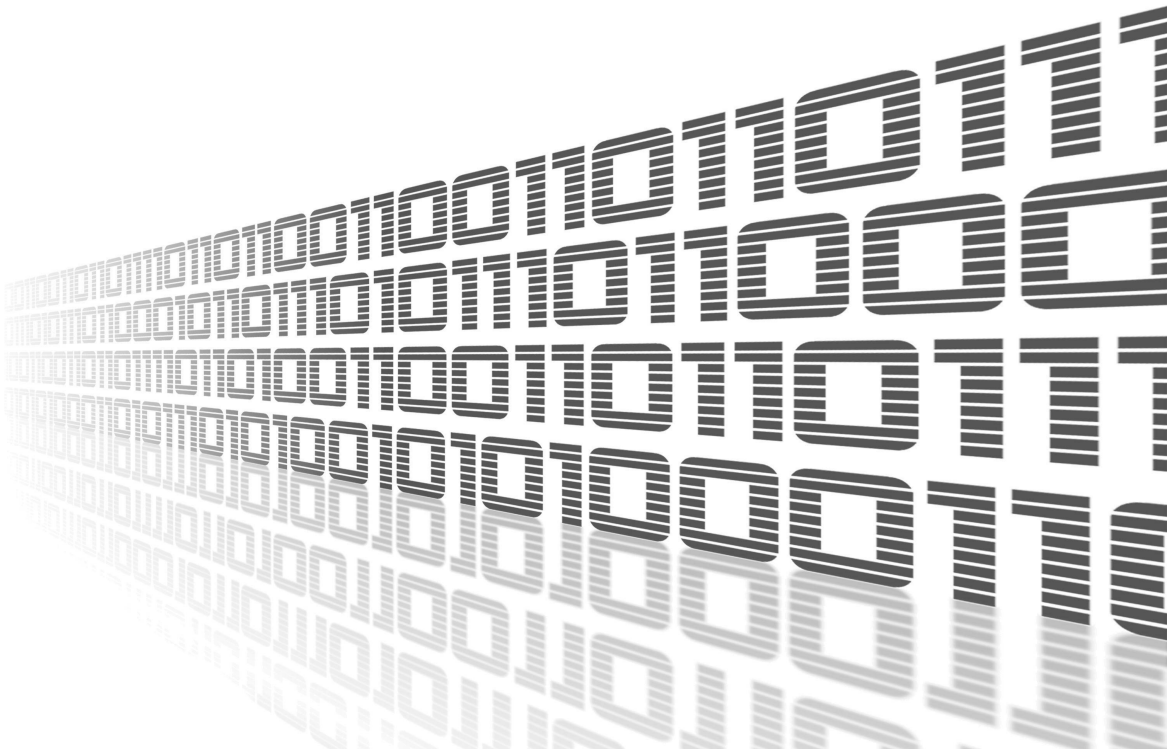
RouterApp

User Module

# Cumulocity Agent

APPLICATION NOTE



ADVANTECH



19-01-21

# Used symbols

*Danger* – Information regarding user safety or potential damage to the router.



*Attention* – Problems that can arise in specific situations.

*Information, notice* – Useful tips or information of special interest.

*Example* – example of function, command or script.

Advantech Czech s.r.o., Sokolska 71, 562 04 Usti nad Orlici, Czech Republic.
Document No. APP-0076-EN, revision from November 23, 2020. Released in the Czech Republic.

# Contents







19-01-21

# List of Figures







19-01-21

# 1. User Module Cumulocity Agent

## 1.1 Description

This user module is only compatible with *Advantech* routers of v2, v3 and v4 platforms.

This user module is not installed on *Advantech* routers by default. See the *Configuration Manual*, chapter *Customization -> User Modules*, for the description of how to upload a user module to the router.

Due to its space requirements there is no support for Geolocation and LUA scripts with platform V2.

For more information about Cumulocity service, see the Cumulocity online guides at https://www.softwareag.cloud/site/dev-center/cumulocity-iot.html#/.

*Cumulocity* service is a cloud-based subscription service designed for creating Internet of Things (IoT) solutions. It gives you very fast visibility and control over your remote devices. Cumulocity works with any network architecture, but it is specifically designed to work out of the box with mobile networks, therefore it is an ideal solution for Advantech routers.

Cumulocity stores all information about your devices (routers) on one place and displays it in a visual form through the web interface. The *Cumulocity Agent* user module allows you to monitor and store information about Advantech routers.

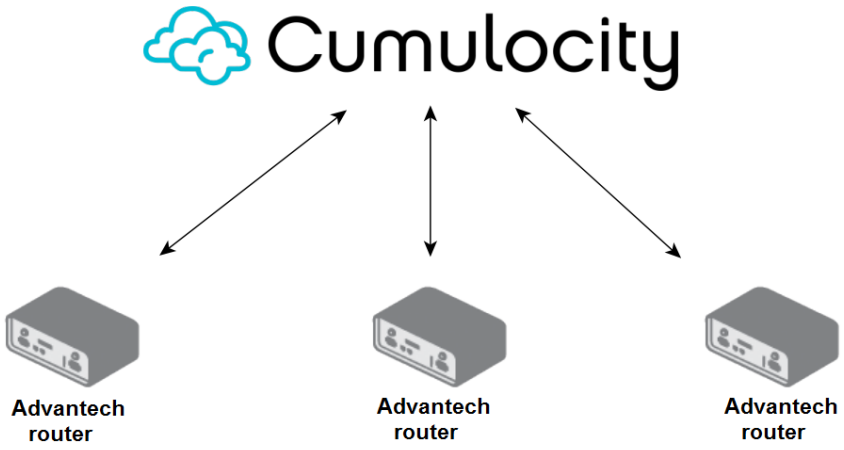



Figure 1: Routers Connected to Cumulocity Cloud Service via Cumulocity Agent

*Cumulocity Agent* user module installed in the router communicates with the Cumulocity cloud server secured by HTTPs or MQTTS. Immediately after connecting the router to the Cumulocity cloud, basic information about the router is available. At the same time Cumulocity starts to create statistics about the signal strength and other parameters.





19-01-21

*Cumulocity* user module makes these router information accessible through Cumulocity:

- Basic information about the router (name, type, model, serial number, IMEI etc.).
- Graphs of measurements and statistics – signal strength, router's memory usage and data traffic.

## 1.2 Installation

The latest version of *Cumulocity Agent* user module can be downloaded from the Engineering Portal [EP] at `https://ep.advantech-bb.cz/products/software/user-modules#cumulocity`.

In the GUI of the router navigate to *Customization -> User Modules* page. Here choose the downloaded module's installation file and click to the *Add or Update* button.

Once the installation of the module is complete, the module's GUI can be invoked by clicking the module name on the *User modules* page. In Figure 2 is shown the main menu of the module. It contains the *Status* menu section, followed by the *Configuration* and *General* menu sections. To return back to the router's web GUI, click on *Return* item.

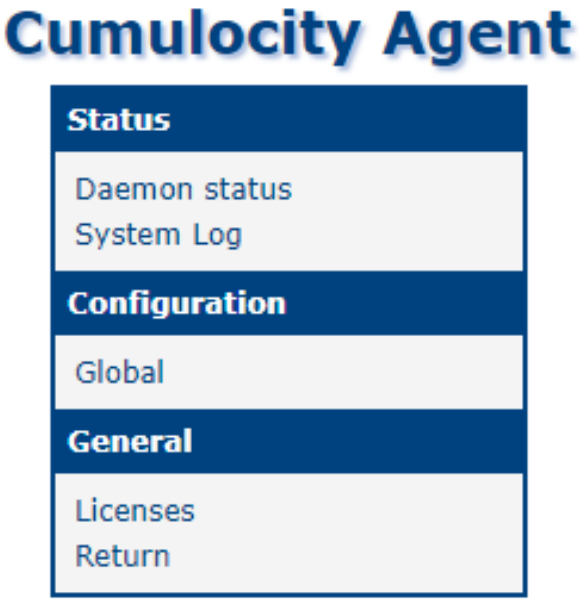



Figure 2: Main Menu





19-01-21

## 1.3 Configuration

Configuration of the *Cumulocity* user module is accessible through the web interface of the router in section *User Modules*. Click on the *Cumulocity Agent -> Configuration -> Global* and you will see the configuration page as shown in Figure 3.

