WolkGatewayModule-SDK-Python

Release v1.0.0

Nov 24, 2020

Contents

This package is meant to be used for developing Wolk gateway modules that enable devices without IP connectivity to send their data to WolkAbout IoT Platform.

The user is responsible for providing the custom implementation that usually contains the device's network communication protocol, as well as for providing the business logic and everything related to the used hardware and the specifics of their particular use case.

However, all the communication that is directed towards the gateway through WolkConnect - BUS Handler is already provided with this package, an open source implementation written in Python 3.7 that uses the MQTT protocol over TCP/IP to communicate with WolkGateway.

Contents:

CHAPTER 1

Creating devices

Creating devices consists of defining individual templates for each device's sensor, alarm, actuator and configuration option. All these templates are joined together to create a device template that will contain information about all data the device will yield and will be used to register the device on WolkAbout IoT Platform.

All of these templates have a field named *reference* that is used to identify that particular source of information. This field needs to be **unique per device**.

On this page, each of these templates will be disambiguated and finally a device will be created.

The concept behind device templates is similar to classes in object-oriented programming, so it helps to think about them like this:

> class -> object == device_template -> device

Where at the end of the process of creating a device, there is an uniquely identifiable object that is then registered on WolkAbout IoT Platform

1.1 Sensors

class wolk_gateway_module.model.sensor_template.**SensorTemplate**(*name: str*, *reference: str*, *data_type: wolk_gateway_module.model.data_type.* = *None*, *reading_type_name: Union[wolk_gateway_module.model.rea* *str]* = *None*, *unit: Union[wolk_gateway_module.model.rea* *str]* = *None*, *description: Optional[str]* = *None*)

Sensor template for registering device on Platform.

Variables

- **description** (*str or None*) – Description detailing this sensor
- **name** (*str*) – Name of sensor
- **reference** (*str*) – Unique sensor reference
- **unit** (ReadingType) – Sensor reading type measurement name and unit

__init__(*name: str*, *reference: str*, *data_type: wolk_gateway_module.model.data_type.DataType = None*, *reading_type_name: Union[wolk_gateway_module.model.reading_type_name.ReadingTypeName, str] = None*, *unit: Union[wolk_gateway_module.model.reading_type_measurement_unit.ReadingTypeMeasurement str] = None*, *description: Optional[str] = None*)

Sensor template for device registration request.

Define a reading type for sensors, either a generic type by specifying a DataType (numeric, boolean or string) or entering a predefined one by using the enumerations provided in ReadingTypeName and ReadingTypeMeasurementUnit.

Custom reading types that have been previously defined on WolkAbout IoT Platform can be used by passing string values for reading_type_name and unit.

Parameters

- **name** (*str*) – Sensor name
- **reference** (*str*) – Sensor reference
- **data_type** (*Optional[DataType]*) – Sensor data type for generic reading type
- **reading_type_name** (*Optional[Union[ReadingTypeName, str]]*) – Reading type name from defined enumeration or string for custom
- **unit** (*Optional[Union[ReadingTypeMeasurementUnit, str]]*) – Reading type measurement unit from defined enumeration or string for custom
- **description** (*Optional[str]*) – Description detailing the sensor's specification

to_dto() → Dict[str, Union[int, float, str, Dict[str, str]]]

Create data transfer object used for registration.

Returns dto

Return type Dict[str, Union[str, int, float]]

class wolk_gateway_module.model.data_type.**DataType**
Use to create a generic reading type.

Variables

- **BOOLEAN** (*int*) – Generic boolean reading type
- **NUMERIC** (*int*) – Generic numeric reading type
- **STRING** (*int*) – Generic string reading type

1.1.1 Reading types

class wolk_gateway_module.model.reading_type.**ReadingType**(*data_type: Optional[wolk_gateway_module.model.data_type.Da = None, name: Union[wolk_gateway_module.model.reading_type str, None] = None, unit: Union[wolk_gateway_module.model.reading_type str, None] = None*)
Reading type used for registering sensors on WolkAbout IoT Platform.

Define a reading type for sensors, either a generic type by specifying a `DataType` (numeric, boolean or string) or entering a predefined one by using the enumerations provided in `ReadingTypeName` and `ReadingTypeMeasurementUnit`.

Custom reading types can be used by passing string values for the name and measurement unit.

Variables

- **name** (*Union[ReadingTypeName, str]*) – Name of reading type
- **unit** (*Union[ReadingTypeMeasurementUnit, str]*) – Measurement unit of reading type

__init__(*data_type: Optional[wolk_gateway_module.model.data_type.DataType] = None, name: Union[wolk_gateway_module.model.reading_type_name.ReadingTypeName, str, None] = None, unit: Union[wolk_gateway_module.model.reading_type_measurement_unit.ReadingTypeMeasurementUnit, str, None] = None*)
Reading type used for registering device's sensors.

Parameters

- **data_type** (*Optional[DataType]*) – Data type for generic reading type
- **name** (*Optional[Union[ReadingTypeName, str]]*) – Reading type name from defined enumeration or string for custom
- **unit** (*Optional[Union[ReadingTypeMeasurementUnit, str]]*) – Reading type measurement unit from defined enumeration or string for custom

Raises **ValueError** – Unable to create a reading type from given input

To view all available reading type names, view source

class wolk_gateway_module.model.reading_type_name.**ReadingTypeName**
Enumeration of defined reading type names on WolkAbout IoT Platform.

To view all available reading type measurement units and their symbols, view source

class wolk_gateway_module.model.reading_type_measurement_unit.**ReadingTypeMeasurementUnit**
Enumeration of defined reading type measurement units.

1.2 Alarms

class wolk_gateway_module.model.alarm_template.**AlarmTemplate**(*name: str*, *reference: str*, *description: Optional[str] = ''*)

Alarm template for registering device on WolkAbout IoT Platform.

Variables

- **name** (*str*) – Alarm name
- **reference** (*str*) – Alarm reference
- **description** (*str*) – Alarm description

__init__(*name: str*, *reference: str*, *description: Optional[str] = ''*) → None

to_dto() → Dict[str, str]

Create data transfer object used for registration.

Returns dto

Return type Dict[str, str]

1.3 Actuators

class wolk_gateway_module.model.actuator_template.**ActuatorTemplate**(*name: str*, *reference: str*, *data_type: wolk_gateway_module.model.data_* = *None*, *reading_type_name: str* = *None*, *unit: str* = *None*, *description: str* = *None*)

Actuator template for registering device on Platform.

Variables

- **description** (*str or None*) – Description detailing this actuator
- **name** (*str*) – Name of actuator
- **reference** (*str*) – Unique actuator reference
- **unit** (*dict*) – Actuator reading type measurement name and unit

__init__(*name: str*, *reference: str*, *data_type: wolk_gateway_module.model.data_type.DataType* = *None*, *reading_type_name: str* = *None*, *unit: str* = *None*, *description: str* = *None*)

Actuator template for device registration request.

Define a reading type either by using the `data_type` to select a generic type (boolean, numeric, string) or use `reading_type_name` and `unit` to use a custom reading type that was previously defined on WolkAbout IoT Platform.

Parameters

- **name** (*str*) – Actuator name
- **reference** (*str*) – Actuator reference
- **data_type** (*Optional[DataType]*) – Actuator data type
- **reading_type_name** (*Optional[str]*) – Custom reading type name
- **unit** (*Optional[str]*) – Custom reading type measurement unit
- **description** (*Optional[str]*) – Description detailing the actuator

to_dto() → Dict[str, Union[int, float, str, Dict[str, str]]]

Create data transfer object used for registration.

Returns dto

Return type Dict[str, Union[int, float, str]]

1.4 Configurations

class wolk_gateway_module.model.configuration_template.**ConfigurationTemplate**(*name: str*, *reference: str*, *data_type: wolk_gateway_mod*, *description: Optional[str] = None*, *size: int = 1*, *labels: Optional[List[str]] = None*, *default_value: Optional[str] = None*)

Configuration template for registering device on Platform.

