Vendor Device Certification Requirements

The device certification process requires the device to follow integration best practices. This means that more fields are mandatory for a certified device compared to platform minium requirements. In the following section, mandatory information and behavior is described.

Currently Supported Device Capabilities of Self-Service Certification Microservice

- ☐ Foundation Modules
  - ☐ Device Information
    - ☑ type
    - ☐ c8y_Agent
    - ☑ c8y_Hardware
    - ☑ c8y_Firmware
    - ☑ c8y_RequiredAvailability
    - ☑ c8y_SupportedOperations
  - ☑ External ID
  - ☑ Sending Operational Data
    - ☑ Measurements
    - ☑ Events
    - ☑ Alarms
- ☐ Optional Modules
  - ☑ Gateways
    - ☑ Child Device Types
  - ☑ Log File Retrieval
  - ☑ Device Configuration
    - ☑ Text Based Configuration
    - ☑ File Based Configuration
  - ☑ Managing Device Software
  - ☑ Managing Device Firmware
  - ☑ Managing Device Firmware
  - ☑ Device Profile
  - ☑ Restart
  - ☐ Measurement Request
  - ☐ Shell
  - ☐ Cloud Remote Access
  - ☐ Location & Tracking

Foundation Modules (Mandatory)

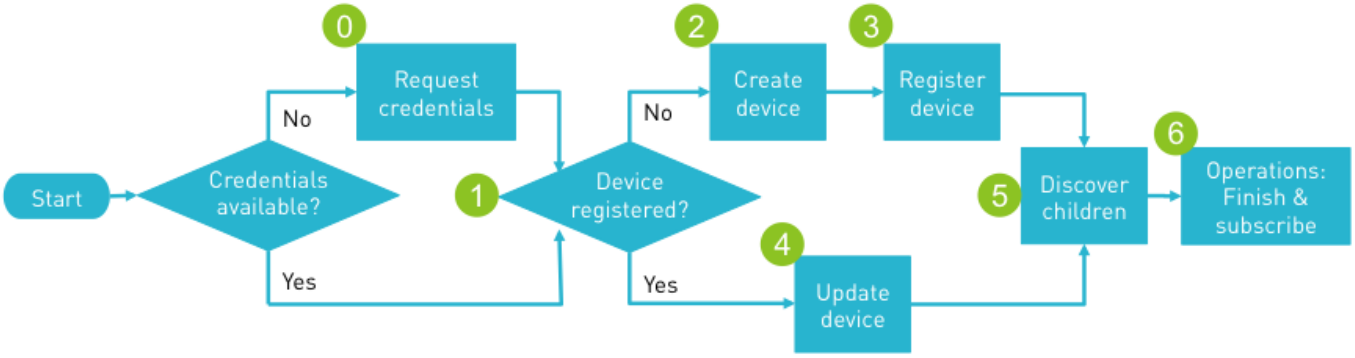
Status: Reviewed and Ready

Device Behavior

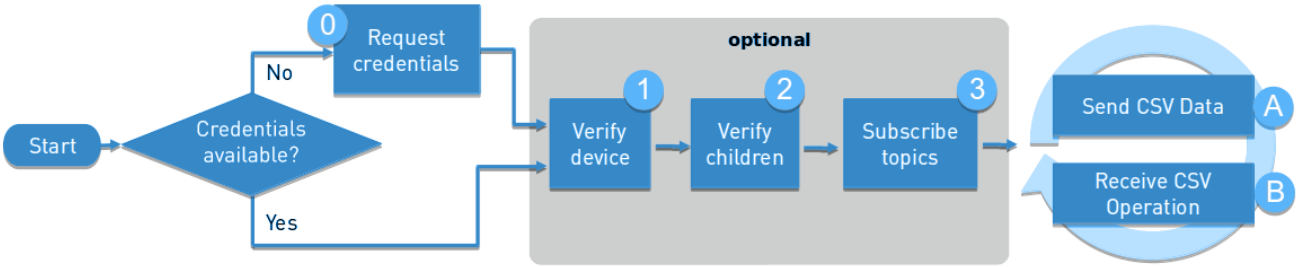
When started, the device follows the process flow defines for REST or MQTT based integration respectively.

Please read and follow one of these guides:

For REST based integrations:



For MQTT based integrations (using Smart-Rest 2 is recommended):



Cumulocity IoT fulfills SSL Labs A+ rating and therefor supports exclusively the following cypher suits from release Release 10.10:

- rsa_pkcs1_sha256
- dsa_sha256
- ecdsa_secp256r1_sha256
- rsa_pkcs1_sha384
- ecdsa_secp384r1_sha384
- rsa_pkcs1_sha512
- ecdsa_secp521r1_sha512
- rsa_pss_rsae_sha256
- rsa_pss_rsae_sha384
- rsa_pss_rsae_sha512
- ed25519 ed448
- rsa_pss_pss_sha256
- rsa_pss_pss_sha384
- rsa_pss_pss_sha512

Device Information

For details and examples, compare metadata section of documentation as well as the detail sections below.

