
    "name": "Started",
    "type": "string"
   }
  ]
 },
 {
  "id": 501,
  "name": "StopDataSubmission",
  "commandRequest": [],
  "commandResponse": [
   {
    "name": "Stopped",
    "type": "string"
   }
  ]
 },
 {
  "id": 256,
  "name": "ChangeSamplingRate",
  "commandRequest": [
   {
    "name": "SampleIntervalMilliseconds",
    "type": "dint"
   },
   {
    "name": "CollectionId",
    "type": "dint"
   }
  ],
  "commandResponse": [
   {
    "name": "Result",
    "type": "string"
   },
   {
    "name": "CollectionId",
    "type": "string"
   },
   {
    "name": "SampleIntervalMilliseconds",
    "type": "string"
   }
  ]
 },
 {
  "id": 257,
  "name": "ChangePublishingRate",
  "commandRequest": [
   {
    "name": "PublishIntervalMilliseconds",
    "type": "dint"
   },
   {
    "name": "CollectionId",
    "type": "dint"
   }
  ],
  "commandResponse": [
   {
    "name": "Result",
    "type": "string"
   },
      {
        "name": "CollectionId",
        "type": "string"
      },
      {
        "name": "PublishIntervalMilliseconds",
        "type": "string"
      }
    ]
  },
  {
    "id": 500,
    "name": "DeleteController",
    "commandRequest": [],
    "commandResponse": []
  },
  {
    "id": 504,
    "name": "GetNodeInfo",
    "commandRequest": [],
    "commandResponse": [
      {
        "name": "TransferTriggered",
        "type": "string"
      }
    ]
  },
  {
    "id": 503,
    "name": "SetHeartbeatInterval",
    "commandRequest": [
      {
        "name": "IntervalInSeconds",
        "type": "string"
      }
    ],
    "commandResponse": [
      {
        "name": "IntervalInSeconds",
        "type": "string"
      }
    ]
  },
  {
    "id": 509,
    "name": "StartRemoteAccessMediator",
    "commandRequest": [
      {
        "name": "URL",
        "type": "string"
      }
    ],
    "commandResponse": [
      {
        "name": "Started",
        "type": "bool"
      }
    ]
  },
  {
    "id": 510,
    "name": "StopRemoteAccessMediator",
    "commandRequest": [],
    "commandResponse": [
      {
        "name": "Stopped",
        "type": "bool"
      }
    ]
```

```
 }
]
```

4.4 Send a Command to the Device

To send a command to the device, you need to invoke the HTTP PUT /api/deviceapp/devices/{id}/commands/{command} method.

For the parameters, you need to provide the device id and the key of the command to send. The system supports 9 general default commands described in below table:

Command Name	Command Id
ChangeSamplingRate	256
ChangePublishingRate	257
DeleteController	500
StopDataSubmission	501
StartDataSubmission	502
SetHeartbeatInterval	503
GetNodeInfo	504
StartRemoteAccessMediator	509
StopRemoteAccessMediator	510

There are difference between commands for the command request and command response. If you are not familiar with the command request part you can go to 4.7 to get the overview of all basic commands.

For instance, if you want to stop the data submission for a running device, you need to provide:

device id (e.g. 84f7b993-d5dc-499f-9d5d-0fa58f4127e5)

command key: “501”

command request in the request body: “[]”

If the request is successful, you will get the HTTP status code 200. The response body would be like this:

```
{
  "parameters": [
    {
      "name": "Stopped",
      "value": "true"
    }
  ]
}
```

4.5 Get the Workspace Structure

To get the workspace structure use the HTTP Get /api/core/workspaces/{id}/structure method. The workspace id has to be passed as a parameter:

```
["<workspaceId>"]
```

You can add an optional parameter “parent Id” if you want to get the node structure where to start the search from:

```
["<parentId>"]
```

If the request is successful, you will get the HTTP status code 200. The response body would be like this:

```
{
  "rootNode": {
    "id": "4c70f62d-f8aa-4da4-a589-469d8fbfed63",
    "parentId": "00000000-0000-0000-0000-000000000000",
    "nodeId": "4c70f62d-f8aa-4da4-a589-469d8fbfed63",
    "name": "API TEST",
    "type": "Project"
  },
  "nodes": [
    {
      "id": "bbd2d91d-8248-4b27-a61c-a37a72d65562",
      "parentId": "4c70f62d-f8aa-4da4-a589-469d8fbfed63",
      "nodeId": "bbd2d91d-8248-4b27-a61c-a37a72d65562",
      "name": "F1",
      "type": "Folder"
    },
    {
      "id": "73d1f164-db90-4899-b25e-7f839ed644e1",
      "parentId": "4c70f62d-f8aa-4da4-a589-469d8fbfed63",
      "nodeId": "73d1f164-db90-4899-b25e-7f839ed644e1",
      "name": "D1 with no parent id",
      "type": "Folder"
    },
    {
      "id": "4a641f6d-fa5d-4682-be20-47edd21661e8",
      "parentId": "4c70f62d-f8aa-4da4-a589-469d8fbfed63",
      "nodeId": "4a641f6d-fa5d-4682-be20-47edd21661e8",
      "name": "f4",
      "type": "Folder"
    }
  ]
}
```

4.6 Get Telemetry Data

To get the device raw data values for a given tag description use the HTTP POST /api/telemetry/telemetrydata/raw. For the parameter you need to provide the start time and end time.

In the request body scheme is:

```
[
  {
    "deviceId": "string",
    "collectionKey": "string",
    "tagKey": "string"
  }
]
```

If the request is successful, you will get the HTTP status code 200. The response body would be like this:

```
{
  "tags": [
    {
      "deviceId": "fb309847-e6cf-431d-a174-17905ba57002",
      "collectionKey": "0",
      "tagKey": "CTDint"
    }
  ],
  "values": [
    [
      "2019-07-09T07:59:55.5867939Z",
      -59
    ],
    [
      "2019-07-09T07:59:56.1864314Z",
      -61
    ],
    [
      "2019-07-09T07:59:56.7867947Z",
      -62
    ],
    [
      "2019-07-09T07:59:57.3869341Z",
      -64
    ],
    [
      "2019-07-09T07:59:57.986446Z",
      -66
    ],
    [
      "2019-07-09T07:59:58.5865269Z",
      -68
    ],
    [
      "2019-07-09T07:59:59.186103Z",
      -69
    ],
    [
      "2019-07-09T07:59:59.7863017Z",
      -71
    ]
  ]
```

5 Public Event Channel

WAGO Cloud allows you to create your own event subscriptions and react to different WAGO Cloud events. The word “subscription” has two different meanings in the following. In order to prevent confusion, we use “event subscription” (subscribe to an event) and “WAGO Cloud subscription” (subscription for a user in the WAGO Cloud, which contains workspaces).

The public event channel feature provides three REST-API methods:

- Subscribing to an event (POST)
- List all event subscriptions (GET)
- Unsubscribe an event subscription (DELETE)

Public events can be triggered in the WAGO Cloud by different activities and components. After the registration to a specific event type, WAGO Cloud will create an public event, if the internal event occurs. The specific content of the received event message is stored in JSON format. The event is managed via the Azure Event Grid, using in detail WebHook as the communication method within the WAGO Cloud. This allows the user to be notified of events, e.g. for scheduled CSV exports, and to further use the data in the event message in a 3rd party system.

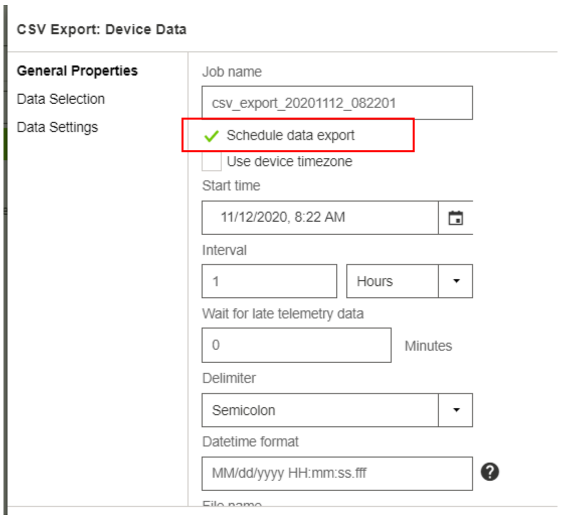
5.1 Supported Public Event Types

Currently, from WAGO Cloud release 2.10 on, scheduled csv exports and alarm events are supported. In future, device status and telemetry events will be supported.

Scheduled csv data exports can be created via the device app or via the trend element in the visualization app. Prerequisite for that is to enable the “Scheduled data export” for the WAGO Cloud subscription. One more prerequisite for triggering an public event is that the device sends data. A scheduled CSV export can be created using the "Schedule data export" checkbox in the device configuration or in the trend element within the visualization app.

The scheduled CSV export events can be subscribed on

- workspace level


CSV Export: Device Data

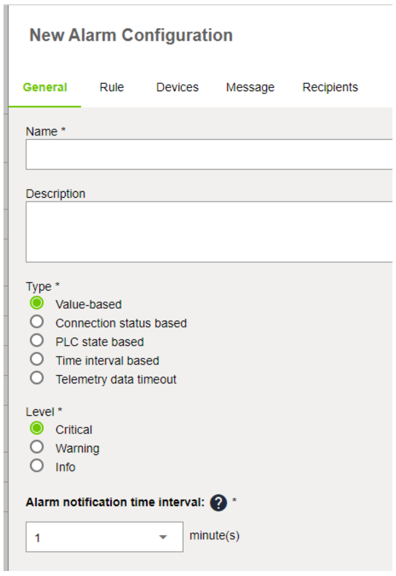
Job name: csv_export_20201112_082201
[✓ Schedule data export]
[ ] Use device timezone
Start time: 11/12/2020, 8:22 AM
Interval: 1 Hours
Wait for late telemetry data: 0 Minutes
Delimiter: Semicolon
Datetime format: MM/dd/yyyy HH:mm:ss.fff

Figure 1: Enable cyclic data export for CSV export events

Alarms can be received via creating an alarm configuration in the alarm app, if the alarm rule meets the current situation. If an alarm event is registered, all alarm types can be received, e.g. Value-based, connection status based, PLC state based, time interval based and telemetry timeout based.

The alarm events can be subscribed on

- device level
- workspace level
- subscription level


New Alarm Configuration

Name *

Description

Type *
[✓ Value-based]
Connection status based
PLC state based
Time interval based
Telemetry data timeout

Level *
[✓ Critical]
Warning
Info

Alarm notification time interval: * 1 minute(s)

Figure 2: Create an alarm configuration

5.2 Subscribe for a Public Event

The required information for the authentication can be obtained via several REST-API methods. For a general process description of how to use the REST-API calls via the Swagger interface, the previous chapters can be used.

To create an event, the WAGO Cloud *Subscription ID* and *Workspace ID* are required.

The *Subscription ID* of a WAGO Cloud Subscription can be retrieved using the `GET /api/core/subscriptions` method. In the following sample code, the *Subscription ID* is highlighted green.

```
{
    "id": "ebf5acb1-1639-443f-a8eb-c2eb3edbb339",
    "name": "BOC Subcription",
    "description": ""
  }
```

The *Workspace ID* of a WAGO Cloud subscription can be retrieved using the `GET /api/core/subscriptions/{id}/workspaces` method. In the following section, the *Workspace ID* is highlighted red.

```
{
      "id": "26333ed1-69c4-4d6d-9369-d3372480ab88",
      "name": "My workspace"
    }
```

This information can be used in the request body of the method `POST /api/core/eventsubscriptions`.