Variables

- **data_type** (*DataType*) – Configuration data type
- **default_value** (*str or None*) – Default value of configuration
- **description** (*str or None*) – Description of configuration
- **labels** (*List[str] or None*) – Labels of fields when data size > 1
- **name** (*str*) – Configuration name
- **reference** (*str*) – Unique configuration reference
- **size** (*int*) – Data size

__init__(*name: str*, *reference: str*, *data_type: wolk_gateway_module.model.data_type.DataType*, *description: Optional[str] = None*, *size: int = 1*, *labels: Optional[List[str]] = None*, *default_value: Optional[str] = None*)

Configuration template for device registration request.

Parameters

- **name** (*str*) – Configuration name

- **reference** (*str*) – Configuration reference
- **data_type** (DataType) – Configuration data type
- **description** (*Optional[str]*) – Configuration description
- **size** (*Optional[int]*) – Configuration data size (max 3)
- **labels** (*Optional[List[str]]*) – List of string lables when data size > 1
- **default_value** (*Optional[str]*) – Default configuration value

to_dto() → Dict[str, Union[int, str, float, List[str]]]
Create data transfer object used for registration.

Returns dto

Return type Dict[str, Union[str, int, float, List[str]]]

1.5 Device template

class wolk_gateway_module.model.device_template.**DeviceTemplate**(*actuators: List[wolk_gateway_module.model.actuat = <factory>, alarms: List[wolk_gateway_module.model.alarm = <factory>, configurations: List[wolk_gateway_module.model.config = <factory>, sensors: List[wolk_gateway_module.model.senso = <factory>, supports_firmware_update: bool = False, type_parameters: Dict[KT, VT] = <factory>, connectivity_parameters: Dict[KT, VT] = <factory>, firmware_update_parameters: Dict[KT, VT] = <factory>*)

Contains information required for registering device on Platform.

A device template consists of lists of templates (actuator, alarm, sensor, configuration) that represent what data the device is expected to send and receive. All references of a device must be unique.

Other than data feed templates, there is a `supports_firmware_update` parameter that specifies if this device has the capability to perform firmware updates.

Finally, there are type, connectivity and firmware update parameters that are dictionaries that will contain more attributes to group together devices, but are unused at this moment.

Variables

- **actuators** (*List[ActuatorTemplate]*) – List of actuators on device
- **alarms** (*List[AlarmTemplate]*) – List of alarms on device
- **configurations** (*List[ConfigurationTemplate]*) – List of configurations on device
- **connectivity_parameters** (*Dict[str, Union[str, int, float, bool]]*) – Device's connectivity parameters
- ***supports_firmware_update*** (*bool*) – Is firmware update enabled for this device
- **sensors** (*List[SensorTemplate]*) – List of sensors on device
- **type_parameters** (*Dict[str, Union[str, int, float, bool]]*) – Device's type parameters
- **firmware_update_parameters** (*Dict[str, Union[str, int, float, bool]]*) – Device's firmware update parameters

__init__(*actuators: List[wolk_gateway_module.model.actuator_template.ActuatorTemplate] = <factory>, alarms: List[wolk_gateway_module.model.alarm_template.AlarmTemplate] = <factory>, configurations: List[wolk_gateway_module.model.configuration_template.ConfigurationTemplate] = <factory>, sensors: List[wolk_gateway_module.model.sensor_template.SensorTemplate] = <factory>, supports_firmware_update: bool = False, type_parameters: Dict[KT, VT] = <factory>, connectivity_parameters: Dict[KT, VT] = <factory>, firmware_update_parameters: Dict[KT, VT] = <factory>*) → None

1.6 Device

After a device template has been created, now a device can be created from it.

class wolk_gateway_module.model.device.**Device**(*name: str, key: str, template: wolk_gateway_module.model.device_template.DeviceTemplate = <factory>*)

Device identified by name and key, as well as its template.

Variables

- **name** (*str*) – Device's name
- **key** (*str*) – Device's unique key
- **template** (DeviceTemplate) – Device template that defines data the device will send and receive.

__init__(*name: str, key: str, template: wolk_gateway_module.model.device_template.DeviceTemplate = <factory>*) → None

get_actuator_references() → List[str]

Get list of actuator references for device.

Returns actuator_references

Return type List[str]

has_configurations() → bool

Return if device has configuration options.

Returns has_configurations

Return type bool

supports_firmware_update() → bool
Return if device supports firmware update.

Returns supports_firmware_update

Return type bool

CHAPTER 2

User implemented functions

In order to enable some functionalities like actuators, configurations and firmware update, certain functions or classes must be implemented.

This page will explain the mechanisms behind each of these functionalities.

2.1 Device status provider

In order to know what devices are currently available to receive commands, the Platform and gateway need to be notified of all of the modules' device's current status. Whether they are connected, offline, in sleep mode or in service mode. Once the connection to the gateway is terminated, the module will automatically publish offline states for all devices that have been added to it.

Note: This function is **required** in order to create a `Wolk` object.

`device_status_provider.`**`get_device_status`**`()` → wolk_gateway_module.model.device_status.DeviceStatus
Get current device status.

Parameters **`device_key`** (*str*) – Device identifier

Returns status

Return type *DeviceStatus*

Available device states:

`class` `wolk_gateway_module.model.device_status.`**`DeviceStatus`**
Enumeration of available device statuses.

Variables

- **`CONNECTED`** (*str*) – Device currently connected
- **`OFFLINE`** (*str*) – Device currently offline
- **`SERVICE_MODE`** (*str*) – Device currently in service mode
- **`SLEEP`** (*str*) – Device currently in sleep mode

An stub implementation would look something like this:

```
def get_device_status(device_key):
    if device_key == "DEVICE_KEY":
        # Handle getting current device status here
        return wolk_gateway_module.DeviceStatus.CONNECTED  # OFFLINE, SLEEP, SERVICE_
↪MODE
```

2.2 Actuator functions

Actuators enable remote control over a device peripheral that can change between predefined states, like turning a switch on or off, or setting a light dimmer to 20% intensity.

In order to enable remote control, the Platform first needs to be notified about the actuators current state - is it ready to receive a command, is it busy changing its position or perhaps something has gone wrong with the actuator and it is unable to perform at that moment. This information about the actuator's current state and value is obtained through an *actuator status provider* function.

actuator_status_provider.**get_actuator_status**(*reference: str*) → Tuple[wolk_gateway_module.model.actuator_state.ActuatorState, Union[bool, int, float, str]]

Get current actuator status identified by device key and reference.

Reads the status of actuator from the device and returns it as a tuple containing the actuator state and current value.

Must be implemented as non blocking. Must be implemented as thread safe.

Parameters

- **device_key** (*str*) – Device key to which the actuator belongs to
- **reference** (*str*) – Actuator reference

Returns (state, value)

Return type (*ActuatorState*, bool or int or float or str)

Available actuator states:

class wolk_gateway_module.model.actuator_state.**ActuatorState**

Enumeration of available actuator states.

Variables

- **BUSY** (*str*) – Actuator currently in busy state
- **ERROR** (*str*) – Actuator currently in error state
- **READY** (*str*) – Actuator currently in ready state

A stub implementation would look something like this:

```
def get_actuator_status(device_key, reference):
    if device_key == "DEVICE_KEY":
        if reference == "SW":
            # Handle getting current actuator value here
            return wolk_gateway_module.ActuatorState.READY, switch.value  # BUSY,␣
↪ERROR
```

Now that the Platform and gateway are able to get information about the actuator's current value and state, it should also be able to send commands to the actuator. This is achieved through another function called *actuation handler*.

actuation_handler.**handle_actuation**(*reference: str, value: Union[bool, int, float, str]*) → None
Set device actuator identified by reference to value.

Must be implemented as non blocking. Must be implemented as thread safe.