Fragment	Meaning in Device Partner Portal	Mandatory
`c8y_IsDevice`	Empty fragment. Declares a Managed Object as a Device	Yes
`com_cumulocity_model_Agent`	Empty fragment. Declares that the device is able to receive operations	Yes, for root devices and gateways that support operations; No, for devices and gateways that don't support operations; Must not be used for child devices;
`name`	Sets the name of the device used e.g. in 'all devices' and 'device info' views	Yes
`type`	Functional type of device e.g. water meter, pump, Gateway, environmental sensor	Yes
`c8y_Hardware`	Hardware information about the device	Yes
`c8y_Firmware`	Firmware information about the device	Yes
`c8y_Agent`	Information about the agent run on the device	Yes
`c8y_RequiredAvailability`	Minimal communication interval to determine if device is offline	No
`c8y_SupportedOperations`	Many optional operations are directly triggering the dynamic UI by invoking the respective operation tabs	No
`externalIds`	Used to identify a device with a unique information from the physical world	Yes

Example structure in device inventory:

```
"c8y_IsDevice": {},
"com_cumulocity_model_Agent": {},
```

```
  "name": "edge-agent-eabdcf5d344b4a52a4ffab13fe3d11cb",
  "type": "c8y_EdgeAgent",
  "c8y_Hardware": {
      "model": "BCM2708",
      "revision": "000e",
      "serialNumber": "00000000e2f5ad4d"
  },
  "c8y_Firmware": {
      "name": "raspberrypi-bootloader",
      "version": "1.20140107-1",
      "url": "31aab9856861b1a587e2094690c2f6e272712cb1"
  },
  "c8y_RequiredAvailability": {
      "responseInterval": 6
  },
  "c8y_SupportedOperations": [
      "c8y_Software",
      "c8y_Firmware"
  ]
```

type

The fragment `type` can be interpreted as *device class*. Meaning, devices with the same `type` can receive the same types of configuration, software, firmware, and operations. The Cumulocity IoT UI uses the device `type` often for filtering purposes like sending a software package to all devices of one specific `type`. Based on the device `type` Cumulocity can assign Dashboards to the same `type`.

Fragment	Mandatory
`type`	Yes

Example structure in device inventory:

```
  "type": "c8y_EdgeAgent"
```

c8y_Hardware

The device certificate will be issued for device defined by: `c8y_Hardware.model`, `c8y_Hardware.revision`, `c8y_Firmware.name`, `c8y_Firmware.version`, `c8y_Agent.name`, and `c8y_Agent.version`. These fragments will also be used in future versions of Device Partner Portal (display one "Device" entry in the overview device list per `c8y_Hardware.model` and a dropdown in the device detail view for each `c8y_Hardware.revision`).

Fragment	Meaning in Device Partner Portal	Mandatory
`model`	Device in list view	Yes
`revision`	Dropdown inside device detail view to select device revision or version	Yes

Fragment	Meaning in Device Partner Portal	Mandatory
`serialNumber`	Not used in Device Partner Portal	Yes

Example structure in device inventory:

```
"c8y_Hardware": {
    "model": "BCM2708",
    "revision": "000e",
    "serialNumber": "00000000e2f5ad4d"
}
```

c8y_Firmware

The device certificate will be issued for device defined by: `c8y_Hardware.model`, `c8y_Hardware.revision`, `c8y_Firmware.name`, `c8y_Firmware.version`, `c8y_Agent.name`, and `c8y_Agent.version`.

Fragment	Mandatory
`name`	Yes
`version`	Yes
`url`	No

Example structure in device inventory:

```
"c8y_Firmware": {
    "name": "raspberrypi-bootloader",
    "version": "1.20140107-1",
    "url": "31aab9856861b1a587e2094690c2f6e272712cb1"
}
```

c8y_Agent

The device certificate will be issued for device defined by: `c8y_Hardware.model`, `c8y_Hardware.revision`, `c8y_Firmware.name`, `c8y_Firmware.version`, `c8y_Agent.name`, and `c8y_Agent.version`.

Fragment	Mandatory
`name`	Yes
`version`	Yes
`url`	No

Example structure in device inventory:

```
"c8y_Agent": {
    "name": "myCustomAgent",
    "version": "1.2.34",
    "url": "https//link-to-agent-repo.url"
}
```

c8y_RequiredAvailability

For details and examples, compare device availability section of the documentation.

Fragment	Mandatory
`responseInterval`	No

Example structure in device inventory:

```
"c8y_RequiredAvailability": {
    "responseInterval": 6
}
```

c8y_SupportedOperations

The fragment `c8y_SupportedOperations` is used to identify which operations (and hence certifiable optional modules) are supported by the device. This fragment is optional. If not present, no optional modules will be certified.

Fragment	Mandatory
`c8y_SupportedOperations`	No

Example structure in device inventory:

```
"c8y_SupportedOperations": [
    "c8y_Software",
    "c8y_Firmware"
]
```

External ID

For details and examples, compare external id section of the documentation.

Used to identify the device in Cumulocity by its unique serial number, MAC, IMEI or similar unique identification string. If you don't want to specify a type, its recommend to use `c8y_Serial`.