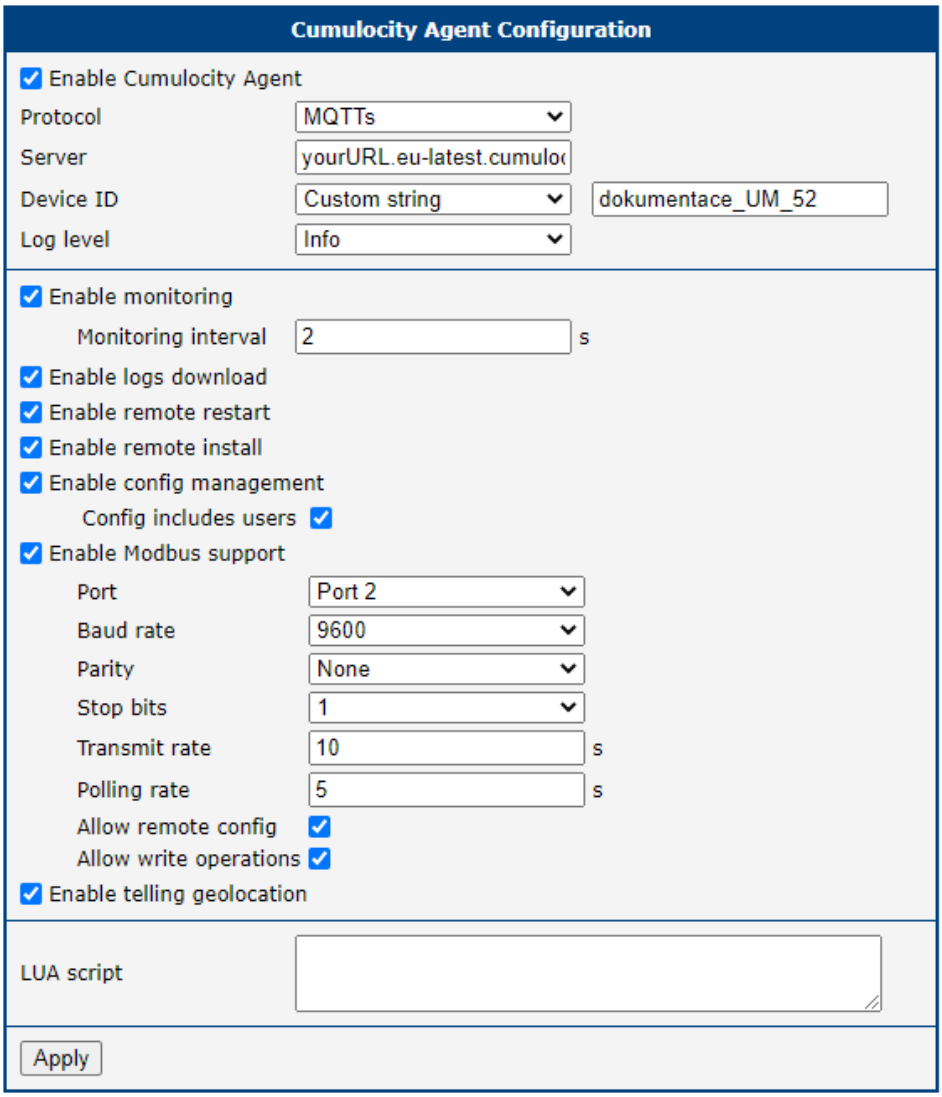



Figure 3: Configuration of *Cumulocity Agent* User Module

To get the server address, log in to the Cululocity cloud (see Chapter 2) and copy the part of URL address as shown on Figure 4.

Adding the router to Cumulocity cloud can be done from the Cumulocity web interface and is described in Chapter 2. To add the router, you just need to know the *Device ID* of the router.





19-01-21


Figure 4: Get the Server Address

In the table below are described all intems from user's module configuration page.

| Item | Description |
|---|---|
| Enable Cumulocity Agent | Enables the Cumulocity Agent user module. |
| Protocol | Choose the communication protocol, **MQTTs** or **HTTPs**. Both protocols are forced to use secure connection only, HTTPs uses **port 443**, MQTTs uses **port 8883**. These ports are enabled on the router automatically. Ensure they are enabled for the rest of the infrastructure as well. |
| Server | Adress of the Cumulocity cloud server without the "http" prefix, for example *yourURL.eu-latest.cumulocity.com*. Note that this address may differ for another region. |
| Device ID | ID of the device which is used as a unique identifier. There are these options:<br>• **Serial number** – serial number of the device.<br>• **Cellular IMEI** – IMEI of the 1st cellular module.<br>• **LAN MAC** – LAN MAC address of the 1st Ethernet (eth0).<br>• **Own string** – custom ID, ensure the uniqueness! |
| Log level | Choose a level for the logging. Available levels are: Critical, Error, Warning, Notice, Info and Debug. |
| Enable monitoring | Enables monitoring of cellular signal strength, power supply voltage, storage, temperature, memory and CPU. |
| Monitoring interval | Interval for the monitoring. |
| Enable logs download | Enables the server to download logs (dmesg, main log messages and modules logs) from the device. |
| Enable remote restart | Enables the server to initiate the device restart remotely. |
| Enable remote install | Enables the Cumulocity server to install modules or firmware to the device. Installation resources can be managed in the Management section. Direct link to a package on the Engineering Portal [EP] can be declared here. |

Continued on the next page

Continued from previous page

| Item | Description |
|---|---|
| Enable config management | Enables the server to get the configuration of the device. |
| Config include users | The configuration gained from the device remotely will include information about local users, including their credentials. |
| Enable Modbus support | Enables the communication with Modbus devices |
| Port, Baud rate, Parity, Stop bits, Transmit rate, Polling rate | Modbus communication settings |
| Allow remote config | Settings can be changed in cloud |
| Allow write operations | This option needs to be checked when we want to write from cloud to Modbus devices via router (otherwise the data will be rejected by router) |
| Enable tell geolocation[1] | Enables the server to gain the device location remotely. It is functional for the routers having a GNSS module only. |
| LUA script[1] | This is an optional script that will be executed during the agent start. The length of this script is limited by 12000 characters. For more information see chapter 1.5. |

Figure 5: Items of the *Cumulocity Agent* configuration

Most of the options will be described in detail later in this document.



[1] Not supported on routers of v2 production platform.

### 1.3.1 Monitoring

Enables monitoring of cellular signal strength, power supply voltage, free space on system and user part of storage, device temperature, total and used memory, CPU usage in selected device under *Measurements* menu item.

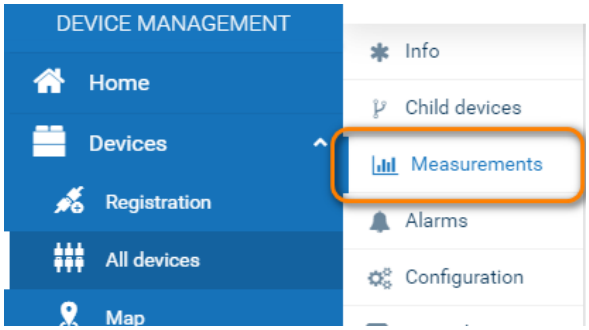



Figure 6: Measurements

**Monitoring interval** is value in seconds which determines how often the monitored values are sent to cloud.

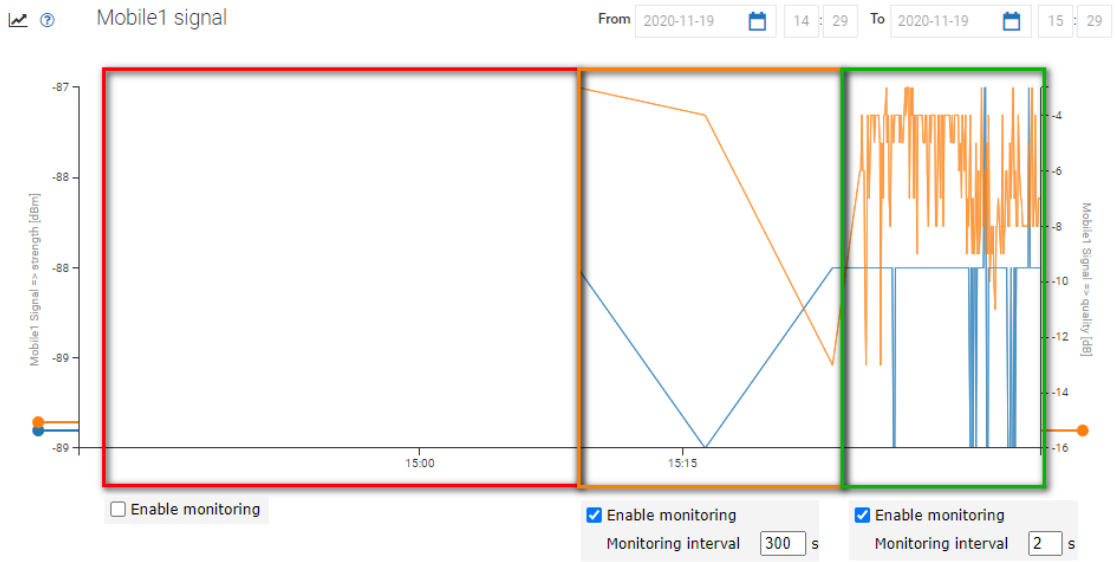



Figure 7: Device monitoring

### 1.3.2 Logs download

Enables the server to download logs (dmesg, main log messages and modules logs) from the device. When this checkbox is checked, menu item *Logs* appears. Then you are able to

request log file with your own criteria and parameters. When log file is ready, you can simply download log file. When rotated log is available it will join with actual log while downloading so the customer gets one continuous log file.

