
DCD Lab Python Kit

Release 0.1.30

Mar 31, 2023

Contents:

1	Getting Started	1
1.1	Create a Thing	1
1.2	Setup a Python project	1
1.3	Basic example	2
1.4	Environment Variables	3
2	Bucket	5
2.1	Thing	5
2.2	Authentication	6
2.3	Network	7
2.4	Properties	8
3	Environment Variables	11
4	Indices and tables	13
	Index	15

Requirements: Python 3

1.1 Create a Thing

To interact with Bucket, the first step is to visit [Bucket](#) to create an account and a Thing.

During this process you will get the ID of your Thing (starting with *dcd:things:...*) and you will generate a public/private key.

These 2 pieces of information are needed for your Python code to interact with Bucket.

1.2 Setup a Python project

Create a folder for your project and open it with VisualStudio Code.

To avoid disturbing other Python setup on your machine, we setup a virtual environment with *virtualenv*. To create a virtual environment called *venv*, open the terminal (VisualStudio Code, directly in your project) and execute the following command:

```
virtualenv venv
```

Then we activate this environment with source:

```
source venv/bin/activate
```

If it worked properly, you should see *(venv)* appearing on the left side of the line in the Terminal.

We can now install the DCD SDK library

```
pip install dcd-sdk
```

At this stage you can be prompted to update your pip module, you can do so with:

```
pip install --upgrade pip
```

Your Python setup is now ready.

1.3 Basic example

In this example, we will create a property Accelerometer generating random values. It shows how to establish a connection with Bucket using your Thing id and your private key. This is a typical case for a Python code running on a device to collect data.

In the file explorer (left-side panel), create a new file *example.py* and add the following lines.

```
1 # Import Thing from the Data-Centric Design
2 from dcd.bucket.thing import Thing
3
4 # Create an instance of Thing
5 # (Replace with your thing id and the path to your private key)
6 my_thing = Thing(thing_id="dcd:things:7f7fe4c6-45e9-42d2-86e2-a6794e386108",
7                    private_key_path="/path/to/private.pem")
```

You can run this example in the terminal:

```
python example.py
```

To stop the program, press *CTRL+C*.

Once the connection is established with your Thing, we can get an overview of this Thing by printing the output of the method `to_json()`. Add the following line at the bottom of the file and run the program again. If you just registered your Thing on Bucket, it has only an id, a name and a type.

```
print(my_thing.describe())
```

Let's create a property 'My Python accelerometer'. The method `find_or_create()` looks for an existing property with this name. If none is found, it creates a new one with the type 'ACCELEROMETER'

```
my_property = my_thing.find_or_create_property(
    "My Python Accelerometer", "ACCELEROMETER")
```

Let's have a look at the property, it should contain the name and a unique id. The type also contains the dimensions, 3 in the case of an accelerometer.

```
print(my_property.describe())
```

We are ready to send data. In the code below we create a function that generates an array with 3 random values and add them to the property. We then make an infinite loop (`while True`) to send these random values every 2 seconds.

To generate random numbers we need the library *random* and to wait 2 seconds we need the library *time*. These are part of Python, we just import them at the top of the file.

```
from random import random
import time
```

Then, we can write our function at the bottom of the file.

```
1  # Let's create a function that generate random values
2  def generate_dum_property_values(the_property):
3      # Define a tuple with the current time, and 3 random values
4      values = (random(), random(), random())
5      # Update the values of the property
6      the_property.update_values(values)
7
8  # Finally, we call our function to start generating dum values
9  while True:
10     generate_dum_property_values(my_property)
11     # Have a 2-second break
12     time.sleep(2)
```

1.4 Environment Variables

To avoid credentials in your code, the DCD Python Kit is looking for your thing id and private key from the environment variables. To set these variables, create a file `.env` and add the following lines (replace the thing id and the path by yours).

```
THING_ID=dcd:things:7f7fe4c6-45e9-42d2-86e2-a6794e386108
PRIVATE_KEY_PATH=/path/to/private.pem
```

The full example can be found [Here](#)

2.1 Thing

```
class dcd.bucket.thing.Thing (thing_id: str = None, private_key_path: str = None, private_key:  
                             str = None, json_thing: dict = None, connect=True)
```

This is a conceptual class representation of a physical or virtual entity collecting data.