Parameters

- **device_key** (*str*) – Device identifier
- **reference** (*str*) – Reference of the actuator
- **value** (*str*) – Value to which to set the actuator

A stub implementation would look something like this:

```
def handle_actuation(device_key, reference, value):
    if device_key == "DEVICE_KEY":
        if reference == "SW":
            # Handle setting the actuator value here
            switch.value = value
```

Finally, these two functions are passed as arguments to the `Wolk` class as `actuation_handler` and `actuator_status_provider`

```
wolk_module = wolk.Wolk(
    host=configuration["host"],
    port=configuration["port"],
    module_name=configuration["module_name"],
    device_status_provider=get_device_status,
    actuation_handler=handle_actuation,
    actuator_status_provider=get_actuator_status,
)
```

Once `Wolk.connect()` has been called, it will call `actuator_status_provider` to get the current actuator status for each actuator of all added device. However, publishing actuator statuses can be done explicitly by calling:

```
wolk_module.publish_actuator_status("DEVICE_KEY" ,"ACTUATOR_REFERENCE")
```

2.3 Configuration option functions

Configuration options enable modification of device properties from WolkAbout IoT Platform with the goal to change device behavior, eg. measurement heartbeat, enabling/disabling device interfaces, increase/decrease device logging level etc.

Configuration options require a similar way of handling messages as actuators. When a configuration command is issued from WolkAbout IoT Platform, it will be passed to a `configuration_handler` that will attempt to execute the command. Then the `configuration_provider` will report back to WolkAbout IoT Platform with the current values of the device's configuration options.

Configuration options are always sent as a whole, even when only one value changes. They are sent as a dictionary, where the key represents the configuration's reference and the value is the current value.

configuration_provider.**get_configuration**() → Dict[str, Union[int, float, bool, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str]]]

Get current configuration options.

Reads device configuration and returns it as a dictionary with device configuration reference as key, and device configuration value as value. Must be implemented as non blocking. Must be implemented as thread safe.

Parameters **device_key** (*str*) – Device identifier

Returns configuration

Return type dict

Stub implementation:

```
def get_configuration(device_key):
    if device_key == "DEVICE_KEY":
        # Handle getting configuration values here
        return {
            "configuration_1": configuration_1.value,
            "configuration_2": configuration_2.value,
        }
```

After implementing how to get current configuration option values, another function for setting new values is required

configuration_handler.**handle_configuration**(*configuration: Dict[str, Union[int, float, bool, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str]]]*) → None

Change device's configuration options.

Must be implemented as non blocking. Must be implemented as thread safe.

Parameters

- **device_key** (*str*) – Device identifier
- **configuration** (*dict*) – Configuration option reference:value pairs

```
def handle_configuration(device_key,configuration):
    if device_key == "DEVICE_KEY":
        # Handle setting configuration values here
        for reference, value in configuration.items():
            if reference == "configuration_1":
                configuration_1.value = value
            elif reference == "configuration_2":
                configuration_2.value = value
```

Finally, these two functions are passed as arguments to the `Wolk` class as `configuration_handler` and `configuration_provider`

```
wolk_module = wolk.Wolk(
    host=configuration["host"],
    port=configuration["port"],
    module_name=configuration["module_name"],
    device_status_provider=get_device_status,
    configuration_handler=handle_configuration,
    configuration_provider=get_configuration,
)
```

Once `Wolk.connect()` has been called, it will call `configuration_provider` to get the current configuration options for each added device with configurations. However, publishing configurations can be done explicitly by calling:

```
wolk_module.publish_configuration("DEVICE_KEY")
```

2.4 Enabling firmware update

WolkAbout IoT Platform has the option of updating device software/firmware. In order to enable this functionality on a device, the user has to implement the `FirmwareHandler` abstract base class and pass it to `Wolk`.

class wolk_gateway_module.interface.firmware_handler.**FirmwareHandler**

Handle firmware installation and abort commands, and report version.

Once an object of this class is passed to a Wolk object, it will set callback methods *on_install_success* and *on_install_fail* used for reporting the result of the firmware update process. Use these callbacks in *install_firmware* and *abort_installation* methods.

Variables

- **on_install_fail** (*Callable[[str, FirmwareUpdateStatus], None]*) – Installation failure callback method
- **on_install_success** (*Callable[[str], None]*) – Installation successful callback method

abort_installation(*device_key: str*) → None

Attempt to abort the firmware installation process for device.

Call `self.on_install_fail(device_key, status)` to report if the installation process was able to be aborted with `status = FirmwareUpdateStatus(FirmwareUpdateState.ABORTED)`. If unable to stop the installation process, no action is required.

Parameters **device_key** (*str*) – Device for which to abort installation

get_firmware_version(*device_key: str*) → str

Return device's current firmware version.

Parameters **device_key** (*str*) – Device identifier

Returns version

Return type str

install_firmware(*device_key: str*, *firmware_file_path: str*) → None

Handle the installation of the firmware file.

Call `self.on_install_success(device_key)` to report success. Reporting success will also get new firmware version.

If installation fails, call `self.on_install_fail(device_key, status)` where:

```
status = FirmwareUpdateStatus(
    FirmwareUpdateState.ERROR,
    FirmwareUpdateErrorCode.INSTALLATION_FAILED
)
```

or use other values from `FirmwareUpdateErrorCode` if they fit better.

Parameters

- **device_key** (*str*) – Device for which the firmware command is intended
- **firmware_file_path** (*str*) – Path where the firmware file is located

The enumerations used to report current firmware update states are listed below:

Firmware update status model.

class wolk_gateway_module.model.firmware_update_status.**FirmwareUpdateErrorCode**
Enumeration of possible firmware update errors.

Variables

- **DEVICE_NOT_PRESENT** (*int*) – Unable to pass firmware install command to device
- **FILE_NOT_PRESENT** (*int*) – Firmware file was not present at specified location
- **FILE_SYSTEM_ERROR** (*int*) – File system error occurred
- **INSTALLATION_FAILED** (*int*) – Firmware installation failed
- **UNSPECIFIED_ERROR** (*int*) – Unspecified error occurred

class wolk_gateway_module.model.firmware_update_status.**FirmwareUpdateState**
Enumeration of available firmware update states.

Variables

- **ABORTED** (*str*) – Firmware installation aborted
- **COMPLETED** (*str*) – Firmware installation completed
- **ERROR** (*str*) – Firmware installation error
- **INSTALLATION** (*str*) – Firmware installation in progress

class wolk_gateway_module.model.firmware_update_status.**FirmwareUpdateStatus**(*status: wolk_gateway_modul*
er-
ror_code:
Op-
*tional[wolk_gateway_*
=
None)

Holds information about current firmware update status.

Variables

- **status** (FirmwareUpdateState) – Firmware update status
- **error_code** (*Optional[FirmwareUpdateErrorCode]*) – Description of error that occured

```
class FirmwareHandlerImplementation(wolk_gateway_module.FirmwareHandler):

    def install_firmware(self, device_key, firmware_file_path):
        if device_key == "DEVICE_KEY":
            print(
                f"Installing firmware: '{firmware_file_path}' "
                f"on device '{device_key}'"
            )
            # Handle the actual installation here

            if True:
                # If installation was successful
                self.on_install_success(device_key)
            else:
                # If installation failed
```

(continues on next page)

(continued from previous page)

```
                status = wolk_gateway_module.FirmwareUpdateStatus(
                    wolk_gateway_module.FirmwareUpdateState.ERROR,
                    wolk_gateway_module.FirmwareUpdateErrorCode.INSTALLATION_FAILED,
                )
                self.on_install_fail(device_key, status)

    def abort_installation(self, device_key):
        if device_key == "DEVICE_KEY":
            # Manage to stop firmware installation
            status = wolk_gateway_module.FirmwareUpdateStatus(
                wolk_gateway_module.FirmwareUpdateState.ABORTED
            )
            self.on_install_fail(device_key, status)

    def get_firmware_version(self, device_key):
        if device_key == "DEVICE_KEY":
            return device.firmware_version
```

An object of this class needs to be passed to `Wolk` like so:

```
wolk_module = wolk.Wolk(
    host=configuration["host"],
    port=configuration["port"],
    module_name=configuration["module_name"],
    device_status_provider=get_device_status,
    firmware_handler=FirmwareHandlerImplementation(),
)
```

When `Wolk.connect()` is called it will use `firmware_handler.get_firmware_version()` for each added device that has support for firmware update and report to WolkAbout IoT Platform.