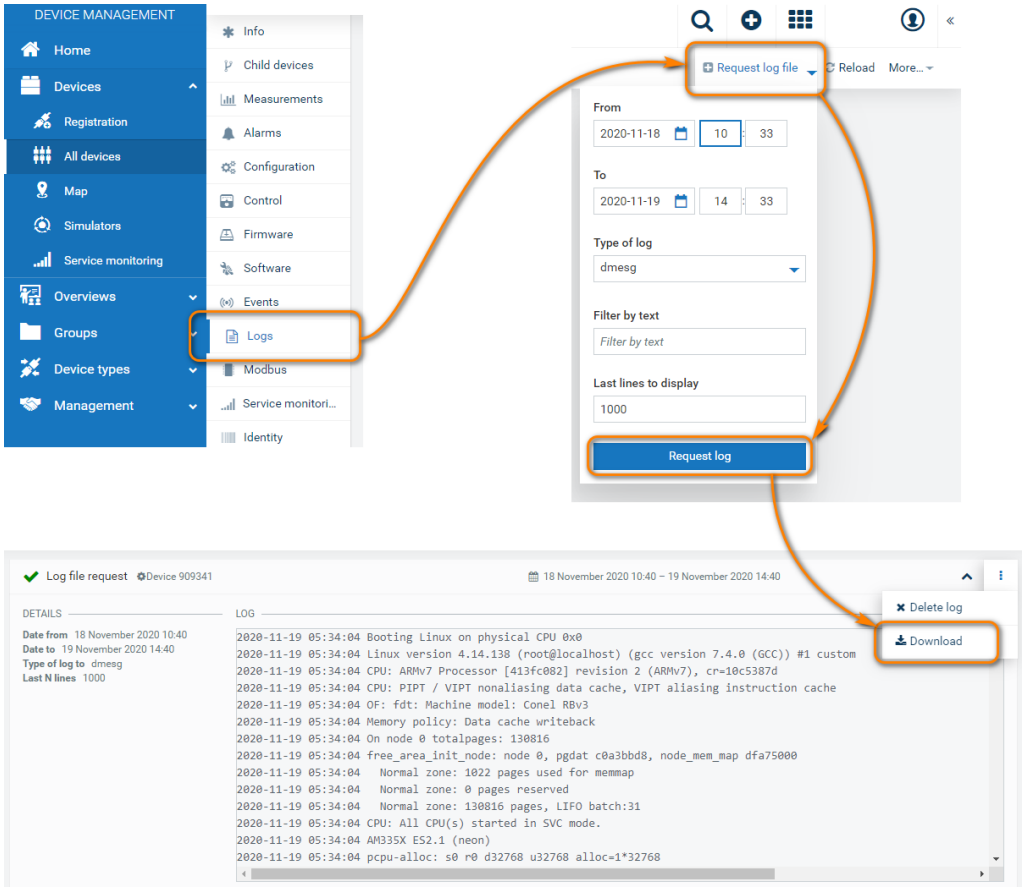



Figure 8: Log download

When log is requested, record with result of this action appears under the Control menu item.

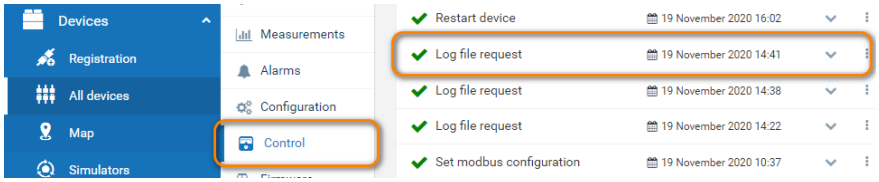



Figure 9: Log record

### 1.3.3 Remote restart

Enables the server to initiate the device restart remotely. When checkbox is checked you can find the *Restart device* item under *More...* dropdown menu.

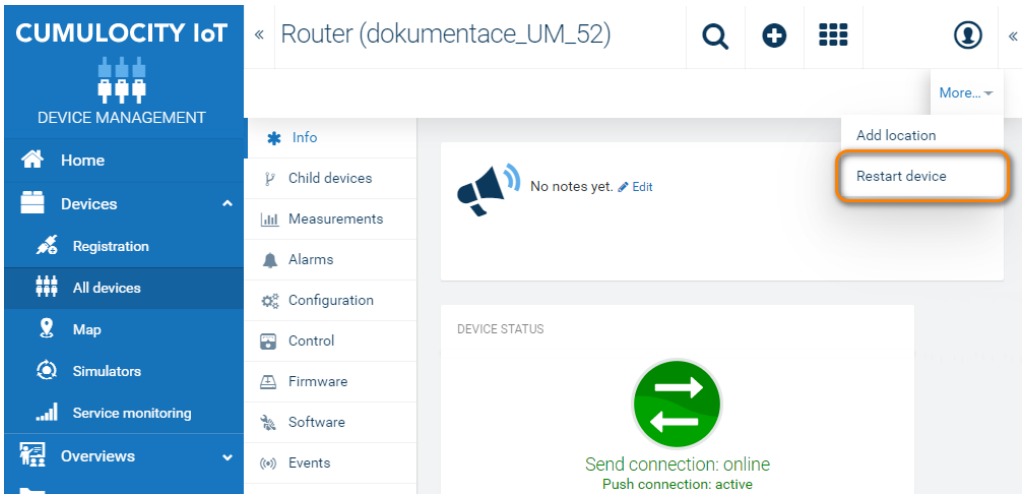



Figure 10: Remote restart

When restart is requested, record of this action appears under the Control menu item.

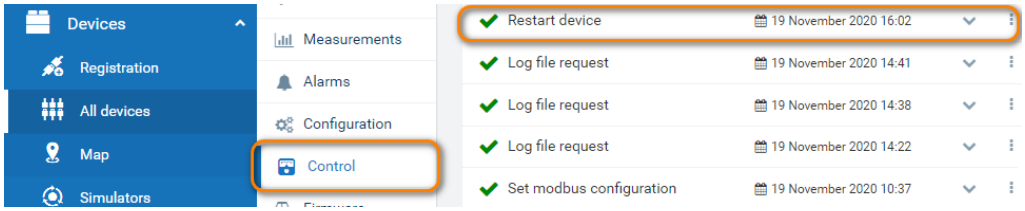



Figure 11: Restart device record

### 1.3.4 Remote install

Enables the Cumulocity server to install user modules (software) or firmware to the device. Installation resources can be managed in the Management section. You can select adding software to repository by upload binary file or by providing URL to software archive. Direct link to a package on the Engineering Portal [EP] can be declared here.

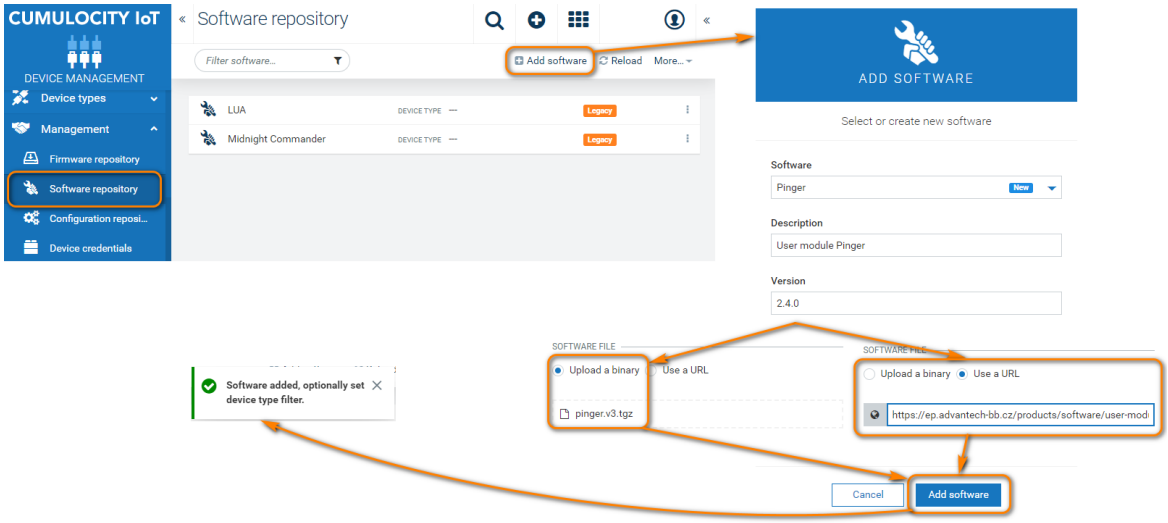



Figure 12: Add software

After adding desired software (or firmware) it is ready for install. Software installation is handled in the Devices Management / Devices / All devices / *your_router* / Software.

In case of adding firmware, there is possibility to upload whole package or even single *.bin file.