```
 {
        "SubscriptionId": " ebf5acb1-1639-443f-a8eb-c2eb3edbb339",
        "NodeId": "26333ed1-69c4-4d6d-9369-d3372480ab88",
        "EventType": "CsvExportEvents",
        "Configuration": {
            "endpointType": "WebHook",
            "subscriberUri": "[your callback client URI]"
        }
    }
```

The *SubscriptionId* is the ID of the WAGO Cloud subscription, whereas the *Node ID* is defined via the *Workspace ID* for the WAGO Cloud Subscription. The *Node ID* represents the ID that defines the scope of the event-causing devices. Depending on the type of event, this can denote different levels, such as a workspace, a single device or an entire WAGO Cloud subscription.

The *EventType* is the type of the specific event. In this example the event type *CsvExportEvents* is specified.

In the configuration of the event the *endpointType* can be set. At the moment only WebHook is used here, which offers the possibility to inform other applications about a certain event and to provide it promptly with information.

The *subscriberUri* defines the destination address the event is sent to. If an Azure Function has been created for events, the URI for the callback client is constructed

with the following parts: [base URL] + /api/ + [FunctionName]
The *base URL* can be found for example in the Azure Portal in an Azure Function. The *FunctionName* can be obtained in the source code of the Azure Function in the attribute *FunctionName*. Based on the information, the *subscriberURI* should look like this:

```
{
        "SubscriptionId": " ebf5acb1-1639-443f-a8eb-c2eb3edbb339",
        "NodeId": "26333ed1-69c4-4d6d-9369-d3372480ab88",
        "EventType": "CsvExportEvents",
        "Configuration": {
            "endpointType": "WebHook",
            "subscriberUri": "https://wagocloud-
eventchannelcb.azure.net/api/HttpTriggerCallback"
        }
    }
```

At the same time, the Azure Function in the portal allows you to see which calls were made via the Azure Function.

The usage of an Azure Function is only one of the many possibilities of an event receiver. Any web service that can receive an `HTTP POST` and respond to the validation code sent by the Event Grid with a suitable response can be used here.

5.3 Sequence of Events: Synchronous Handshake

There are many ways to receive events from the Azure Event Grid. Within the WAGO Cloud, WebHook is currently used as the only option, but it offers maximum flexibility regarding the event receiver, since the `HTTP` protocol is used as the lowest common denominator.

To understand the basic principle, the following figure can be used:

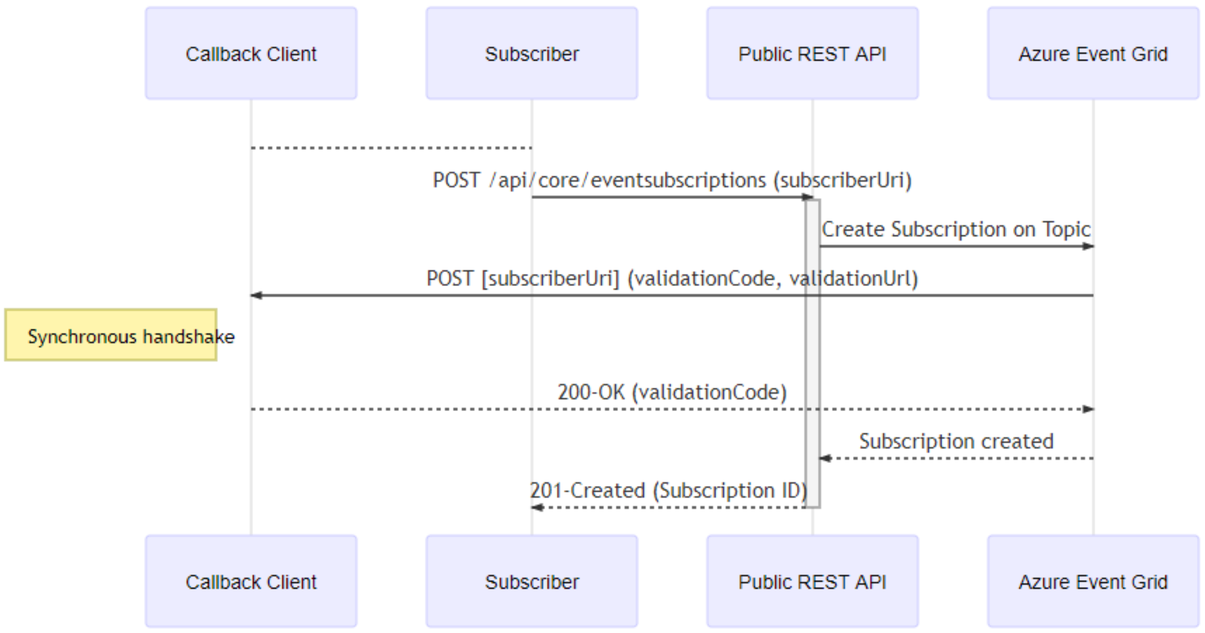


Figure 3: How the event subscription handshake works

By providing a web server, which also serves as a callback client, event subscriptions can be received. The server must be able to receive the actual events, consisting of `HTTP POST`s with JSON-body. In addition, the callback client must be running at the time the Public REST-API is called, as Azure is actively trying to reach it.

Event subscriptions are realized by the Azure Event Grid from Microsoft. As soon as a new event is ready, the Event Grid service sends a `POST` containing an `HTTP` request, which is sent to the configured endpoint including the event in the request text. The type of event is a `Microsoft.EventGrid.SubscriptionValidationEvent`:

```
[
    {
        "id": "45c11881-5b58-4d90-8601-1460fea12e64",
        "topic": "/subscriptions/700327a3-18e1-470b-9f06-
bc3bd2e61c99/resourceGroups/wagocloudtest/providers/microsoft.eventgrid/topics/wago-test-eun-
egri-csvexport",
        "subject": "",
        "data": {
            "validationCode": "01C525AF-AA91-4BB6-BF63-FF64426ACA75",
            "validationUrl": "https://rp-
northeurope.eventgrid.azure.net:553/eventsubscriptions/csvexportfinished-12ca6a2b-50d9-42b9-
9480-c23ecce2aede/validate?id=01C525AF-AA91-4BB6-BF63-FF64426ACA75&t=2020-10-
12T09:34:07.3031219Z&apiVersion=2020-06-01&[Hidden Credential]"
        },
        "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",
        "eventTime": "2020-10-12T09:34:07.3031219Z",
        "metadataVersion": "1",
        "dataVersion": "1.0"
    }
]
```

Using the `HTTP POST` mentioned above, Azure Event Grid sends a *validationCode* event to the callback client to identify an actual recipient behind the URL. For successful verification, the callback client must return the received *validationCode* to Azure Event Grid as part of its response. As status code the client must return `HTTP RESPONSE 200 - OK` and specify *validationCode* as the only JSON element *validationResponse* in the body:

```
{
    "validationResponse": "01C525AF-AA91-4BB6-BF63-FF64426ACA75"
}
```

In addition, the *validationUrl* is also sent with an URL for manual verification of the event. The event subscription has been created and receiving events is enabled.

The event message contains the *dataVersion*, which is the REST-API version used for the event subscription with the POST method. Moreover, it contains the *metadataVersion*, which is the version of the event schema in the Azure Event Grid. The content of the *data* tag is not part of the event schema.

Hint: If the client is unavailable, the retry strategy of the Event Grid supports repeated event delivery attempts. Those attempts are made based on performance after 10 seconds up to once per hour up to 24 hours. All events that are not delivered within 24 hours are marked accordingly by Event Grid.

In the case of handshake, there is only one attempt to reach the client. If this fails, no event subscription is created.

5.4 List all Public Event Subscriptions

Depending on the use case, several event subscriptions can be created. To see how many active events exist for a specific WAGO Cloud subscription, `GET /api/core/eventsubscriptions/{id}` can be used. This API method lists all running events using the *Subscription ID* (WAGO Cloud subscription ID) identified above.

```
[
    {
        "id": "ebf5acb1-1639-443f-a8eb-c2eb3edbb339",
        "eventType": "CsvExportEvents",
        "filterNodeType": "Workspace",
        "filterNodeId": "0120ea86-e839-443d-a7ea-66db41ef8899",
        "version": "1.0",
        "endpointConfiguration": {
            "endpointType": "WebHook",
            "subscriberUri": "https://wagocloud-
eventchannelcb.azure.net/api/HttpTriggerCallback",
            "resourceId": null
    }
]
```

Once again, you can see the details of what was entered when the event subscription was created. In the code example above, a CSV export event with corresponding Workspace ID (*filterNodeId*) is sent to the *subscriberUri* for event handling.

5.5 Unsubscribe from a Public Event

For deleting existing events, the ID of the event is required. Simply call the public REST-API `DELETE /api/core/eventsubscriptions/{id}` using the event subscription ID.

If the ID is unknown, the method `GET /api/core/eventsubscriptions` can be used to identify which events are explicitly CSV export events. Then the required ID can be viewed (see code example above). Afterwards the `DELETE` method can be executed and the event registration is deleted.

5.6 Receive Scheduled CSV Export Events

Event subscriptions can be created using the code example above and the `POST /api/core/eventsubscriptions` API method with the event type "CsvExportEvents". The CSV export events can be subscribed on workspace level (nodeID = workspaceID).

A prerequisite for triggering an event is that the device sends data, otherwise no event can be triggered.

The JSON body for the event message after a triggered event looks like this:

```
[
    {
        "id": "81209303-37a0-4487-b832-98dc40f4aa26",
        "subject": "CsvDataExport",
        "data": {
            "CsvLink": "https://wctstorage.blob.core.windows.net/dataexportjob/bedb0667-630e-
4911-94e2-b6877bdfed92/cbec9a3a-6303-4426-82c2-6ff24abe79b3/20201012/913efd47-995c-40cd-8566-
fd8883b13dc8/f1ccf3ec-8a7a-41d9-ad0b-4d6adcfd0d81/dx_My%20workspace_20201012083900000-
20201012093234000.csv?sv=2018-03-
28&sr=b&sig=v87X82XVAukLelnda3fdFhfuyaBFOjka%2BZjynkS3Oco%3D&se=2020-10-
19T09%3A39%3A52Z&sp=rwdl",
            "DeviceId": "913efd47-995c-40cd-8566-fd8883b13dc8",
            "DeviceName": "My sample device",
            "IntervalEndTime": "2020-10-12T09:39:00Z",
            "IntervalStartTime": "2020-10-12T08:39:00Z",
            "JobId": "f1ccf3ec-8a7a-41d9-ad0b-4d6adcfd0d81",
            "SubscriptionId": "bedb0667-630e-4911-94e2-b6877bdfed92",
            "WorkspaceId": "cbec9a3a-6303-4426-82c2-6ff24abe79b3"
        },
        "eventType": "CsvExportFinished",
        "dataVersion": "1.0",
        "metadataVersion": "1",
        "eventTime": "2020-10-12T09:39:53.2313174Z",
        "topic": "/subscriptions/700327a3-18e1-470b-9f06-
bc3bd2e61c99/resourceGroups/wagocloudtest/providers/Microsoft.EventGrid/topics/WAGO-TEST-EUN-
EGRI-CsvExport"
    }
]
```

5.7 Receive Alarm Events

Event subscriptions can be created using the code example above and the `POST /api/core/eventsubscriptions` API method with the event type “AlarmEvents”. The alarm events can be subscribed on device level (nodeID = deviceID), on workspace level (nodeID = workspaceID) or on subscription level (no nodeID in the request body of the POST method).

The JSON body for the event message, e.g. connection state based event, after a triggered event looks like this:

```
[
    {
        "id": "81209303-37a0-4487-b832-98dc40f4aa26",
        "subject": "Alarms",
        "data": {
          "DeviceId": "758b1b79-97a5-4508-a76a-6d757a4f243c",
          "AlarmMessage": "<p>Connection status based - disconnected (message)</p><p>Connection
state: Disconnected</p><p>Device: SimulationDevice</p>",
          "SubscriptionId": "0dc39e2a-30f1-4c71-92b5-d152aecc456c",
          "WorkspaceId": "df098db9-6c9e-4fef-80f7-5268c3fd2c16",
          "AlarmType": "ConnectionStatusBased",
          "AlarmLevel": "Critical",
          "TagValues": []
        },
        "eventType": "Alarm",
        "dataVersion": "1.0",
        "metadataVersion": "1",
        "eventTime": "2020-12-07T10:50:44.2781234Z",
        "topic": "/subscriptions/700327a3-18e1-470b-9f06-
bc3bd2e61c99/resourceGroups/wagocloudtest/providers/Microsoft.EventGrid/topics/WAGO-TEST-EUN-
EGRI-Alarms"
    }
]
```