Attributes:

thing_id [str] The id of the Thing, starting with “dcd:things:”.

name [str] Name of the Thing

description [str] Description of the Thing

thing_type [str] Type of the Thing

properties [Property[]] Properties of the Thing

private_key_path [str] Path to the private key to use for the generation of authentication tokens.

created_at [int] Creation time of the Thing on Bucket (UNIX timestamp)

updated_at [int] Last update time of the Thing on Bucket (UNIX timestamp)

describe ()

Prints formatted JSON with the details of the Thing

```
find_or_create_property (property_name: str, type_id: str) →  
                        dcd.bucket.properties.property.Property
```

Search for a property in thing by name, create it if not found & return it.

Args:

property_name [str] The name of the property to look for.

type_id [str] The type of the property, so that we can create it if it is not found.

Returns: Property: The found or newly created Property.

find_property_by_name (*property_name_to_find: str*) → dcd.bucket.properties.property.Property
Search for a property in thing by name

Args:

property_name_to_find [str] The name of the property to look for.

Returns: Property: The found property, None if not found

find_shared_properties (*group='*'*) → List[dcd.bucket.properties.property.Property]
Search for properties that are accessible by the Thing.

Args: group (str, optional): [description]. Defaults to “*”, fetching for all groups.

Returns: [Property]: Shared properties accessible by the Thing.

read_property (*property_id: str, from_ts: int = None, to_ts: int = None, time_interval=None, time_fct=None, fill=None, shared_with=None*) → dcd.bucket.properties.property.Property
Read the details of a property from Bucket

Args:

property_id [str] The id of the property to read

from_ts [int, optional] The start time of the values to fetch. Defaults to None.

to_ts [int, optional] The end time of the values to fetch. Defaults to None.

Raises: ValueError: The requested property is not part of the Thing
ValueError: Could not parse the reponse

Returns: Property: The property with its details and values.

update_property (*prop: dcd.bucket.properties.property.Property, file_name: str = None*)
Send new property values to Bucket

Args:

prop [Property] The property containing values to send

file_name [str, optional] If media type property, the path to the file to upload. Defaults to None.

2.2 Authentication

class dcd.bucket.thing.**ThingToken** (*private_key_path: str, subject: str, issuer: str, audience: str, algorithm='RS256'*)

Handle JSON web token for the Thing authentication

decode (*public_key_path: str = None, jwt_token: str = None*) → dict
Decode a JWT, revealing the dictionary of its values

Args:

public_key_path [str, optional] The path to the public key. If none provided, looking at PUBLIC_KEY_PATH environment variable, or use './public.pem' as default. Defaults to None.

jwt_token [str, optional] String representing the JSON web token. If none provided, taking the one from the class Defaults to None.

Returns: dict: Decoded JSON Web Token including the issuer (iss), audience (aud), subject (sub), the creation date (iat) and the expiration date (exp)

get_token() → str

Check if the current JWT is still valid, refresh it if necessary and returns it.

Returns: str: The existing (and still valid) JWT or a newly generated JWT

refresh(*duration_sec: int = 3600*) → str

Use the private key to generate a new JWT.

Args: *duration_sec* (int, optional): The life time of the token in seconds

Returns: str: the resulting JSON web token

2.3 Network

class dcd.bucket.thing.**ThingHTTP**(*thing, http_uri: str*)

Handle Bucket interaction for a Thing via HTTP

create_property(*name: str, type_id: str*)

Create a new property on Bucket.

Args:

name [str] Name of the property to create

type_id [str] Type id of the property to create

Returns: Property: The newly created property

is_connected() → bool

Check whether the HTTP connection was established.

Returns: bool: Whether the initial HTTP request *read()* succeeded.

read() → bool

Read details of the Thing from Bucket.

Returns: bool: True if succeeded in reading the Thing details from Bucket

read_property(*property_id: str, from_ts: int = None, to_ts: int = None, time_interval=None, time_fct=None, fill=None, shared_with=None*) → dcd.bucket.properties.property.Property

Read the details of a property from Bucket

Args:

property_id [str] The id of the property to read

from_ts [int, optional] The start time of the values to fetch. Defaults to None.

to_ts [int, optional] The end time of the values to fetch. Defaults to None.