CHAPTER 3

Wolk API

The `Wolk` class is a wrapper and an API for everything this package has to offer.

class wolk_gateway_module.wolk.**Wolk**(*host: str, port: int, module_name: str, device_status_provider: Callable[[str], wolk_gateway_module.model.device_status.DeviceStatus], actuation_handler: Optional[Callable[[str, str, Union[bool, int, float, str]], None]] = None, actuator_status_provider: Optional[Callable[[str, str], Tuple[wolk_gateway_module.model.actuator_state.ActuatorState, Union[bool, int, float, str]]]] = None, configuration_handler: Optional[Callable[[str, Dict[str, Union[int, float, bool, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str]]]], None]] = None, configuration_provider: Optional[Callable[[str], Dict[str, Union[int, float, bool, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str]]]]] = None, firmware_handler: Optional[wolk_gateway_module.interface.firmware_handler.FirmwareHandler] = None, connectivity_service: Optional[wolk_gateway_module.connectivity.connectivity_service.ConnectivityServi = None, data_protocol: Optional[wolk_gateway_module.protocol.data_protocol.DataProtocol] = None, firmware_update_protocol: Optional[wolk_gateway_module.protocol.firmware_update_protocol.FirmwareUpda = None, registration_protocol: Optional[wolk_gateway_module.protocol.registration_protocol.RegistrationProtocol = None, status_protocol: Optional[wolk_gateway_module.protocol.status_protocol.StatusProtocol] = None, outbound_message_queue: Optional[wolk_gateway_module.persistence.outbound_message_queue.OutboundMe = None*)

Core of this package, tying together all features.

Variables

- **actuation_handler** (*Optional[Callable[[str, str,str], None]]*) – Set new actuator values for your devices
- **actuator_status_provider** (*Optional[Callable[[str, str], Tuple[ActuatorState, Union[bool, int, float, str]]]]*) – Get device's current actuator state
- **configuration_handler** (*Optional[Callable[[str, Dict[str, str]], None]]*) – Set new configuration values for your devices
- **configuration_provider** (*Optional[Callable[[str],Dict[]]]*) – Get device's current configuration options
- **connectivity_service** (ConnectivityService) – Service that enables connection to WolkGateway
- **data_protocol** (DataProtocol) – Parse messages related to device data
- **device_status_provider** (*Callable[[str], DeviceStatus]*) – Get device's current status
- **devices** (*List[Device]*) – List of devices added to module
- **firmware_handler** (*Optional[FirmwareHandler]*) – Handle commands related to firmware update
- **firmware_update_protocol** (FirmwareUpdateProtocol) – Parse messages related to firmware update
- **host** (*str*) – WolkGateway's host address
- **log** (*logging.Logger*) – Logger instance
- **module_name** (*str*) – Name of module used for identification on WolkGateway
- **outbound_message_queue** (OutboundMessageQueue) – Means of storing messages
- **port** (*int*) – WolkGateway's connectivity port
- **registration_protocol** (RegistrationProtocol) – Parse messages related to device registration
- **status_protocol** (StatusProtocol) – Parse messages related to device status

__init__(*host: str, port: int, module_name: str, device_status_provider: Callable[[str], wolk_gateway_module.model.device_status.DeviceStatus], actuation_handler: Optional[Callable[[str, str, Union[bool, int, float, str]], None]] = None, actuator_status_provider: Optional[Callable[[str, str], Tuple[wolk_gateway_module.model.actuator_state.ActuatorState, Union[bool, int, float, str]]]] = None, configuration_handler: Optional[Callable[[str, Dict[str, Union[int, float, bool, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str]]]], None]] = None, configuration_provider: Optional[Callable[[str], Dict[str, Union[int, float, bool, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str]]]]] = None, firmware_handler: Optional[wolk_gateway_module.interface.firmware_handler.FirmwareHandler] = None, connectivity_service: Optional[wolk_gateway_module.connectivity.connectivity_service.ConnectivityService] = None, data_protocol: Optional[wolk_gateway_module.protocol.data_protocol.DataProtocol] = None, firmware_update_protocol: Optional[wolk_gateway_module.protocol.firmware_update_protocol.Firmware = None, registration_protocol: Optional[wolk_gateway_module.protocol.registration_protocol.RegistrationProtoco = None, status_protocol: Optional[wolk_gateway_module.protocol.status_protocol.StatusProtocol] = None, outbound_message_queue: Optional[wolk_gateway_module.persistence.outbound_message_queue.Outbou = None*)

Construct an instance ready to communicate with WolkGateway.

Parameters

- **host** (*str*) – Host address of WolkGateway
- **port** (*int*) – TCP/IP port of WolkGateway
- **module_name** (*str*) – Module identifier used when connecting to gateway
- **device_status_provider** (*Callable[[str], DeviceStatus]*) – Provider of device's current status
- **actuation_handler** (*Optional[Callable[[str, str, str], None]]*) – Setter of new device actuator values
- **actuator_status_provider** (*Optional[Callable[[str, str], Tuple[ActuatorState, Union[bool, int, float, str]]]]*) – Provider of device's current actuator status
- **configuration_handler** (*Optional[Callable[[str, Dict[str, Union[bool, int, float, str]]], None]]*) – Setter of new device configuration values
- **configuration_provider** (*Optional[Callable[[str], Dict[str, Union[int, float, bool, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str]]]]]*) – Provider of device's configuration options
- **install_firmware** (*Optional[Callable[[str, str], None]]*) – Handling of firmware installation
- **connectivity_service** (*Optional[ConnectivityService]*) – Custom connectivity service implementation
- **data_protocol** (*Optional[DataProtocol]*) – Custom data protocol implementation
- **firmware_update_protocol** (*Optional[FirmwareUpdateProtocol]*) – Custom firmware update protocol implementation

- **registration_protocol** (*Optional[RegistrationProtocol]*) – Custom registration protocol implementation
- **status_protocol** (*Optional[StatusProtocol]*) – Custom device status protocol implementation
- **outbound_message_queue** (*Optional[OutboundMessageQueue]*) – Custom persistent storage implementation

Raises

- **ValueError** – Bad values provided for arguments.
- **RuntimeError** – An argument is missing its pair.

add_alarm(*device_key: str*, *reference: str*, *active: bool*, *timestamp: Optional[int] = None*) → None

Serialize alarm event and put into storage.

Alarms without a specified timestamp will be assigned a timestamps via `int(round(time.time() * 1000))`

Parameters

- **device_key** (*str*) – Device on which the sensor reading occurred
- **reference** (*str*) – Alarm reference (unique per device)
- **value** – Current state of alarm
- **timestamp** (*Optional[int]*) – Unix time

Raises **RuntimeError** – Unable to place in storage

add_device(*device: wolk_gateway_module.model.device.Device*) → None

Add device to module.

Will attempt to send a registration request and update list of subscribed topics.

Parameters **device** (Device) – Device to be added to module

Raises

- **RuntimeError** – Unable to store message
- **ValueError** – Invalid device given

add_sensor_reading(*device_key: str, reference: str, value: Union[bool, int, float, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str]], timestamp: Optional[int] = None*) → None

Serialize sensor reading and put into storage.

Readings without a specified timestamp will be assigned a timestamps via `int(round(time.time() * 1000))`

Parameters

- **device_key** (*str*) – Device on which the sensor reading occurred
- **reference** (*str*) – Sensor reference (unique per device)
- **value** (*Union[bool, int, float, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str],]*) – Value(s) that the reading yielded
- **timestamp** (*Optional[int]*) – Unix time

Raises **RuntimeError** – Unable to place in storage

add_sensor_readings(*device_key: str, readings: Dict[str, Union[int, float, bool, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str]]], timestamp: Optional[int] = None*) → None

Serialize multiple sensor readings and put into storage.