When adding software via URL with secure https protocol, you need to upload certificates as described in the FAQ in chapter 3. In case of installing directly from Engineering Portal the certificate is part of the package.

When software is installed or uninstalled, record with result of this action appears under the Control menu item.

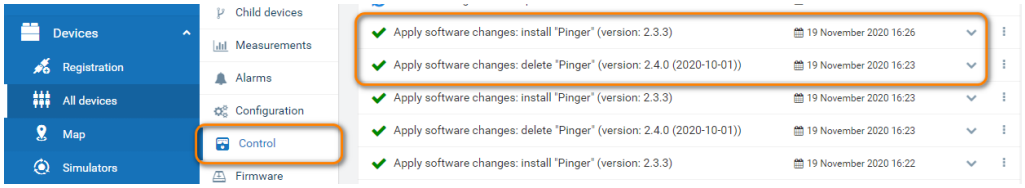



Figure 13: Record of sucessful software install

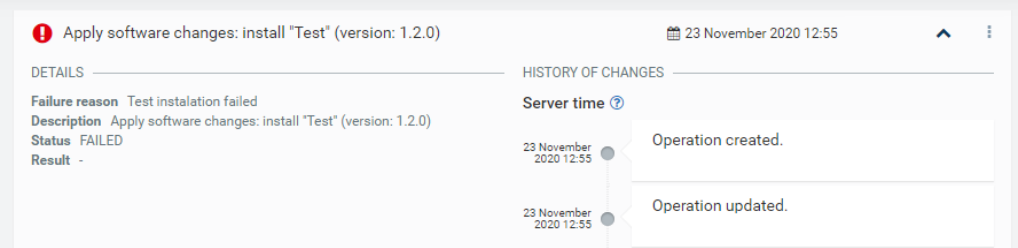



Figure 14: Record with failed software install

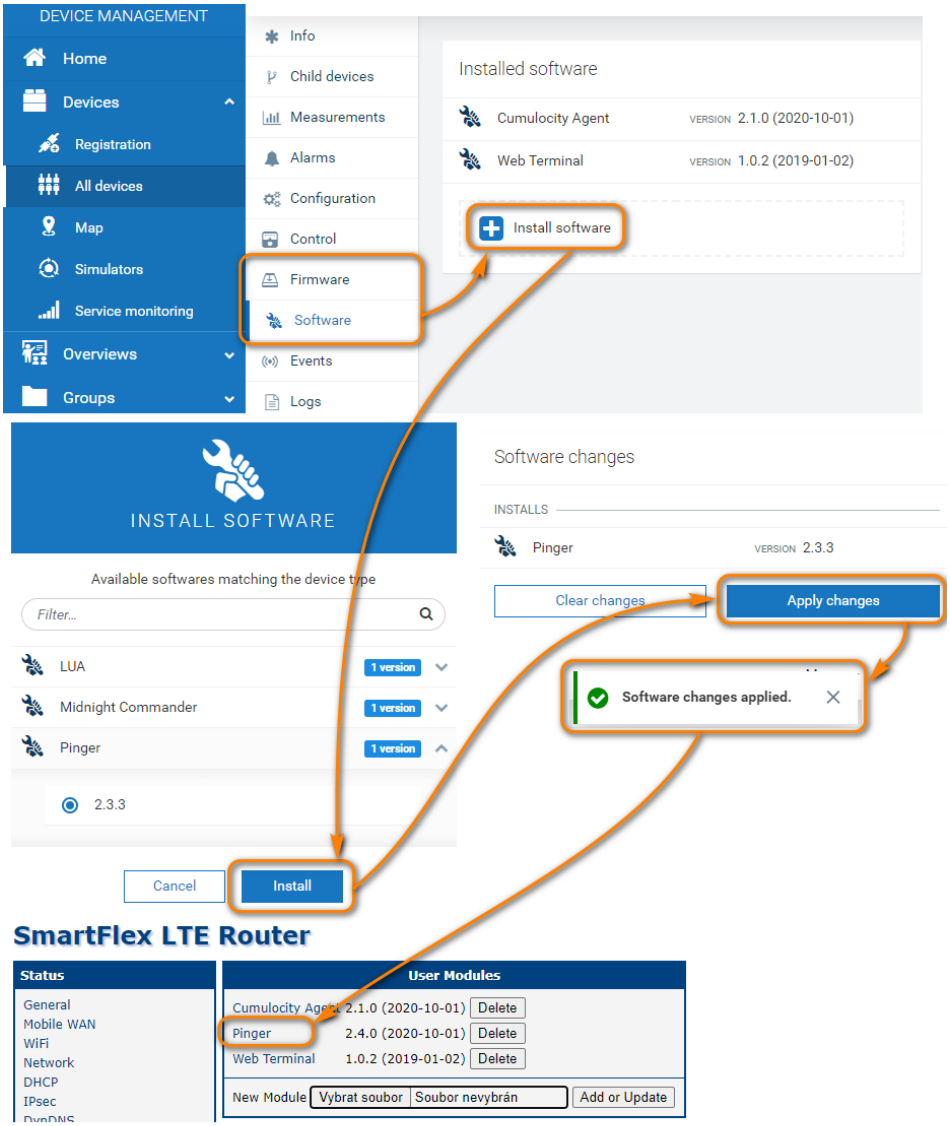



Figure 15: Remote install

### 1.3.5 Config management

Enables the server to get the configuration of the device. When checkbox is checked you can find *Configuration* menu item in the Devices Management / Devices / All devices / *your_router* / Configuration.

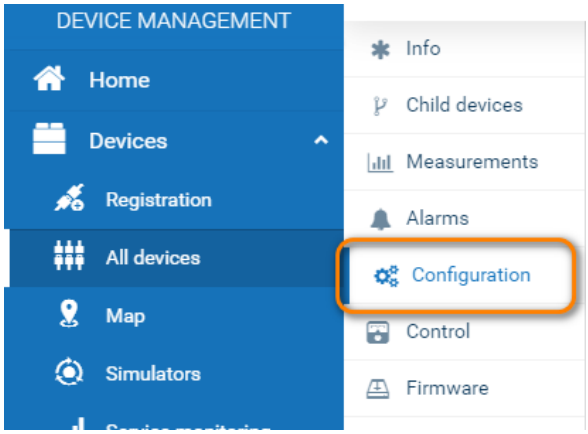



Figure 16: Configuration

**Config include users** means, that we could read list of the users and password hashs from downloaded config. That option might not be wanted due to security reasons.

When config is retrieved, record with result of this action appears under the Control menu item.

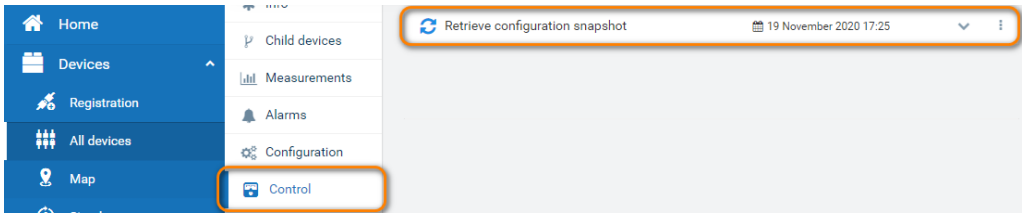



Figure 17: Config record

### 1.3.6 Modbus support

Enables the Cumulocity server to connect to the Modbus devices.

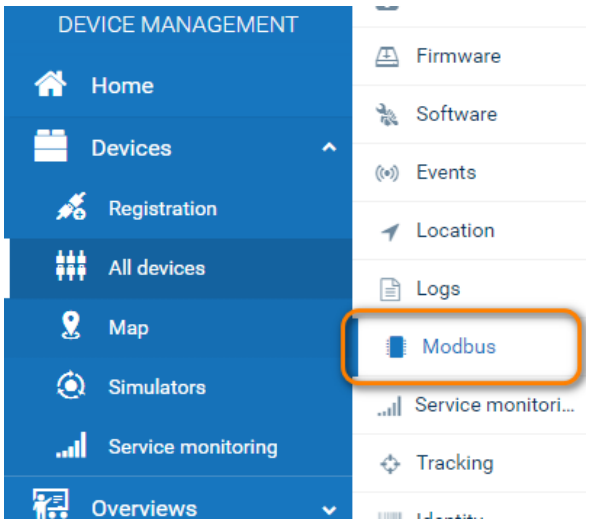



Figure 18: Configuration

For Modbus to work, you have to have Cumulocity full account and have Fieldbus activated in the Administration / Subscribed applications

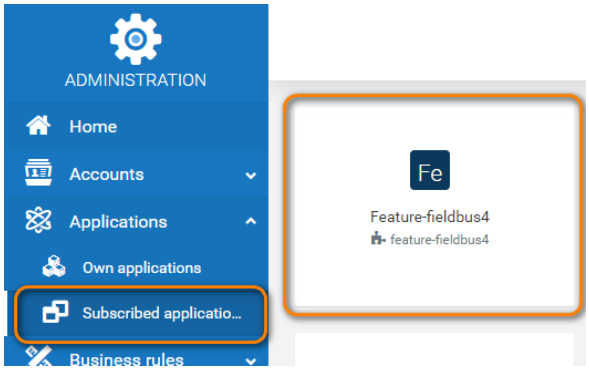



Figure 19: Subscribed applications

**Port**, **Baud rate**, **Parity**, **Stop bits** applies only for connection to Modbus via RS-485 or RS-232. Check the serial communication settings of the device according to the instructions provided with the Modbus device. These have to match with all devices on the bus.