Fragment	Mandatory

Fragment	Mandatory
`externalId`	Yes
`type`	Yes

Example structure in external id managed object (this information is not stored in the device inventory):

```
"externalIds": [
        {
            "externalId": "edge-agent-eabdcf5d344b4a52a4ffab13fe3d11cb",
            "type": "c8y_Serial"
        }
    ]
```

Sending Operational Data

Sending measurements, events, and alarms are basic capabilities of any IoT enabled device. Therefore vendors should aim to support all three. However, there might be some instance where devices only send events (e.g. basic switches) or only measurements (e.g. basic sensor). It is only mandatory to send either measurements, or events, or alarms in order to get certified while it is still recommended to implement all three capabilities.

Functionality	Content	Mandatory
Measurements (M), Events (E), Alarms (A)	Information send from the device to the platform	Yes, at least one of the three

Measurements

For details and examples, compare measurements section of the documentation. It is only mandatory to send either measurements, or events, or alarms in order to get certified while it is still recommended to implement all three capabilities.

The device creates measurements with the following content:

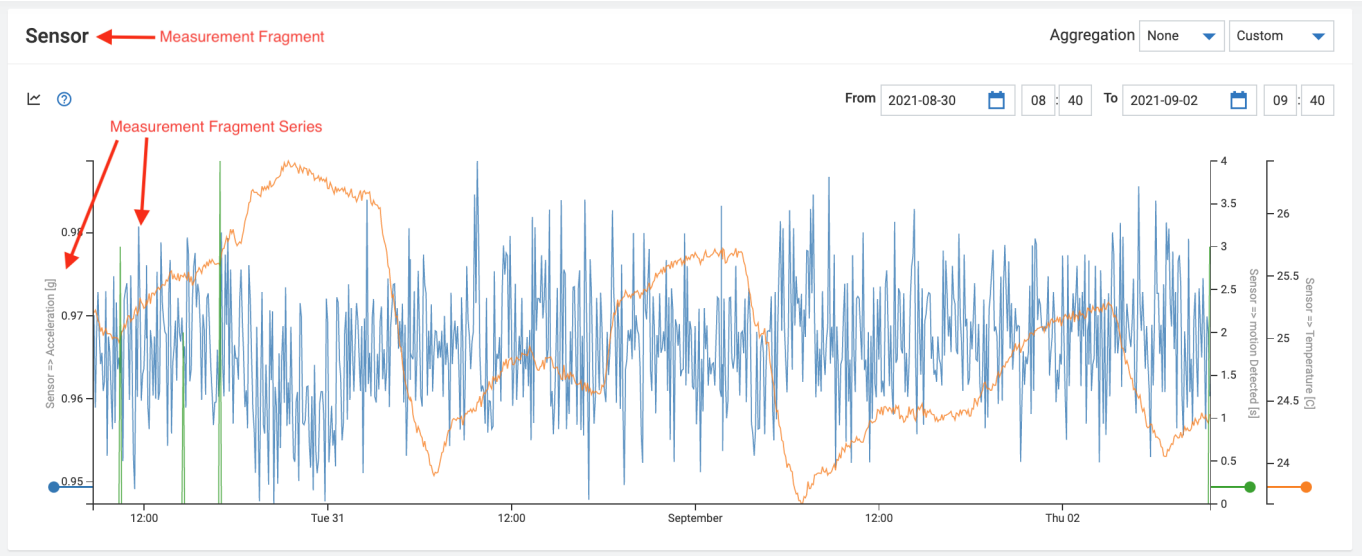
Fragment	Content	Mandatory for Measurements
`source`	Device ID	Yes
`type`	Type of measurement	Yes
`time`	Date and time when the measurement was made	Yes
Measurement Fragment	The category of measurement	Yes
Measurement Fragment Series	The name of the measurement series. Contains at least the `value` fragment, optionally the `unit` fragment	Yes

Measurement names should be written in camel-case. Cumulocity IoT UI inserts a blank space between a lower-case and an upper-case letter. Two or more consecutive upper-case letters are not separated with blank spaces. The UI also hides the prefix of a measurement name that is ending with a "_" (underline) symbol.

Example POST body:

```
{
  "source": {
    "id": "251982",
  },
  "type": "c8y_TemperatureMeasurement",
  "time": "2021-10-19T12:03:27.845Z",
  "c8y_Steam": {
    // Measurement Fragment
    "Temperature": {
      // Measurement Fragment Series
      "unit": "C",
      "value": "100",
    },
  },
}
```

The *Measurement Fragment* and *Measurement Fragment Series* are used in the Cumulocity IoT UI in the following way:



The following *Measurement Fragments* are standard measurement fragments in Cumulocity IoT: `c8y_AccelerationMeasurement`, `c8y_Battery`, `c8y_CPUMeasurement`, `c8y_CurrentMeasurement`, `c8y_DistanceMeasurement`, `c8y_HumidityMeasurement`, `c8y_LightMeasurement`, `c8y_MeasurementPollFrequencyOperation`, `c8y_MeasurementRequestOperation`, `c8y_MemoryMeasurement`, `c8y_MotionMeasurement`, `c8y_MoistureMeasurement`, `c8y_SignalStrength`, `c8y_SinglePhaseEnergyMeasurement`, `c8y_Steam`, `c8y_Temperature`, `c8y_TemperatureMeasurement`, `c8y_ThreePhaseEnergyMeasurement`, `c8y_VoltageMeasurement`,

Events

For details and examples, compare events section of the documentation. It is only mandatory to send either measurements, or events, or alarms in order to get certified while it is still recommended to implement all three capabilities.