Readings without a specified timestamp will be assigned a timestamps via `int(round(time.time() * 1000))`

Parameters

- **device_key** (*str*) – Device on which the sensor reading occurred
- **readings** (*Dict[str, Union[bool, int, float, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str],]*) – dictionary in sensor_reference:value format
- **timestamp** (*Optional[int]*) – Unix time

Raises **RuntimeError** – Unable to place in storage

connect() → None

Establish connection with WolkGateway.

Will attempt to publish actuator statuses, configuration options, and current firmware version for all added devices.

Raises **RuntimeError** – Error publishing actuator status or configuration

disconnect() → None

Terminate connection with WolkGateway.

publish(*device_key: Optional[str] = None*) → None

Publish stored messages to WolkGateway.

If device_key parameter is provided, will publish messages only for that specific device.

Parameters **device_key** (*Optional[str]*) – Device for which to publish stored messages

publish_actuator_status(*device_key: str, reference: str, state: Optional[wolk_gateway_module.model.actuator_state.ActuatorState] = None, value: Union[bool, int, float, str, None] = None*) → None

Publish device actuator status to WolkGateway.

Getting the actuator status is achieved by calling the user's implementation of `actuator_status_provider` or optionally an actuator status can be published explicitly by providing `ActuatorState` as `state` argument and the current actuator value via `value` argument

If message is unable to be sent, it will be placed in storage.

If no `actuator_status_provider` is present, will raise exception.

Parameters

- **device_key** (*str*) – Device on which the sensor reading occurred
- **reference** (*str*) – Alarm reference (unique per device)
- **state** (*Optional[ActuatorState]*) – Current actuator state for explicitly publishing status
- **value** (*Optional[Union[bool, int, float, str]]*) – Current actuator value for explicitly publishing status

Raises

- **ValueError** – Provided state is not an instance of ActuatorState
- **RuntimeError** – Unable to place in storage or no status provider

publish_configuration(*device_key: str*) → None
Publish device configuration options to WolkGateway.

If message is unable to be sent, it will be placed in storage.

Getting the current configuration is achieved by calling the user's implementation of `configuration_provider`.

If no `configuration_provider` is present, will raise exception.

Parameters **device_key** (*str*) – Device to which the configuration belongs to

Raises **RuntimeError** – No configuration provider present or no data returned

publish_device_status(*device_key: str*, *status: Optional[wolk_gateway_module.model.device_status.DeviceStatus] = None*) → None
Publish current device status to WolkGateway.

Getting the current device status is achieved by calling the user's provided `device_status_provider` or a device status can be published explicitly by passing a `DeviceStatus` as the `status` parameter.

Parameters

- **device_key** (*str*) – Device to which the status belongs to
- **status** (*Optional[DeviceStatus]*) – Current device status

Raises

- **ValueError** – status is not of `DeviceStatus`
- **RuntimeError** – Failed to publish and store message

remove_device(*device_key: str*) → None
Remove device from module.

Removes device for subscription topics and lastwill message.

Parameters **device_key** (*str*) – Device identifier

CHAPTER 4

Protocol & connectivity

The `Wolk` class is modular, in the sense that it relies on implementation of certain abstract base classes that each handle a certain aspect of the package:

- **Data protocol** - Parses messages related to device data (actuators, alarms, sensors & configurations)
- **Firmware update protocol** - Parses messages related to firmware update (sending current firmware version, handling installation commands, reporting current firmware installation status)
- **Registration protocol** - Parses messages related to device registration (sending registration requests, handling registration responses)
- **Status protocol** - Parses messages related to device status(send current device status, respond to device status requests)
- **Outbound message queue** - Store serialized messages before publishing them to the gateway/platform
- **Connectivity service** - Means of establishing connection to WolkGateway and exchanging messages

All listed items have already been implemented to work over MQTT using WolkAbout's JSON_PROTOCOL for serializing messages and they will be listed in the remainder of this page.

If you are interested in implementing a different means of storing messages, or want to use a different MQTT client, or even implement a custom message formatting protocol you can do so by implementing these *Abstract base classes*.

4.1 JSON Data Protocol

class wolk_gateway_module.json_data_protocol.**JsonDataProtocol**

Parse inbound messages and serialize outbound messages.

extract_key_from_message(*message: wolk_gateway_module.model.message.Message*) → str

Extract device key from message.

Parameters **message** (Message) – Message received

Returns device_key

Return type str

get_inbound_topics_for_device(*device_key: str*) → list
Return list of inbound topics for given device key.

Parameters **device_key** (*str*) – Device key for which to create topics

Returns inbound_topics

Return type list

is_actuator_get_message(*message: wolk_gateway_module.model.message.Message*) → bool
Check if message is actuator get command.

Parameters **message** (Message) – Message received

Returns is_actuator_get_message

Return type bool

is_actuator_set_message(*message: wolk_gateway_module.model.message.Message*) → bool
Check if message is actuator set command.

Parameters **message** (Message) – Message received

Returns is_actuator_set_message

Return type bool

is_configuration_get_message(*message: wolk_gateway_module.model.message.Message*) → bool
Check if message is configuration get command.

Parameters **message** (Message) – Message received

Returns is_configuration_get_message

Return type bool

is_configuration_set_message(*message: wolk_gateway_module.model.message.Message*) → bool
Check if message is configuration set command.

Parameters **message** (Message) – Message received

Returns is_configuration_set_message

Return type bool

make_actuator_command(*message: wolk_gateway_module.model.message.Message*) → wolk_gateway_module.model.actuator_command.ActuatorCommand
Make actuator command from message.

Parameters **message** (Message) – Message received

Returns actuator_command

Return type *ActuatorCommand*

make_actuator_status_message(*device_key: str*, *actuator_status: wolk_gateway_module.model.actuator_status.ActuatorStatus*) → wolk_gateway_module.model.message.Message
Make message from actuator status for device key.

Parameters

- **device_key** (*str*) – Device on which the actuator status occurred
- **actuator_status** (ActuatorStatus) – Actuator status data

Returns message

Return type *Message*

make_alarm_message(*device_key: str*, *alarm: wolk_gateway_module.model.alarm.Alarm*) → wolk_gateway_module.model.message.Message

Make message from alarm for device key.

Parameters

- **device_key** (*str*) – Device on which the alarm occurred
- **alarm** (Alarm) – Alarm data

Returns message

Return type *Message*

make_configuration_command(*message: wolk_gateway_module.model.message.Message*) → wolk_gateway_module.model.configuration_command.ConfigurationCommand

Make configuration command from message.

Parameters **message** (Message) – Message received

Returns configuration_command

Return type *ConfigurationCommand*

make_configuration_message(*device_key: str*, *configuration: dict*) → wolk_gateway_module.model.message.Message

Make message from configuration for device key.

Parameters

- **device_key** (*str*) – Device to which the configuration belongs to.
- **configuration** (*dict*) – Current configuration data

Returns message

Return type *Message*

make_sensor_reading_message(*device_key: str*, *sensor_reading: wolk_gateway_module.model.sensor_reading.SensorReading*) → wolk_gateway_module.model.message.Message

Make message from sensor reading for device key.

Parameters

- **device_key** (*str*) – Device on which the sensor reading occurred
- **sensor_reading** (SensorReading) – Sensor reading data

Returns message

Return type *Message*

make_sensor_readings_message(*device_key: str*, *sensor_readings: List[wolk_gateway_module.model.sensor_reading.SensorReading]*, *timestamp: int = None*) → wolk_gateway_module.model.message.Message

Make message from multiple sensor readings for device key.