**Transmit rate** declared how often are data send to cloud.

**Polling rate** says how often are data read from Modbus devices.

Transmit rate cannot be lower than polling rate. You might want to read data significantly more often, because there can be alarms etc. binded to that value. For example in case of water level sensor you might not want detailed records, but when water level rises too much, you need to react fast with alarm.

To define what data you want to get via Modbus, you have to register device registers in Device types / Device protocols. Once you define this definition, you can use it in multiple devices without the need of registering needed registers all over again for other devices.

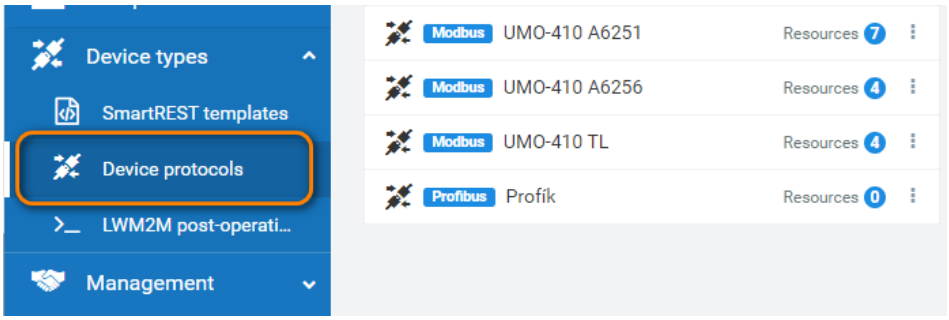



Figure 20: Device protocols

In this protocol you define binary coils/discrete inputs or value holding registers/input register (as usual for Modbus). But be aware, that there is write option even for discrete inputs/input registers. If you allow write (update in their terminology) you have to check "Allow write operations" on your router, otherwise the router reject those.

In protocol definition there are 3 switches:

- **Send measurements** – values received from Modbus device are taken as measurement data and will be displayed as graph.
- **Raise alarm** – If value is not zero, then alarm is raised.
- **Send event** – If value changes, alarm is registered.

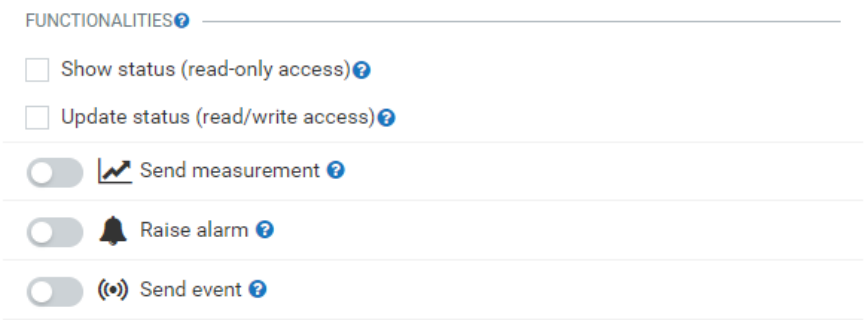



Figure 21: Modbus protocol functionalities





19-01-21

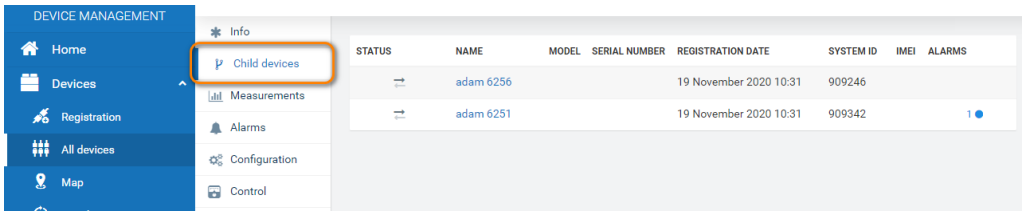



Figure 22: Modbus child devices

Results from those 3 switches are found in the Child devices menu item.

Remaining 2 options are:

- **Show status** – Displays value in widget.
- **Update status** – Value can be changed in widget and this value is written to modbus device (as mentioned before, *Allow write operations* has to be allowed on the router).

Results of those two remaining options can be seen in the Cockpit. Fieldbus device widget needs to be added to the cockpit.


Figure 23: Cockpit

Now, when protocols are defined, we could go to the devices Modbus menu where we could add TCP and RTU devices by entering addresses and choosing protocol defined before.

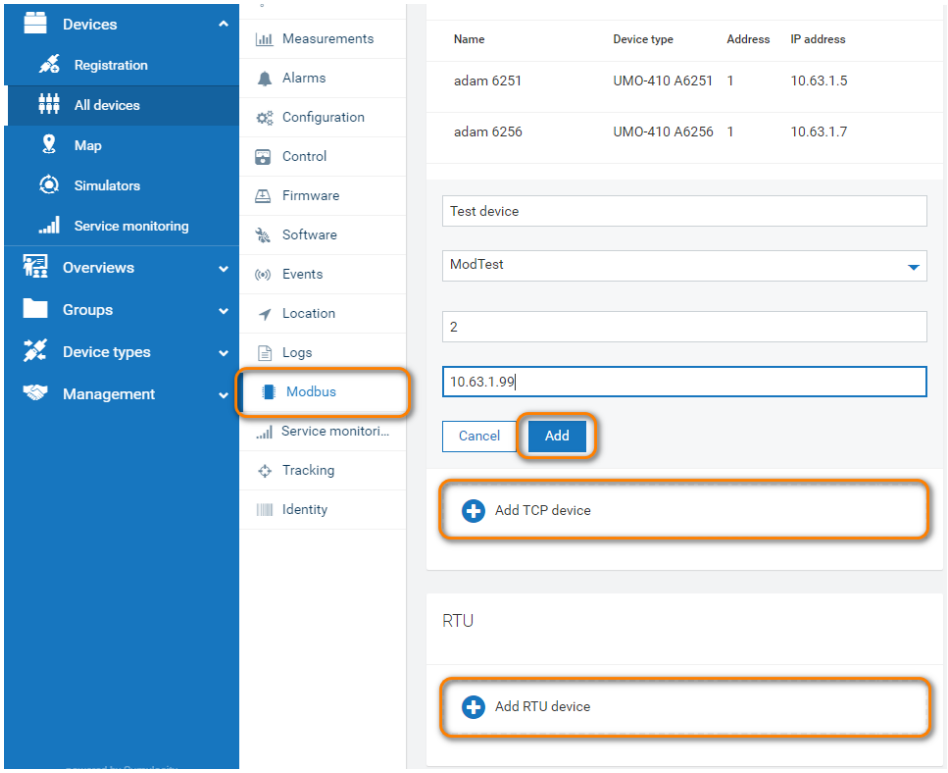



Figure 24: Modbus menu

As we mentioned before, results of our effort could be seen in *Child devices* and *Cockpit*

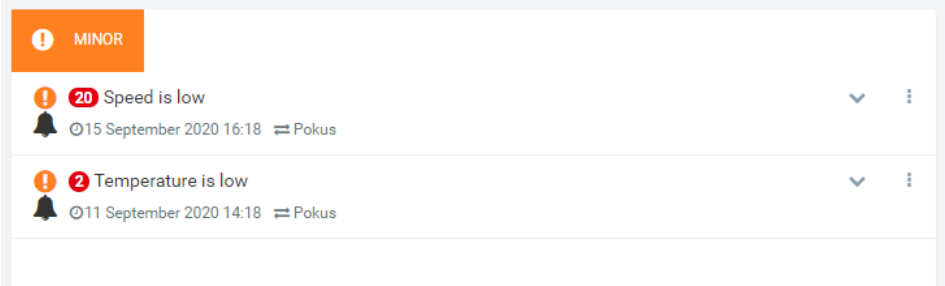



Figure 25: Child device alarm

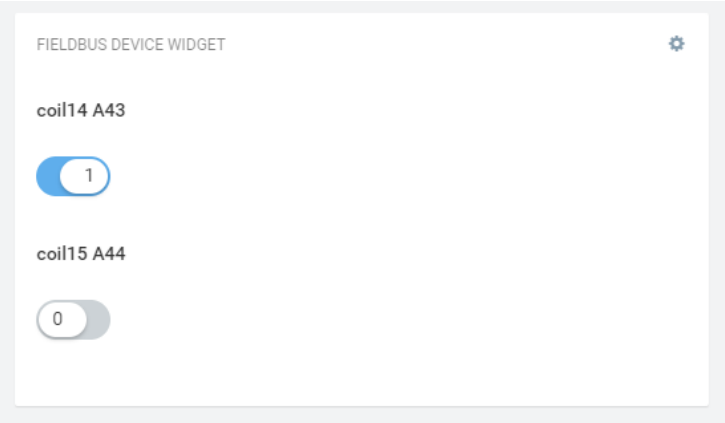



Figure 26: Cockpit widget

### 1.3.7 Telling geolocation

Enables the server to gain the device location remotely. It is functional for the routers having a GNSS module only. Map in the lower right corner in the selected device shows current location.