The device creates events with the following content:

Fragment	Content	Mandatory for Events
`source`	Device ID	Yes
`type`	Type of event	Yes
`time`	Date and time when the event was created	Yes
`text`	Description of the event	Yes

Example POST body:

```
{
  "source": {
    "id": "251982",
  },
  "type": "Intrusion detection",
  "text": "Door sensor was triggered",
  "time": "2021-10-19T12:03:27.845Z",
}
```

Alarms

For details and examples, compare alarms section of the documentation. It is only mandatory to send either measurements, or events, or alarms in order to get certified while it is still recommended to implement all three capabilities.

The device creates alarms with the following content:

Fragment	Content	Mandatory for Alarms
`source`	Device ID	Yes
`type`	Type of alarm	Yes
`time`	Date and time when the alarm was created	Yes
`text`	Description of the alarm	Yes
`severity`	One of the following severities: `CRITICAL`, `MAJOR`, `MINOR`, `WARNING`	Yes
`status`	`ACTIVE` or `CLEARED`. If not specified, a new alarm will be created as `ACTIVE`. The state `ACKNOWLEDGED` is only set by the user, not by the device.	No

Example POST body:

```
{
  "source": {
    "id": "251982",
  },
  "type": "Operational State Alarms",
  "text": "Machine stopped unexpectedly with exit reason 3",
  "severity": "MAJOR",
  "status": "ACTIVE",
  "time": "2021-10-19T12:03:27.845Z",
}
```

Optional Modules

All sections below are **optional**. If a device partner decides to certify optional modules, they are documented in the certificate and shown on the device partner portal. Customer can filter and search for devices that support certain modules. Therefore, it is recommended to certify all capabilities (aka. "Optional Modules") offered by the device. The Modules are listed below in descending order of importance based on Software AG's experience. To indicate that a device wants to certify an Optional Module, it has to add the respective element to the list of supported operations in the inventory object of the device. All optional modules that recieve operations require the fragment `com_cumulocity_model_Agent` to be present in the inventory as described in Device Information.

Example structure in device inventory:

```
"c8y_SupportedOperations": [
    "c8y_LogfileRequest",
    "c8y_Configuration",
    "c8y_SendConfiguration"
]
```

Gateways

For details and examples, compare child operations section of the documentation.

Child Device Types

Cumulocity uses the concept of *child device types* to distinguish the capabilities of child devices *behind* a gateway device. For example, a child device connected via a simple analog wire connection (like a temperature sensor) may only be able to send measurements (the temperature), while a child device connected via OPC-UA is able to send measurements, events, alarms, and log files in addition to the ability to process incoming requests to upgrade its firmware. In this case, the child device type `Analog` is only supporting the minimum requirements for certification without any *optional modules*. The child device type `OPC-UA` is supporting the *foundation modules* as well at the optional modules `Logs` and `Firmware`.

Child device types can be freely named, however, here are some examples as orientation:.

Fragment	Content	Required for optional module
`c8y_SupportedChildDeviceTypes`	List contains supported child device types	Yes

Example structure in device inventory:

```
"c8y_SupportedChildDeviceTypes": [
    "Analog",
    "Canbus",
    "CanOpen",
    "Modbus",
    "OPCUA",
    "Profibus",
    "Sigfox",
    "SNMP"
]
```

Log File Retrieval

Device capability to upload (filtered) log files to C8Y. For details and examples, compare chapter `c8y_LogfileRequest` of section miscellaneous in the documentation.

The following fragments are related to the optional device capability with a remark if they are required for the capability to work:

Fragment	Content	Required for optional module
`com_cumulocity_model_Agent`	Must be present in the inventory; Enables a device to receive operations	Yes
`c8y_SupportedOperations`	List contains element `c8y_LogfileRequest`	Yes
`c8y_SupportedLogs`	List of supported log file types	Yes (at least 1 type)

Example structure in device inventory:

```
"c8y_SupportedOperations": [
    "c8y_LogfileRequest"
],
"c8y_SupportedLogs": [
    "syslog",
    "dmesg"
]
```

Example operation sent to the device:

```
"c8y_LogfileRequest": {
    "logFile": "syslog",
    "dateFrom": "2016-01-27T13:45:24+0100",
    "dateTo": "2016-01-28T13:45:24+0100",
    "searchText": "sms",
    "maximumLines": 1000
}
```

When the device receives the operation `c8y_LogfileRequest`, the following steps are executed:

Step	Action	Documentation
0.	Listen for operation created by platform with `"status" : "PENDING"`	Real-time notifications
1.	Update operation `"status" : "EXECUTING"`	Update operation
2.	Internally retrieve log file and filter w.r.t. criteria found in operation	
3.	Create an event with `"type": "c8y_LogfileRequest"`	Create event
4.	Upload the log file as attachment to the event	Attach file to event
5.	Update operation accordingly `"status": "SUCCESSFUL", "c8y_LogfileRequest": {"file": "https://<TENANT_DOMAIN>/event/events/{id}/binaries"`	Update operation

Device Configuration

Device capability that enables text- and / or profile-based device configuration. Text based configuration is the more basic approach. It provides a plain text box in the UI to retrieve, edit, and send a configuration text to the device. The text is send as one string, however, it can be structured using json, xml, key-value pairs or any other markup that the device is able to parse. File based configuration allows to have multiple *types* of configurations (e.g. one file for defining polling intervals and another to configure the internal log-levels). For details and examples, compare configuration management section in the documentation.