Parameters

- **device_key** (*str*) – Device on which the sensor reading occurred
- **sensor_reading** (*List[SensorReading]*) – Sensor readings data
- **timestamp** (*Optional[int]*) – Timestamp

Returns message

Return type *Message*

4.2 JSON Firmware Update Protocol

class wolk_gateway_module.json_firmware_update_protocol.**JsonFirmwareUpdateProtocol**

Parse inbound messages and serialize outbound firmware messages.

extract_key_from_message(*message: wolk_gateway_module.model.message.Message*) → str

Return device key from message.

Parameters **message** (Message) – Message received

Returns device_key

Return type str

get_inbound_topics_for_device(*device_key: str*) → list

Return list of inbound topics for given device key.

Parameters **device_key** (*str*) – Device key for which to create topics

Returns inbound_topics

Return type list

is_firmware_abort_command(*message: wolk_gateway_module.model.message.Message*) → bool

Check if received message is firmware abort command.

Parameters **message** (Message) – Message received

Returns is_firmware_abort_command

Return type bool

is_firmware_install_command(*message: wolk_gateway_module.model.message.Message*) → bool

Check if received message is firmware install command.

Parameters **message** (Message) – Message received

Returns is_firmware_install_command

Return type bool

make_firmware_file_path(*message: wolk_gateway_module.model.message.Message*) → str

Extract file path from firmware install message.

Parameters **message** (Message) – Message received

Returns firmware_file_path

Return type str

make_update_message(*device_key: str*, *status: wolk_gateway_module.model.firmware_update_status.FirmwareUpdateStat*
→ wolk_gateway_module.model.message.Message

Make message from device firmware update status.

Parameters

- **device_key** (*str*) – Device key to which the firmware update status belongs to
- **status** (FirmwareUpdateStatus) – Device firmware update status

Returns message

Return type *Message*

make_version_message(*device_key: str*, *firmware_verison: str*) → wolk_gateway_module.model.message.Message

Make message from device firmware update version.

Parameters

- **device_key** (*str*) – Device key to which the firmware update belongs to
- **firmware_verison** (*str*) – Current firmware version

Returns message

Return type *Message*

4.3 JSON Registration Protocol

class wolk_gateway_module.json_registration_protocol.**JsonRegistrationProtocol**

Send device registration requests and handle their responses.

extract_key_from_message(*message: wolk_gateway_module.model.message.Message*) → str

Return device key from message.

Parameters **message** (Message) – Message received

Returns device_key

Return type str

get_inbound_topics_for_device(*device_key: str*) → list

Return list of inbound topics for given device key.

Parameters **device_key** (*str*) – Device key for which to create topics

Returns inbound_topics

Return type list

is_registration_response_message(*message: wolk_gateway_module.model.message.Message*) → bool

Check if message is device registration response.

Parameters **message** (Message) – Message received

Returns is_device_registration_response

Return type bool

make_registration_message(*request: wolk_gateway_module.model.device_registration_request.DeviceRegistrationReq* → wolk_gateway_module.model.message.Message

Make message from device registration request.

Parameters **request** (DeviceRegistrationRequest) – Device registration request

Returns message

Return type *Message*

make_registration_response(*message: wolk_gateway_module.model.message.Message*) → wolk_gateway_module.model.device_registration_response.DeviceRegistrationResponse

Make device registration response from message.

Parameters **message** – Message received

Rtype message Message

Returns device_registration_response

Return type *DeviceRegistrationResponse*

4.4 JSON Status Protocol

class wolk_gateway_module.json_status_protocol.**JsonStatusProtocol**

Parse inbound messages and serialize device status messages.

extract_key_from_message(*message: wolk_gateway_module.model.message.Message*) → str

Extract device key from message.

Parameters **message** (Message) – Message received

Returns device_key

Return type str

get_inbound_topics_for_device(*device_key: str*) → list

Return list of inbound topics for given device key.

Parameters **device_key** (*str*) – Device key for which to create topics

Returns inbound_topics

Return type list

is_device_status_request_message(*message: wolk_gateway_module.model.message.Message*) → bool

Check if message is device status request.

Parameters **message** (Message) – Message received

Returns is_device_status_request

Return type bool

make_device_status_response_message(*device_status: wolk_gateway_module.model.device_status.DeviceStatus*, *device_key: str*) → wolk_gateway_module.model.message.Message

Make message from device status response.

Parameters

- **device_key** (*str*) – Device to which the status belongs to
- **device_status** (DeviceStatus) – Device's current status

Returns message

Return type *Message*

make_device_status_update_message(*device_status: wolk_gateway_module.model.device_status.DeviceStatus*, *device_key: str*) → wolk_gateway_module.model.message.Message

Make message from device status update.

Parameters

- **device_key** (*str*) – Device to which the status belongs to
- **device_status** (DeviceStatus) – Device's current status

Returns message

Return type *Message*

make_last_will_message(*device_keys: list*) → wolk_gateway_module.model.message.Message
Make last will message from list of device keys.

Parameters **device_keys** (*list(str)*) – List of device keys

Returns message

Return type *Message*

4.5 Outbound Message Deque

class wolk_gateway_module.outbound_message_deque.**OutboundMessageDeque**
Responsible for storing messages before being sent to WolkGateway.

Messages are sent in the order they were added to the queue.

Storing readings and alarms without Unix timestamp will result in all sent messages being treated as live readings and will be assigned a timestamp upon reception, so for a valid history add timestamps to readings via *int(round(time.time() * 1000))*

get() → Optional[wolk_gateway_module.model.message.Message]
Get the first message from storage without removing it.

Returns message

Return type *Message*, None

get_messages_for_device(*device_key: str*) → List[wolk_gateway_module.model.message.Message]
Return a list of messages that belong to a certain device.

Does not remove from storage.

Parameters **device_key** (*str*) – Device identifier

Returns messages

Return type List[*Message*]

put(*message: wolk_gateway_module.model.message.Message*) → bool
Place a message in storage.

Parameters **message** (Message) – Message to be stored

Returns result

Return type bool

queue_size() → int
Return current number of messages in storage.

Returns size

Return type int

remove(*message: wolk_gateway_module.model.message.Message*) → bool
Remove specific message from storage.

Returns result

Return type bool

4.6 MQTT Connectivity Service

class wolk_gateway_module.mqtt_connectivity_service.**MQTTConnectivityService**(*host: str*, *port: int*, *client_id: str*, *qos: int*, *last-will_message: wolk_gateway_modul top-ics: list*)

Responsible for exchanging data with WolkGateway through MQTT.

add_subscription_topics(*topics: List[str]*) → None
Add subscription topics.

Adding these topics will not subscribe to them immediately, a new connection needs to happen to subscribe to them.

Parameters **topics** (*List[str]*) – List of topics

connect() → bool
Establish connection with WolkGateway.

Returns result

Return type bool

connected() → bool
Return if currently connected.

Returns connected

Return type bool

disconnect() → None
Terminate connection with WolkGateway.

publish(*message: wolk_gateway_module.model.message.Message*) → bool
Publish serialized data to WolkGateway.

Parameters **message** (Message) – Message to be published

Returns result

Return type bool

reconnect() → bool
Terminate existing and create new connection with WolkGateway.

Returns result

Return type bool

Raises **RuntimeError** – Reason for connection being refused

remove_topics_for_device(*device_key: str*) → None
Remove topics for device from subscription topics.

Parameters **device_key** (*str*) – Device identifier

set_inbound_message_listener(*on_inbound_message: Callable[[wolk_gateway_module.model.message.Message], None]*) → None
Set the callback function ot handle inbound messages.

Parameters **on_inbound_message** (*Callable[[Message], None]*) – Callable that handles inbound messages

set_lastwill_message(*message: wolk_gateway_module.model.message.Message*) → None
Send offline state for module devices on disconnect.

CHAPTER 5

Abstract base classes

5.1 Data Protocol

class wolk_gateway_module.protocol.data_protocol.**DataProtocol**

Parse inbound messages and serialize outbound messages.

extract_key_from_message(*message: wolk_gateway_module.model.message.Message*) → str

Extract device key from message.

Parameters **message** (Message) – Message received

Returns device_key

Return type str

get_inbound_topics_for_device(*device_key: str*) → List[str]

Return list of inbound topics for given device key.