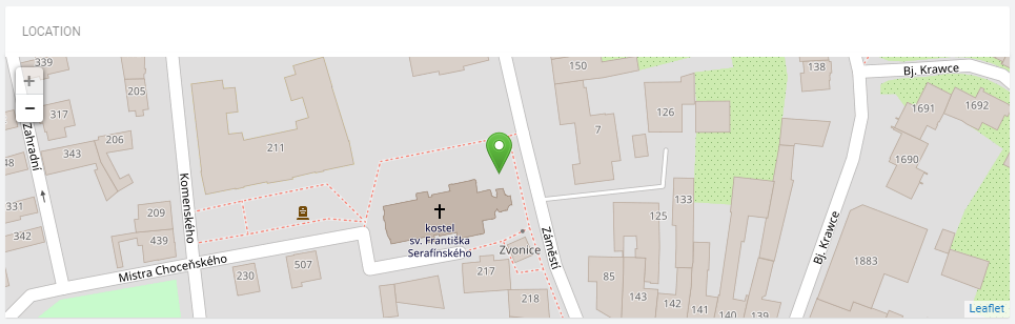



Figure 27: Geolocation

Due to its space requirements there is no support for Geolocation and LUA scripts with platform V2.

## 1.4 Status

The status of the daemon and agent running on the device can be observed on the *Daemon status* page, see Figure 28.

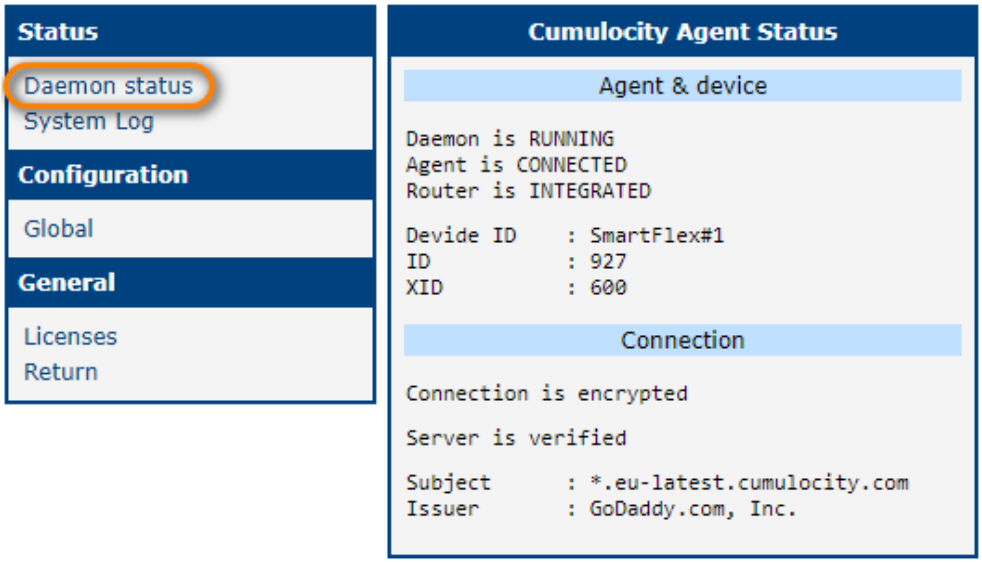



Figure 28: Daemon Status

There are following information that may appear on this page:

- Daemon is **RUNNING** / **NOT RUNNING** - Tells if the daemon process of the agent is running or not.
- Agent is **CONNECTED** / **NOT CONNECTED** - Tells if the network communication between the router and cloud is established or not.
- Router is **INTEGRATED** / **NOT INTEGRATED** - Tells if the router is registered and accepted on the cloud or not. If so, it will get the device ID.

The nexst status page called *System Log* displays the log of the router which is the same as *System Log* in the *Status* section of router's configuration.





19-01-21

## 1.5 LUA script

LUA script can be used to start a timer for a periodic task (e.g for sending of measurements) or to register a messages handler for cloud requests.

The length of the script is limited by 12000 characters. You can get around this limitation with splitting code to the modules and then load this module via `require()` command. LUA searches the modules in `/usr/lib/lua_modules` folder. You can also put third-party LUA modules to this folder or install some our router modules with LUA modules (e.g. *lua-libmodbus* for Modbus support in LUA scripts).

Every LUA script should have an `init()` function, which is processed on the module start. When this function finishes successfully, it must return 0, otherwise, a non-zero value.

A proprietary *SmartREST* protocol is used for a between the communication device and Cumulocity cloud. It is a standard REST protocol optimized for small data transfers. The message templates are used for this purpose. Each message template contains a REST query with variables. All templates are sent to a Cumulocity server and a future communication contains only values for variables. The server complete from the template and this values a final REST query. See official documentation for SmartREST details.

You can find the module's built-in message templates in `/opt/cumulocity/etc/srtemplate`. If you need the other template, put it to `/var/data/cumulocity/srtemplate`. The first line must contain a unique identifier. Use numbers between 900 and 999 for your messages to avoid conflicts with the module's built-in messages in the future. See the second example in Chapter 1.5.2 for details.

Due to its space requirements there is no support for Geolocation and LUA scripts with platform V2.

### 1.5.1 Example #1: Measurement

LUA script in this example is compatible with Cumulocity version 2.0.1.

The script below regularly sents the number of connected AP clients. For testing, you must define AP on a router and connect to it with some clients.

```
function init()
  timer = c8y:addTimer(1*10000, 'sendStat')
  timer:start()
  return 0
end

function sendStat()
  local file = io.popen('iw dev wlan0 station dump | grep Station | wc -l')
  local count = file:read('*n')
  file:close()
  c8y:send('200,' .. c8y.ID .. ',WifiAccessPoint,WifiAccessPoint,Clients,' .. count
      .. ',')
end
```

The `init()` function starts a timer which calls the `sendStat()` function in 10 seconds interval. The `sendStat()` function looks for the number of AP clients and sends this value to the cloud. It uses a built-in message 200 for a general measurement. See official documentation for Measurements.

### 1.5.2 Example #2: Shell commands handling

LUA script in this example is compatible with Cumulocity version 2.0.1.

This script waits for the shell commands from the Cumulocity server, executes it and returns results. This example needs other messages. Create file `/var/data/cumulocity/srtemplate` with following content:

```
lua_examples
11,901,,$.c8y_Command,$.id,$.deviceId,$.c8y_Command.text
10,902,PUT,/devicecontrol/operations/%%,application/json,,%%,UNSIGNED STRING STRING
    STRING,"{""status"":""%%"",""c8y_Command"":{""text"":""%%"", ""result"":""%%""}}"
```

The first line is a unique identifier. The second line is a definition of receiving requests. The third line is a definition for the sending of responses.

```
function init()
  c8y:addMsgHandler(901, 'shell')
  c8y:send('103,' .. c8y.ID .. ',"""c8y_Command"",""c8y_Restart"",""
      c8y_LogfileRequest"""')
  return 0
end

function run(cl)
  local output
  local file = io.popen(cl)
  if file then
    output = file:read('*a')
    file:close()
  end
  return output
end

function shell(r)
  local operID, devID, command = r:value(2), r:value(3), r:value(4)
  local cmd, arg = command:match('(%S+)%s+([%a%d%.%-]+)')
  if cmd ~= 'ping' and cmd ~= 'traceroute' or not arg then
    c8y:send('105,' .. operID .. ',"Not supported command"')
    return
  end
  if cmd == 'ping' then
    command = 'ping -c 3 ' .. arg
  end
```

```
  c8y:send('104,' .. operID .. ',EXECUTING')
  local output = run(command)

  if output then
    output = output:gsub('"', '""'):gsub('\n', '\\n')
    c8y:send('902,' .. operID ..',SUCCESSFUL,"' .. r:value(4)
.. '","' .. output .. '"')
    c8y:send('104,' .. operID .. ',SUCCESSFUL')
  else
    c8y:send('105,' .. operID .. ',"Can not execute command"')
  end
end
```