For a successful certification of the Device Configuration module, Text Based Configuration or File Based Configuration or both have to be implemented. The certificate will state which configuration methods is supported as information.

Text Based Configuration

The following fragments are related to the optional device capability with a remark if they are required for the capability to work:

Fragment	Content	Required for optional module

Fragment	Content	Required for optional module
`com_cumulocity_model_Agent`	Must be present in the inventory; Enables a device to receive operations	Yes
`c8y_SupportedOperations`	List contains element `c8y_Configuration`	Yes
`c8y_SupportedOperations`	List contains element `c8y_SendConfiguration`	Yes

Example structure in device inventory:

```
"c8y_SupportedOperations": [
    "c8y_Configuration",
    "c8y_SendConfiguration"
],
"c8y_SendConfiguration": {}
```

Example operation `c8y_Configuration: {}` as it is sent to the device:

```
    "creationTime": "2021-09-20T13:10:25.933Z",
    "deviceId": "181119",
    "self": "https://t635974191.eu-
latest.cumulocity.com/devicecontrol/operations/440452",
    "id": "440452",
    "status": "PENDING",
    "c8y_Configuration": {
        "config": "test"
    },
    "description": "Configuration update"
```

When the device receives the operation `c8y_Configuration`, the following steps are executed:

Step	Action	Documentation
0.	Listen for operation created by platform with `"status" : "PENDING"`	Real-time notifications
1.	Update operation `"status" : "EXECUTING"`	Update operation
2.	Internally interpret transmitted string and execute configuration	
3.	Update operation accordingly `"status": "SUCCESSFUL"`	Update operation

Example operation `c8y_SendConfiguration: {}` as it is sent to the device:

```
creationTime: "2021-09-20T13:53:29.419Z", deviceName: "123456789", deviceId:
"440366",…
c8y_SendConfiguration: {}
```

```
creationTime: "2021-09-20T13:53:29.419Z"
description: "Requested current configuration"
deviceId: "440366"
deviceName: "123456789"
id: "440472"
self: "https://t635974191.eu-
latest.cumulocity.com/devicecontrol/operations/440472"
status: "PENDING"
```

When the device receives the operation `c8y_SendConfiguration`, the following steps are executed:

Step	Action	Documentation
0.	Listen for operation created by platform with `"status" : "PENDING"`	Real-time notifications
1.	Update operation `"status" : "EXECUTING"`	Update operation
2.	Internally get current configuration and update the fragment `c8y_Configuration` of the device inventory object	Update managed object
3.	Update operation accordingly `"status": "SUCCESSFUL"`	Update operation

File Based Configuration

The following fragments are related to the optional device capability with a remark if they are required for the capability to work:

Fragment	Content	Required for optional module
`com_cumulocity_model_Agent`	Must be present in the inventory; Enables a device to receive operations	Yes
`c8y_SupportedOperations`	List contains element `c8y_DownloadConfigFile`	Yes
`c8y_SupportedOperations`	List contains element `c8y_UploadConfigFile`	Yes
`c8y_SupportedConfigurations`	List of supported configuration file types	Yes (at least 1 type)

Example structure in device inventory:

```
"c8y_SupportedOperations": [
    "c8y_DownloadConfigFile",
    "c8y_UploadConfigFile"
    ],
"c8y_SupportedConfigurations":[
    "config1",
```

```
    "config2"
    ]
```

Example operation sent to the device for `c8y_DownloadConfigFile`:

```
"c8y_DownloadConfigFile": {
    "url": "<download url>"
}
```

When the device receives the operation `c8y_DownloadConfigFile`, the following steps are executed:

Step	Action	Documentation
0.	Listen for operation created by platform with `"status" : "PENDING"`	Real-time notifications
1.	Update operation `"status" : "EXECUTING"`	Update operation
2.	Download referenced binary and internally apply configuration	
3.	Update operation `"status": "SUCCESSFUL"`	Update operation

Example operation sent to the device for `c8y_UploadConfigFile`:

```
"c8y_UploadConfigFile":{}
```

When the device receives the operation `c8y_UploadConfigFile`, the following steps are executed:

Step	Action	Documentation
0.	Listen for operation created by platform with `"status" : "PENDING"`	Real-time notifications
1.	Update operation `"status" : "EXECUTING"`	Update operation
2.	Internally retrieve the configuration type requested in operation e.g. `"c8y_UploadConfigFile": {"type": "someConfig"}`	
3.	Create an event with the same `type` e.g. `"type": "someConfig"`	Create event
4.	Upload the configuration as attachment to the event	Attach file to event
5.	Update operation accordingly `"status": "SUCCESSFUL"`	Update operation

Managing Device Software

Device capability to manage and deploy software packages to the device. For details and examples, compare c8y_SoftwareList section in the documentation .