Parameters **device_key** (*str*) – Device key for which to create topics

Returns inbound_topics

Return type list

is_actuator_get_message(*message: wolk_gateway_module.model.message.Message*) → bool

Check if message is actuator get command.

Parameters **message** (Message) – Message received

Returns is_actuator_get_message

Return type bool

is_actuator_set_message(*message: wolk_gateway_module.model.message.Message*) → bool

Check if message is actuator set command.

Parameters **message** (Message) – Message received

Returns is_actuator_set_message

Return type bool

is_configuration_get_message(*message: wolk_gateway_module.model.message.Message*) → bool

Check if message is configuration get command.

Parameters **message** (Message) – Message received

Returns is_configuration_get_message

Return type bool

is_configuration_set_message(*message: wolk_gateway_module.model.message.Message*) → bool

Check if message is configuration set command.

Parameters **message** (Message) – Message received

Returns is_configuration_set_message

Return type bool

make_actuator_command(*message: wolk_gateway_module.model.message.Message*) → wolk_gateway_module.model.actuator_command.ActuatorCommand

Make actuator command from message.

Parameters **message** (Message) – Message received

Returns actuator_command

Return type *ActuatorCommand*

make_actuator_status_message(*device_key: str*, *actuator_status: wolk_gateway_module.model.actuator_status.ActuatorStatus*) → wolk_gateway_module.model.message.Message

Make message from actuator status for device key.

Parameters

- **device_key** (*str*) – Device on which the actuator status occurred
- **actuator_status** (ActuatorStatus) – Actuator status data

Returns message

Return type *Message*

make_alarm_message(*device_key: str*, *alarm: wolk_gateway_module.model.alarm.Alarm*) → wolk_gateway_module.model.message.Message

Make message from alarm for device key.

Parameters

- **device_key** (*str*) – Device on which the alarm occurred
- **alarm** (Alarm) – Alarm data

Returns message

Return type *Message*

make_configuration_command(*message: wolk_gateway_module.model.message.Message*) → wolk_gateway_module.model.configuration_command.ConfigurationCommand

Make configuration command from message.

Parameters **message** (Message) – Message received

Returns configuration_command

Return type *ConfigurationCommand*

make_configuration_message(*device_key: str, configuration: Dict[str, Union[int, float, bool, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str]]]*) → wolk_gateway_module.model.message.Message

Make message from configuration for device key.

Parameters

- **device_key** (*str*) – Device to which the configuration belongs to.
- **configuration** (*dict*) – Current configuration data

Returns message

Return type *Message*

make_sensor_reading_message(*device_key: str, sensor_reading: wolk_gateway_module.model.sensor_reading.SensorReading*) → wolk_gateway_module.model.message.Message

Make message from sensor reading for device key.

Parameters

- **device_key** (*str*) – Device on which the sensor reading occurred
- **sensor_reading** (SensorReading) – Sensor reading data

Returns message

Return type *Message*

make_sensor_readings_message(*device_key: str, sensor_readings: List[wolk_gateway_module.model.sensor_reading.SensorReading], timestamp: Optional[int] = None*) → wolk_gateway_module.model.message.Message

Make message from multiple sensor readings for device key.

Parameters

- **device_key** (*str*) – Device on which the sensor reading occurred
- **sensor_readings** (*List[SensorReading]*) – List of sensor readings data
- **timestamp** (*Optional[int]*) – Timestamp

Returns message

Return type *Message*

5.2 Firmware Update Protocol

class wolk_gateway_module.protocol.firmware_update_protocol.**FirmwareUpdateProtocol**

Parse inbound messages and serialize outbound firmware messages.

extract_key_from_message(*message: wolk_gateway_module.model.message.Message*) → str

Return device key from message.

Parameters **message** (Message) – Message received

Returns device_key

Return type str

get_inbound_topics_for_device(*device_key: str*) → List[str]
Return list of inbound topics for given device key.

Parameters **device_key** (*str*) – Device key for which to create topics

Returns inbound_topics

Return type list

is_firmware_abort_command(*message: wolk_gateway_module.model.message.Message*) → bool
Check if received message is firmware abort command.

Parameters **message** (Message) – Message received

Returns is_firmware_abort_command

Return type bool

is_firmware_install_command(*message: wolk_gateway_module.model.message.Message*) → bool
Check if received message is firmware install command.

Parameters **message** (Message) – Message received

Returns is_firmware_install_command

Return type bool

make_firmware_file_path(*message: wolk_gateway_module.model.message.Message*) → str
Extract file path from firmware install message.

Parameters **message** (Message) – Message received

Returns firmware_file_path

Return type str

make_update_message(*device_key: str*, *status: wolk_gateway_module.model.firmware_update_status.FirmwareUpdateStat*
→ wolk_gateway_module.model.message.Message
Make message from device firmware update status.

Parameters

- **device_key** (*str*) – Device key to which the firmware update belongs to
- **status** (FirmwareUpdateStatus) – Device firmware update status

Returns message

Return type *Message*

make_version_message(*device_key: str*, *firmware_verison: str*) → wolk_gateway_module.model.message.Message
Make message from device firmware update version.

Parameters

- **device_key** (*str*) – Device key to which the firmware update belongs to
- **firmware_verison** (*str*) – Current firmware version

Returns message

Return type *Message*

5.3 Registration Protocol

class wolk_gateway_module.protocol.registration_protocol.**RegistrationProtocol**

Parse inbound messages and serialize outbound registration messages.

extract_key_from_message(*message: wolk_gateway_module.model.message.Message*) → str

Return device key from message.

Parameters **message** (Message) – Message received

Returns device_key

Return type str

get_inbound_topics_for_device(*device_key: str*) → List[str]

Return list of inbound topics for given device key.

Parameters **device_key** (*str*) – Device key for which to create topics

Returns inbound_topics

Return type list

is_registration_response_message(*message: wolk_gateway_module.model.message.Message*) → bool

Check if message is device registration response.

Parameters **message** (Message) – Message received

Returns is_device_registration_response

Return type bool

make_registration_message(*request: wolk_gateway_module.model.device_registration_request.DeviceRegistrationReq* → wolk_gateway_module.model.message.Message

Make message from device registration request.

Parameters **request** (DeviceRegistrationRequest) – Device registration request

Returns message

Return type *Message*

make_registration_response(*message: wolk_gateway_module.model.message.Message*) → wolk_gateway_module.model.device_registration_response.DeviceRegistrationResponse

Make device registration response from message.

Parameters **message** – Message received

Rtpe message Message

Returns device_registration_response

Return type *DeviceRegistrationResponse*

5.4 Status Protocol

class wolk_gateway_module.protocol.status_protocol.**StatusProtocol**

Parse inbound messages and serialize device status messages.

extract_key_from_message(*message: wolk_gateway_module.model.message.Message*) → str

Extract device key from message.

Parameters **message** (Message) – Message received

Returns device_key

Return type str

get_inbound_topics_for_device(*device_key: str*) → List[str]
Return list of inbound topics for given device key.

Parameters **device_key** (*str*) – Device key for which to create topics

Returns inbound_topics

Return type list

is_device_status_request_message(*message: wolk_gateway_module.model.message.Message*) → bool
Check if message is device status request.

Parameters **message** (Message) – Message received

Returns is_device_status_request

Return type bool

make_device_status_response_message(*device_status: wolk_gateway_module.model.device_status.DeviceStatus*, *device_key: str*) → wolk_gateway_module.model.message.Message
Make message from device status response.

Parameters

- **device_status** (DeviceStatus) – Device's current status
- **device_key** (*str*) – Device to which the status belongs to

Returns message

Return type *Message*

make_device_status_update_message(*device_status: wolk_gateway_module.model.device_status.DeviceStatus*, *device_key: str*) → wolk_gateway_module.model.message.Message
Make message from device status update.