The `init()` function register a handler for the requests containing shell commands. The server needs to know which functionality the router supports. The second line in *init()* sends this information. As it overwrites a previous definition, you must repeat the capabilities previously send by module core. You can find it in the log with debug log level set as shown below.

```
2019-11-27 17:41:18 cumulocity[12290]: HTTP post: 15,9491 111,4581,"358709050508492,
    358709050508492",,"8942001210298048298, 8942001210298048298" 115,4581,"{""name
    "":""Cumulocity Agent"",""version"":""2.0.0 (2019-06-10)"",""url"":""""},{""name
    "":""GDB"",""version"":""1.0.0 (2015-04-30)"",""url"":""""},{""name"":""GPS"",""
    version"":""1.6.4 (2019-03-27)"",""url"":""""},{""name"":""Node-RED"",""version
    "":""2.0.0 (2019-09-19)"",""url"":""""},{""name"":""Node-RED / FTP"",""version
    "":""2.0.0 (2019-10-04)"",""url"":""""},{""name"":""Node-RED / Modbus"",""version
    "":""2.0.0 (2019-10-04)"",""url"":""""},{""name"":""Node-RED / Splunk event
    collector"",""version"":""1.0.1 (2019-03-22)"",""url"":""""},{""name"":""Node.js
    "",""version"":""2.0.0 (2019-09-19)"",""url"":""""}" 106,4581,"""dmesg"",""
    messages"",""module-nodered""" 103,4581,"""c8y_Restart"",""c8y_LogfileRequest""
    201,4581,0 202,4581,499,21.3 203,4581,77.8,73.9 204,4581,39 205,4581,11.4
```

The `shell()` function registered to handler cares for the shell command processing. Its argument contains data from the request, see the *Message Handler* in Chapter 1.5.3. A next part parses the received command. Remember this functionality is a potential security risk. So this example limits available commands to ping and traceroute only with one IP or domain argument and no options. Script change operation state on the server before executing. If execution is successful, the script returns a result. Then it closes operation on both cases successful and failure.

### 1.5.3 LUA c8y object

All LUA scripts have access to a c8y object. This object expose timers, SmartREST message based callbacks, HTTP binary APIs.

#### Timer

A timer provides periodical execution of a Lua function. Related APIs are as following:

- `c8y:addTimer(interval, callback) -> timer`

  Create a timer object with a time period of interval milliseconds. The Lua function with the name equals to the string callback will be executed when the timer fires. A timer is inactive when first created.

- `timer:start()`

  Start a timer object. timer becomes active.

- `timer:stop()`

  Stop a timer object. timer becomes inactive.

- `timer.isActive`

  Boolean property indicating whether the timer is active or not.

- `timer.interval`

  A property of the timer object. Read this property to get the current period of this timer. Write this property to set a new period for this timer. Setting a timer's interval to 0 is undefined behaviour.

#### Message Handler

A message handler is a SmartREST message callback, which is executed when a message for the registered SmartREST template ID is received.

- `c8y:addMsgHandler(MsgID, callback)`

  Register a message callback for message ID MsgID. The Lua function with the name equals to the string callback is executed when a message with ID MsgID is received. The callback function is called with a single SmartREST record parameter, see below about the record object APIs. Calling this function multiple times with the same MsgID overwrites the callback function.

- `record.size`

  Read-only property denotes the number of the tokens in the record.

- `record:value(i) -> string`

  Return the value of the token at position i. Index i starts from 0.

- `record:type(i) -> int`

  Return the type of the token at position i. Index i starts from 0.

### Networking

- `c8y:send(request, prio)`

  Send request to Cumulocity. Note this is asynchronous sending, this function returns immediately without any guarantee whether the request is sent successful or not. Set prio to 1 if you want the guarantee. Calling this function without prio or with prio equals 0 is the same default no guarantee sending. Be careful not to abuse the prio parameter, as there is a capacity for buffering failed requests, old requests will be discarded when this capacity is reached.

- `c8y:post(dest, ct, data) -> int`

  HTTP multipart POST to upload a binary to Cumulocity. dest is the file name to be stored on the server. ct is the Content-Type of data. data contains the actual content to be posted. This function is synchronous, on success, it returns the size of the response, on failure, -1 is returned.

- `c8y:postf(dest, ct, file) -> int`

  Counterpart of the post function that instead of asking the caller to read the data, the actual content to be posted is in a file pointed by parameter file. This function is also suitable when the file is large and can not be read once all in the memory. This function also returns -1 when read file failed.

- `c8y:get(id) -> int`

  HTTP GET method to download a binary from Cumulocity. id is the unique resource identifier of the binary. This function is synchronous, on success, it returns the size of the response, on failure, -1 is returned.

- `c8y:getf(id, dest) -> int`

  Counterpart of the get function, which instead of download a binary to memory, rather store the binary in a file pointed by dest. On success, the function returns the number of bytes written so far, on failure, the function returns -1. Note this function overwrites the file named dest if it exists.

- `c8y.resp`

  Read-only property stores the server response for post, postf and get. The content of c8y.resp is undefined when these functions failed.

### Misc

- `c8y.server`

  Server address the agent connects to.

- `c8y.ID`

  The managed object ID for the agent.

# 2. Cloud Administration

## 2.1 Cloud Account

Login to your Cumulocity account at *https://yourULR.softwareag.cloud/mycloud*, where *yourURL* is your domain set up during the registration.

If you don't have an account yet, create one at `https://www.softwareag.cloud/site/product/cumulocity-iot.html#/`, in the upper right corner click on *Try for free*. Then wait for the confirmation email and login to your cloud account at the address sent in this email. When logged in to the Cumulocity, choose the Cumulocity cloud product, as shown in Figure 29.

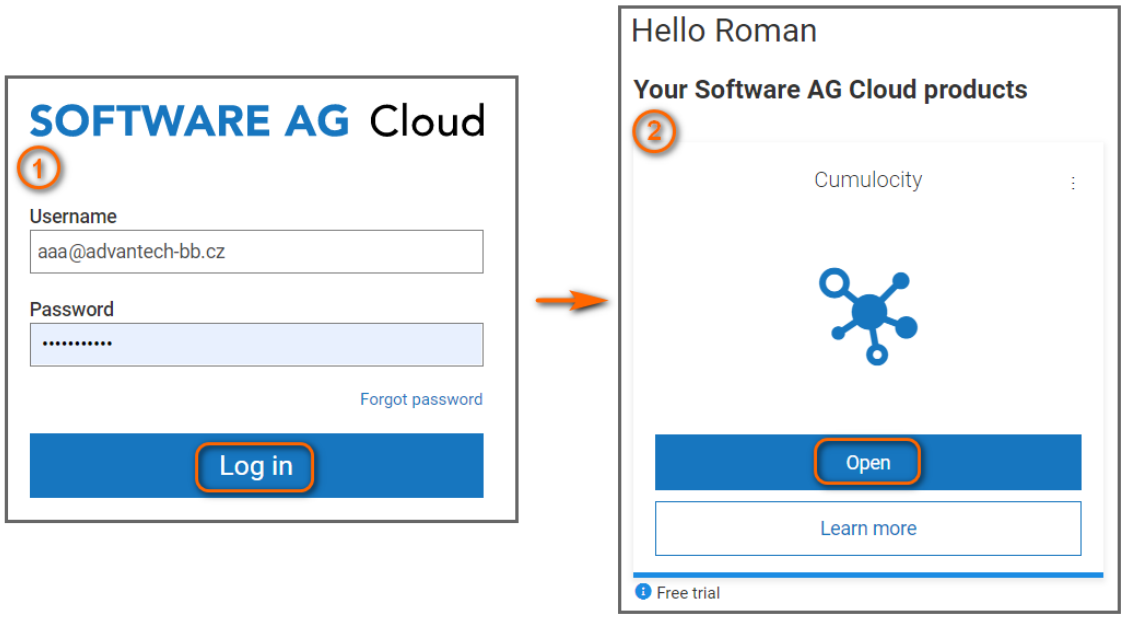



Figure 29: Cumulocity Cloud Login

## 2.2 Device Registration

To register a new device to the Cumulocity cloud go to *Devices -> Registration* section and follow the steps shown in Figure 30. Fill in the router's Device ID identically with the Device ID declared in Cumulocity Agent installed on the device.