Note: *Firmware Management* and *Software Management* are handled separately in Cumulocity IoT and follow different concepts. A device can support one ore both capabilities.

Firmware

- The firmware is the base operating system of a device
- A device can only have 1 firmware installed at a time
- When the firmware is changed, the device usually flashes itself
- Usually used by small / embedded devices

Software

- Software is an optional component executed on the device
- A device can have multiple software installed at a time
- Software can be installed or removed independently of other software and the firmware
- Usually used by larger / more powerfully devices

Info: `"c8y_SupportedOperations": ["c8y_SoftwareList"]` (not to be confused with the *inventory fragment* `c8y_SoftwareList`) and `"c8y_SupportedOperations": ["c8y_Software"]` are deprecated and should not be used for new agent implementations.

The following fragments are related to the optional device capability with a remark if they are required for the capability to work:

Fragment	Content	Required for optional module
`com_cumulocity_model_Agent`	Must be present in the inventory; Enables a device to receive operations	Yes
`c8y_SupportedOperations`	List contains element `c8y_SoftwareUpdate`	Yes
`c8y_SoftwareList`	List of currently installed software on the device	Yes

Example structure in device inventory:

```
"c8y_SupportedOperations": [
    "c8y_SoftwareUpdate"
],
"c8y_SoftwareList": [
    {
        "name": "Software A",
        "version": "1.0.1",
        "url": "www.some-external-url.com"
    },
    {
        "name": "Software B",
        "version": "2.1.0",
```

```
            "url": "mytenant.cumulocity.com/inventory/binaries/12345"
        }
    ]
```

Example operation sent to the device:

```
"c8y_SoftwareUpdate": [
    {
        "name": "mySoftware1",
        "version": "1.0.0",
        "url": "http://www.example.com",
        "action": "install"
    },
    {
        "name": "mySoftware2",
        "version": "1.1.0",
        "url": "http://www.example.com",
        "action": "install"
    }
]
```

When the device receives the operation `c8y_SoftwareUpdate`, the following steps are executed:

Step	Action	Documentation
0.	Listen for operation created by platform with `"status" : "PENDING"`	Real-time notifications
1.	Update operation `"status" : "EXECUTING"`	Update operation
2.	Execute software install / uninstall tasks described in the operation	
3.	Update `c8y_SoftwareList` fragment in the inventory object of the device	Update managed object
4.	Update operation accordingly `"status": "SUCCESSFUL"`	Update Operation

Managing Device Firmware

Device capability that enables firmware management. For details and examples, compare `device information` section in the documentation.

Note: *Firmware Management* and *Software Management* are handled separately in Cumulocity IoT and follow different concepts. A device can support one ore both capabilities.

Firmware

- The firmware is the base operating system of a device
- A device can only have 1 firmware installed at a time
- When the firmware is changed, the device usually flashes itself
- Usually used by small / embedded devices

Software

- Software is an optional component executed on the device
- A device can have multiple software installed ata time
- Software can be installed or removed independently of other software and the firmware
- Usually used by larger / more powerfully devices

The following fragments are related to the optional device capability with a remark if they are required for the capability to work:

Fragment	Content	Required for optional module
`com_cumulocity_model_Agent`	Must be present in the inventory; Enables a device to receive operations	Yes
`c8y_SupportedOperations`	List contains element `c8y_Firmware`	Yes

Firmware tab will be visible on the device page only if `c8y_Firmware` is listed in the device's supported operations.

Example structure in device inventory:

```
"c8y_SupportedOperations": [
    "c8y_Firmware"
]

"c8y_Firmware": {
    "name": "raspberrypi-bootloader",
    "version": "1.20140107-1",
    "url": "31aab9856861b1a587e2094690c2f6e272712cb1"
}
```

Example operation sent to the device:

```
"c8y_Firmware": {
    "name": "firmware_a",
    "version": "2.0.24.3",
    "dependency": "2.0.23.0",
    "url": " ",
    "isPatch": true
}
```

When the device receives the operation having `c8y_Firmware`, the following steps are executed:

Step	Action	Documentation

Step	Action	Documentation
0.	Listen for operation created by platform with `"status" : "PENDING"`	Real-time notifications
1.	Update operation `"status" : "EXECUTING"`	Update operation
2.	Compare name and version stored in the fragment `c8y_Firmware` in the inventory object of the device with the name and version stored in the received operation fragment `c8y_Firmware`. If the name and/or version are differing download from the (device specific) firmware repository referenced in the url and install.	Device information
3.	Update the fragment `c8y_Firmware` of the inventory object of the device.	
4.	Update operation accordingly `"status": "SUCCESSFUL"`	Update operation

Device Profile

Device capability to manage device profiles. Device profiles represent a combination of a firmware version, one or multiple software packages and one or multiple configuration files which can be deployed on a device. Based on device profiles, users can deploy a specific target configuration on devices by using bulk operations. For details and examples, compare `Managing device data` section in the documentation.