Parameters

- **device_status** (DeviceStatus) – Device's current status
- **device_key** (*str*) – Device to which the status belongs to

Returns message

Return type *Message*

make_last_will_message(*device_keys: List[str]*) → wolk_gateway_module.model.message.Message
Make last will message from list of device keys.

Parameters **device_keys** (*list(str)*) – List of device keys

Returns message

Return type *Message*

5.5 Outbound Message Queue

class wolk_gateway_module.persistence.outbound_message_queue.**OutboundMessageQueue**
Responsible for storing messages before being sent to WolkGateway.

get() → Optional[wolk_gateway_module.model.message.Message]
Get the first message from storage without removing it.

Returns message

Return type *Message*, None

get_messages_for_device(*device_key: str*) → List[wolk_gateway_module.model.message.Message]
Return a list of messages that belong to a certain device.

Does not remove from storage.

Parameters **device_key** (*str*) – Device identifier

Returns messages

Return type List[*Message*]

put(*message: wolk_gateway_module.model.message.Message*) → bool
Place a message in storage.

Parameters **message** (Message) – Message to be stored

Returns result

Return type bool

queue_size() → int
Return current number of messages in storage.

Returns size

Return type int

remove(*message: wolk_gateway_module.model.message.Message*) → bool
Remove specific message from storage.

Returns result

Return type bool

5.6 Connectivity Service

class wolk_gateway_module.connectivity.connectivity_service.**ConnectivityService**
Responsible for exchanging data with WolkGateway.

add_subscription_topics(*topics: List[str]*) → None
Add subscription topics.

Parameters **topics** (*List[str]*) – List of topics

connect() → Optional[bool]
Establish connection with WolkGateway.

connected() → bool
Return if currently connected.

Returns connected

Return type bool

disconnect() → Optional[bool]
Terminate connection with WolkGateway.

publish (*message: wolk_gateway_module.model.message.Message*) → bool
Publish serialized data to WolkGateway.

Parameters **message** (Message) – Message to be published

Returns result

Return type bool

reconnect () → Optional[bool]
Reestablish connection with WolkGateway.

remove_topics_for_device (*device_key: str*) → None
Remove topics for device from subscription topics.

Parameters **device_key** (*str*) – Device identifier

set_inbound_message_listener (*on_inbound_message: Callable[[wolk_gateway_module.model.message.Message], None]*) → None
Set the callback function to handle inbound messages.

Parameters **on_inbound_message** (*Callable[[Message], None]*) – Callable that handles inbound messages

set_lastwill_message (*message: wolk_gateway_module.model.message.Message*) → None
Send offline state for module devices on disconnect.

Parameters **message** (Message) – Message to be published

CHAPTER 6

Models

This page lists data models used through out the package

Actuator command received from WolkAbout IoT Platform.

class wolk_gateway_module.model.actuator_command.**ActuatorCommand**(*reference: str*, *command: wolk_gateway_module.model.actuator*, *value: Union[bool, int, float, str, None] = None*)

Actuator command for reference with command and optionally value.

Variables

- **reference** (*str*) – What actuator is the command for
- **command** (ActuatorCommandType) – Type of command received
- **value** (*Optional[Union[bool, int, float, str]]*) – Value to be set

class wolk_gateway_module.model.actuator_command.**ActuatorCommandType**

Actuator command type.

Variables

- **GET** (*int*) – Get current actuator value
- **SET** (*int*) – Set actuator to value

Actuator status model.

class wolk_gateway_module.model.actuator_status.**ActuatorStatus**(*reference: str, state: wolk_gateway_module.model.actuator_s value: Union[bool, int, float, str, None] = None*)

Holds information of a devices actuator current status.

Variables

- **reference** (*str*) – Device actuator's reference as defined in device template
- **state** (ActuatorState) – Actuator's current state
- **value** (*Optional[Union[bool, int, float, str]]*) – Current value of actuator, None only for error state

Alarm event model.

class wolk_gateway_module.model.alarm.**Alarm**(*reference: str, active: bool, timestamp: Optional[int]*)

Holds information about a devices alarm.

Variables

- **reference** (*str*) – Device alarm's reference as defined in device template
- **active** (*bool*) – Alarm's current state
- **timestamp** (*int or None*) – Unix timestamp in miliseconds. If not provided, Platform will assign timestamp when it receives it.

Configuration command received from WolkAbout IoT Platform.

class wolk_gateway_module.model.configuration_command.**ConfigurationCommand**(*command: wolk_gateway_module value: Optional[Dict[str, Union[int, float, bool, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str]]]] = None*)

Configuration command with command and optionally value.

Variables

- **command** (*int*) – Configuration command received
- **value** (*Optional[dict]*) – Set configuration to value

class wolk_gateway_module.model.configuration_command.**ConfigurationCommandType**

Configuration command type.

Variables

- **GET** (*int*) – Get current configuration options
- **SET** (*int*) – Set configuration to value

Device registration request model.

class wolk_gateway_module.model.device_registration_request.**DeviceRegistrationRequest**(*name: str*, *key: str*, *template: wolk_g = <factory>*, *default_b bool = True*)

Request for device registration.

Variables

- **name** (*str*) – Device name
- **key** (*str*) – Unique device key
- **template** (DeviceTemplate) – Device template
- **default_binding** (*bool*) – Create semantic group for device on Platform

Response for device registration request.

class wolk_gateway_module.model.device_registration_response.**DeviceRegistrationResponse**(*key str*, *result wol de- scri tion str = ''*)

Response for device registration request.

Identified by device key and result, with an optional description of the error that occurred.

Variables

- **key** (*str*) – Unique device key
- **result** (DeviceRegistrationResponseResult) – Result of the registration process
- **description** (*str*) – Description of error that occurred

Device registration response results.

class wolk_gateway_module.model.device_registration_response_result.**DeviceRegistrationRespo**

Enumeration of possible registration response results.

Variables

- **OK** (*str*) – Device was successfully registered

- **ERROR_GATEWAY_NOT_FOUND** (*str*) – Gateway that sent the registration request was not found
- **ERROR_NOT_A_GATEWAY** (*str*) – Sender of request is not a gateway
- **ERROR_KEY_CONFLICT** (*str*) – Device with that key already exists
- **ERROR_MAXIMUM_NUMBER_OF_DEVICES_EXCEEDED** (*str*) – Reached limit for number of devices
- **ERROR_VALIDATION_ERROR** (*str*) – Some data in the registration request was not valid
- **ERROR_INVALID_DTO** (*str*) – The request was not valid - faulty JSON
- **ERROR_KEY_MISSING** (*str*) – Device key was not provided
- **ERROR_SUBDEVICE_MANAGEMENT_FORBIDDEN** (*str*) – Gateway is not able to register devices
- **ERROR_UNKNOWN** (*str*) – Unknown error occurred

MQTT message model.

class wolk_gateway_module.model.message.**Message**(*topic: str*, *payload: Union[str, bytes, bytearray, None] = None*)

MQTT message identified by topic and payload.

Variables

- **topic** (*str*) – Topic where the message is from or will be sent to
- **payload** (*bytes or str or bytearray or None*) – Content of the message

Sensor reading model.

class wolk_gateway_module.model.sensor_reading.**SensorReading**(*reference: str, value: Union[bool, int, float, str, Tuple[int, int], Tuple[int, int, int], Tuple[float, float], Tuple[float, float, float], Tuple[str, str], Tuple[str, str, str]], timestamp: Optional[int] = None*)

Holds information about a sensor reading.

Variables

- **reference** (*str*) – Device sensor's reference as defined in device template
- **value** (*bool or int or float or str or Tuple[int, int] or Tuple[int, int, int] or Tuple[float, float] or Tuple[float, float, float] or Tuple[str, str] or Tuple[str, str, str]*) – Data that the sensor reading yielded
- **timestamp** (*Optional[int]*) – Unix timestamp in miliseconds. If not provided, Platform will assign timestamp when it receives it.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

w

Index

Symbols

A

C

D

E

F

G

H

I

J

M

O

P

Q

R

S

T

W