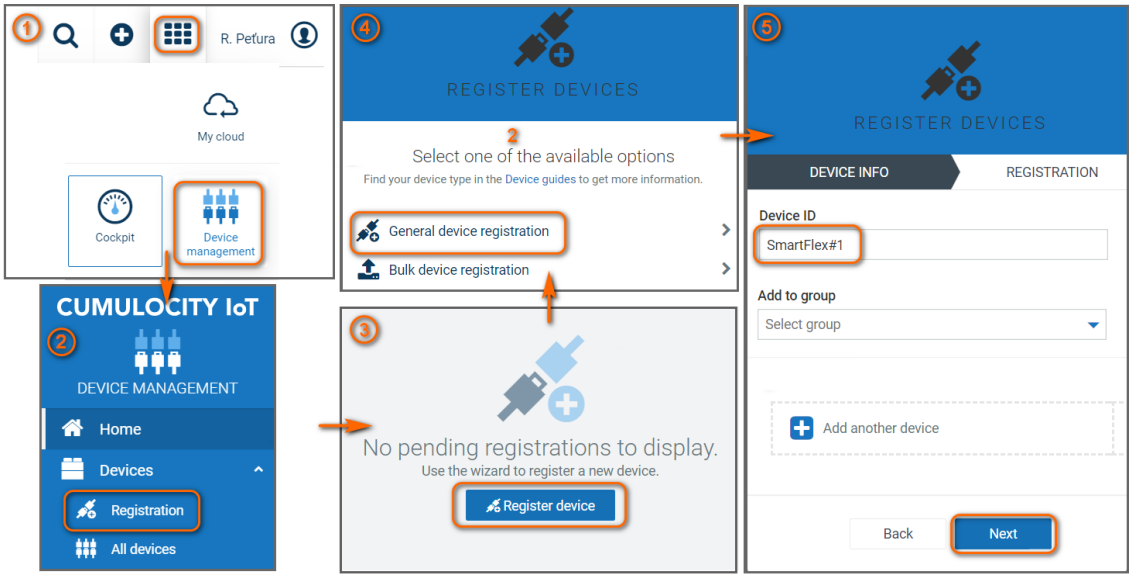



Figure 30: Device Registration

The device will appear in the list of devices with *waiting for connection* status. Cumulocity Agent has to be running and configured properly on the device to finish the registration successfully. Ensure that the device is connected to the Internet. Note, that the Cumulocity Agent may stop after some unsuccessful connection attempts. In a while, the status of the devices will change to *pending acceptance* and the *Accept* button will be available, see Figure 31. Click this button and the registration of the device is done.

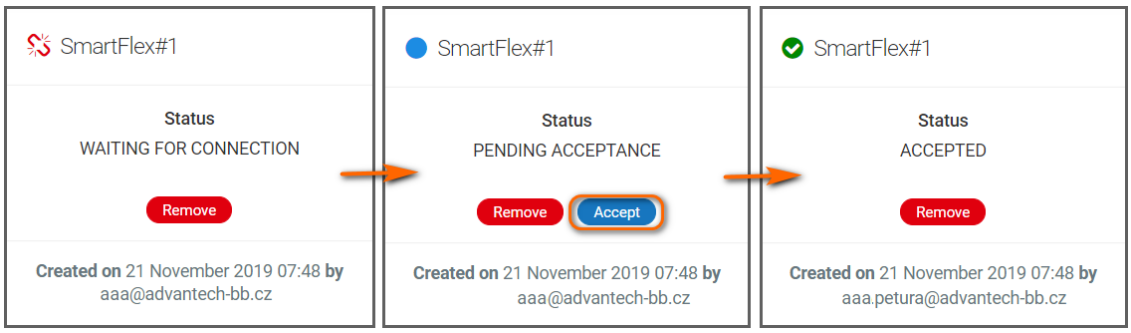



Figure 31: Confirmation of the Device Registration

## 2.3 Device Information

The successfully registered device is present in the *All devices* section, see Figure 32. It can take a few seconds prior to the device will appear in the list after clicking the acceptance.

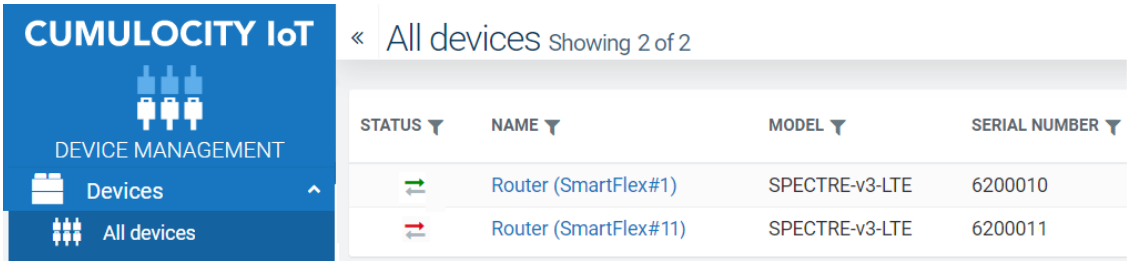



Figure 32: Device in the All Device List

Clicking a device in the *All devices* list, the basic information about the device appears, see Figure 33. There is information about the device name, type, model, serial number, ICCID, IMEI, etc. Some of this information can be edited here as well.

There are some other pages for the device administration, for example, Alarms, Control, Software, Events, Location or Logs as shown in Figure 33. Note that displayed device data and administration pages depend on the configuration made in the Cumulocity Agent on the device.

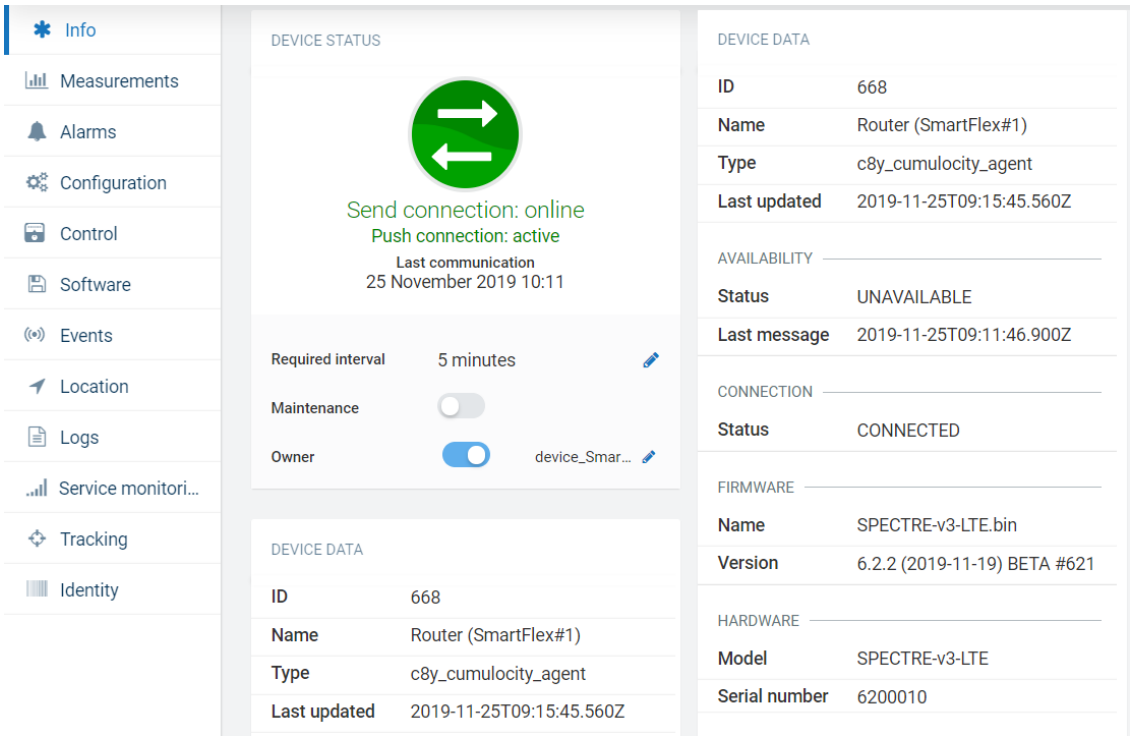



Figure 33: Device Information

# 3. FAQ

**Required Certificates**
If you define a repository with HTTPs sources in the Cumulocity cloud, you need the Root CA certificates on the router to be able to download data from this source. These certificates or symlinks to them should be located in `/opt/cumulocity/etc/certs` directory having the name by hash (see *c_rehash* OpenSSL tool).

**Where is Network Card Information on Cumulocity Website?**
The router has very complex network settings and the default Cumulocity network plugin is very simple so it cannot reflect the whole router's networking.

**Files with Credentials**
Router agent stores credentials into `/var/data/cumulocity/credentials`. If required, these credentials can be deleted or backed-up eventually. This file is deleted while the module is being uninstalled.

**Device ID was changed**
If the device ID, which already was registered to the cloud, was changed, the device won't be able to connect to the cloud. To fix this issue delete the credentials file located here: `/var/data/cumulocity/credentials`.

**I can't see Modbus menu item even if I have checked *Enable Modbus support***
Make sure you have subscribed Fieldbus application (feature-fieldbus4) in Administration/Applications/Subscribed applications





19-01-21

# 4. Related Documents


**[1]** Advantech Czech: **v2 Routers – Configuration Manual** (MAN-0021-EN)
**[2]** Advantech Czech: **SmartFlex – Configuration Manual** (MAN-0023-EN)
**[3]** Advantech Czech: **SmartMotion – Configuration Manual** (MAN-0024-EN)
**[4]** Advantech Czech: **SmartStart – Configuration Manual** (MAN-0022-EN)
**[5]** Advantech Czech: **ICR-3200 – Configuration Manual** (MAN-0042-EN)
**[6]** Advantech Czech: **User Module GPS – Application Note** (APP-0015-EN)


**[EP]** Product related documents and applications can be obtained on *Engineering Portal* at `https://ep.advantech-bb.cz/` address.