The following fragments are related to the optional device capability with a remark if they are required for the capability to work:

Fragment	Content	Required for optional module
`com_cumulocity_model_Agent`	Must be present in the inventory; Enables a device to receive operations	Yes
`c8y_SupportedOperations`	List contains element `c8y_DeviceProfile`	Yes
`c8y_Profile`	List contains element `profileName`, `profileId`, and `profileExecuted`	Yes

Device profile tab will be visible on the device page only if `c8y_DeviceProfile` is listed in the device's supported operations.

Example structure in device inventory:

```
"c8y_SupportedOperations": [
    "c8y_DeviceProfile"
]
"c8y_Profile": {
    "profileName": "Device_Profile",
    "profileId": "60238",
```

```
    "profileExecuted": true
}
```

Example operation sent to the device:

```
"c8y_DeviceProfile": {
    "software": [
        {
            "name": "asd",
            "action": "install",
            "version": "1.0",
            "url": "aURL"
        }
    ],
    "configuration": [],
    "firmware": {
        "name": "aName",
        "version": "aVersion",
        "url": "aURL"
    },
    "description": "Assign device profile Device_Profile to device Device_Name",
    "deviceId": "437604",
    "profileId": "60238",
    "profileName": "Device_Profile"
}
```

When the device receives operation `c8y_DeviceProfile` it will execute the following steps:

Step	Action	Documentation
0.	Listen for operation created by platform with `"status" : "PENDING"`	Real-time notifications
1.	Update operation `"status" : "EXECUTING"`	Update operation
2.	Use the information stored in the operation `c8y_DeviceProfile` regarding software, firmware and configuration to execute changes as described in respective sections.	
3.	Update the fragment `c8y_Profile` of the inventory object of the device by adding the nested fragments `"profileName”:"Device_Profile"`, `"profileId": "60238"` and `"profileExecuted": true/false`.	
3.	Update operation accordingly `"status": "SUCCESSFUL"`	Update operation

Restart

Device capability to restart the device. For details and examples, compare miscellaneous section of the documentation.

The following fragments are related to the optional device capability with a remark if they are required for the capability to work:

Fragment	Content	Required for optional module
`com_cumulocity_model_Agent`	Must be present in the inventory; Enables a device to receive operations	Yes
`c8y_SupportedOperations`	List contains element `c8y_Restart`	Yes

Example structure in device inventory:

```
"c8y_SupportedOperations": [
    "c8y_Restart"
]
```

When the device receives the operation `c8y_Restart` the following steps are executed:

Step	Action	Documentation
0.	Listen for operations created by platform with `"status" : "PENDING"`	Real-time notifications
1.	Update operation `c8y_Restart` to `"status" : "EXECUTING"`	Update operation
2.	Restart the device	
3.	Query all operations in `"status" : "EXECUTING"` to continue processing them	Real-time notifications
4.	Query operations by agent ID and `"status" : "EXECUTING"` to clean up them. This includes the update of `c8y_Restart` to `"status" : "SUCCESSFUL"`	Device Integration, Update operation
5.	Listen to new operations created in Cumulocity IoT	Real-time notifications

Measurement Request

Device capability to send an updated set of measurements on user request. This can be usefully for devices, that send measurements infrequency.

The following fragments are related to the optional device capability with a remark if they are required for the capability to work:

Fragment	Content	Required for optional module
`com_cumulocity_model_Agent`	Must be present in the inventory; Enables a device to receive operations	Yes
`c8y_SupportedOperations`	List contains element `c8y_MeasurementRequestOperation`	Yes

Example structure in device inventory:

```
"c8y_SupportedOperations": [
    "c8y_MeasurementRequestOperation"
]
```

When the device receives the operation `c8y_MeasurementRequestOperation: {"requestName": "LOG"}` the following steps are executed:

Note: For legacy reasons, the `c8y_MeasurementRequestOperation` operation contains the fragment `"requestName": "LOG"`. This is to be ignored by the device. The device vendor can decide if all or a useful subset of measurements are send as response to this operation.

Step	Action	Documentation
0.	Listen for operation created by platform with `"status" : "PENDING"`	Real-time notifications
1.	Update operation `"status" : "EXECUTING"`	Update operation
2.	Send all or a useful subset of measurements	Create measurement
3.	Update operation `"status": "SUCCESSFUL"`	Update operation

Shell

Device capability to send any command to the device. The feature is often used to send shell commands to the device and receive the output as result. For details and examples, compare miscellaneous section of the documentation.

The following fragments are related to the optional device capability with a remark if they are required for the capability to work:

Fragment	Content	Required for optional module
`com_cumulocity_model_Agent`	Must be present in the inventory; Enables a device to receive operations	Yes
`c8y_SupportedOperations`	List contains element `c8y_Command`	Yes

Example structure in device inventory:

```
"c8y_SupportedOperations": [
    "c8y_Command"
]
```

Example operation sent to the device for `c8y_Command`:

```
"c8y_Command": {
    "text": "get uboot.sn"
}
```

When the device receives the operation `c8y_Command`, the following steps are executed:

Step	Action	Documentation
0.	Listen for operation created by platform with `"status" : "PENDING"`	Real-time notifications
1.	Update operation `"status" : "EXECUTING"`	Update operation
2.	Locally execute the command and add the result to the operation in the fragment `result`	Update operation
3.	Update operation `"status": "SUCCESSFUL"`	Update operation

Example operation after it has been executed and fragment `result` has been added to `c8y_Command`:

```
"c8y_Command": {
    "text": "get uboot.sn",
    "result": "123456"
}
```

Cloud Remote Access

Document Section is still under development

Device capability to initiate a remote connection via VNC or SSH. For details and examples, compare Cloud Remote Access section of the documentation.

Note: The connection type `Telnet` is considered insecure, therefore it is deprecated and should not be used for new implementations. Please use VNC or SSH instead.

The following fragments are related to the optional device capability with a remark if they are required for the capability to work:

Fragment	Content	Required for optional module

Fragment	Content	Required for optional module
`com_cumulocity_model_Agent`	Must be present in the inventory; Enables a device to receive operations	Yes
`c8y_SupportedOperations`	List contains element `c8y_RemoteAccessConnect`	Yes
`c8y_RemoteAccessList`	List of supported remote access types	Yes (at least 1 type)

Example structure in device inventory:

```
"c8y_SupportedOperations": [
    "c8y_RemoteAccessConnect"
]
"c8y_RemoteAccessList": [
    {
        "Id": 3234,
        "name": "My Connection",
        "hostname": "10.0.0.67",
        "port": 5900,
        "protocol": "VNC",
        "credentials": {
            "username": "someUser",
            "password":  "{cipher
        }SwRUVOR0gKHmPUuKLulIIM3EMGvxFTg22had6b",
    "privateKey": null,
        "publicKey": null,
        "hostKey": null,
        "type": "PASS_ONLY"
    }
}
]
```

Example operation sent to the device for `c8y_RemoteAccessConnect`:

```
"c8y_RemoteAccessConnect": {
    "hostname": "10.0.0.67", // Endpoint on local network to connect to
    "port": 5900, // Port to be used on local network endpoint
    "connectionKey": "eb5e9d13-1caa-486b-bdda-130ca0d87df8" // Shared secret to
authenticate the connection request from device side
}
```

When the device receives the operation `c8y_RemoteAccessConnect`, the following steps are executed:

Step	Action	Documentation

Step	Action	Documentation
0.	Listen for operation created by platform with `"status" : "PENDING"`	Real-time notifications
1.	Update operation `"status" : "EXECUTING"`	Update operation
2.	Connect to provided `hostname` and `port` using TCP	
3.	Using the `connectionKey` connect to device websocket endpoint of the Remote Access microservice: `wss:///service/remoteaccess/device/`	
4.	Update operation `"status": "SUCCESSFUL"`	Update operation
5.	Start forwarding binary packets between the TCP connection and the websocket in both directions	
6.	Whenever one of these connections is terminated the device considers the session as ended and will also terminate the second connection	

Location & Tracking

Device capability to display and update location information. For details and examples, compare location capabilities section of the documentation.

The following fragments are related to the optional device capability with a remark if they are required for the capability to work:

Fragment	Content	Required for optional module
`c8y_Position`	Position information of the device	Yes
`c8y_Position.lat`	Latitude	Yes
`c8y_Position.lng`	Longitude	Yes
`c8y_Position.alt`	Altitude in meters	No
`c8y_Position.trackingProtocol`	Technology used for position acquisition (e.g. GPS, Galileo, TELIC)	No
`c8y_Position.reportReason`	Reason why the position update was send (e.g. triggered by schedule, action)	No

Example structure in device inventory:

```
"c8y_Position": {
    "lat": 51.211977,
    "lng": 6.15173,
```

```
    "alt": 67,
}
```

Whenever the location shell be updated, the device executes the following steps:

Step	Action	Documentation
1.	Send a position update event	Create event
2.	Update the c8y_Position fragment in device inventory	Update managed object

Example location update event:

```
{
  "source": {
    "id": 4036, // device ID for which you want to update the position
  },
  "type": "c8y_LocationUpdate", // must be c8y_LocationUpdate
  "text": "Location updated", // field mandatory but content can be changed
  "time": "2021-09-07T14:04:27.845Z'",
  "c8y_Position": {
    "lat": 51.211977,
    "lng": 6.15173,
    "alt": 67,
  },
}
```

##MD file change Log

Date	Chapter	Severity
30/09/2021	Added MD file change log	minor
22/10/2021	Added cypher suites information	minor
01/11/2021	shell: Example added; measurements section: Naming convention added; sending operational data: table added with mandatory information; Device Information: com_cumulocity_model_agent mandatory rule changed and externalIds added	medium
03/11/2021	`com_cumulocity_model_agent` added as mandatory for each optional agent module that relies on receiving operations; Moved supported child device types to optional modules;	major
08/11/2021	Updated broken links	minor
09/11/2021	Updated broken links, added Currently Supported Device Capabilities of Self-Service Certification Microservice	minor