Raises: ValueError: The requested property is not part of the Thing
ValueError: Could not parse the reponse

Returns: Property: The property with its details and values.

update_property(*prop: dcd.bucket.properties.property.Property, file_name: str = None*) → int

Update the values of a property on Bucket

Args:

prop [Property] The property to update

file_name [str, optional] The media to upload. Defaults to None.

Returns: int: Status response code

```
class dcd.bucket.thing.ThingMQTT (thing, DigiCertCA='DigicertCA', connect=True)
```

```
find_or_create_property (property_name: str, type_id: str)
```

Search for a property in thing by name, create it if not found & return it.

Args:

property_name [str] The name of the property to look for.

type_id [str] The type of the property, so that we can create it if it is not found.

```
update_property (prop: dcd.bucket.properties.property.Property, file_name: str)
```

Send new property values to Bucket

Args:

prop [Property] The property containing values to send

file_name [str, optional] If media type property, the path to the file to upload. Defaults to None.

2.4 Properties

```
class dcd.bucket.properties.property.Property (property_id: str = None, name: str =  
                                                None, description: str = None, type_id:  
                                                str = None, property_type: dict = None,  
                                                json_property: dict = None, values: dict  
                                                = {}, thing=None)
```

” A DCD “Property” represents a numerical property of a Thing.

```
align_values_to (prop2)
```

Create if missing, an intermediary row of values for each timestamp in prop2

```
merge (prop2)
```

Create a new Property with id and name of form “prop1+prop2”, concat dimension and values (MUST have same number of rows) and return this new property

```
read (from_ts=None, to_ts=None, time_interval=None, time_fct=None, shared_with: str = None)
```

Read the details of a property from Bucket

Args:

from_ts [int|str, optional] The start time of the values to fetch. Can be a UNIX timestamp in milliseconds or a string date ‘%Y-%m-%d %H:%M:%S’. Defaults to None.

to_ts [int|str, optional] The end time of the values to fetch. Can be a UNIX timestamp in milliseconds or a string date ‘%Y-%m-%d %H:%M:%S’. Defaults to None.

Returns: Property: The property with its details and values.

```
sync ()
```

Upload values to the server and clean up the local values

```
update_values (values: dict, time_ms: int = None, file_name: str = None, mode: str = 'w')
```

Update the values of a property.

Args:

values: dict List containing values [val-dim1, val-dim2,...]

time_ms [int, optional] Collection time in milliseconds to add to the values. Default to None

file_name: path to a file to upload (e.g. for video, image or audio properties) mode : str, optional

Control how to manage existing values. - 'w' overwrite existing (local) values, replace with new values, and send to the server - 'a' append new values after existing values. In this mode, you must call sync() to upload the data on the server. TODO: does not work with files yet. Defaults to 'overwrite'.

Environment Variables

There are settings you can provision through environment variables, provisioning them via a file `.env` at the root of your project.

Here is the full list with there default value.

```
# The id of your thing, instead of having to change your code
THING_ID=

# The path to your public and private keys if they are not
# in the root folder of your project
PRIVATE_KEY_PATH=private.pem
PUBLIC_KEY_PATH=public.pem

# The path to the folder where to store logs
LOG_PATH=./logs/
# The level of logs to generate (ERROR, WARN, INFO,DEBUG)
LOG_LEVEL=DEBUG
# The path to the folder where to store data. Data is stored
# per Thing folders and Property files
DATA_PATH=./data/

# The URI to Bucket, when targeting a different version or
# an instance running on a different server
HTTP_API_URI=https://dwd.tudelft.nl:443/bucket/api

# The MQTT host, port and security (mqtt or mqttts) to target
# a different version or an instance running on a different server
MQTT_HOST=dwd.tudelft.nl
MQTT_PORT=8883
MQTT_SECURED=True
```


CHAPTER 4

Indices and tables

- `genindex`
- `modindex`
- `search`

A

`align_values_to()`
(*dcd.bucket.properties.property.Property*
method), 8

C

`create_property()` (*dcd.bucket.thing.ThingHTTP*
method), 7

D

`decode()` (*dcd.bucket.thing.ThingToken method*), 6
`describe()` (*dcd.bucket.thing.Thing method*), 5

F

`find_or_create_property()`
(*dcd.bucket.thing.Thing method*), 5
`find_or_create_property()`
(*dcd.bucket.thing.ThingMQTT method*), 8
`find_property_by_name()`
(*dcd.bucket.thing.Thing method*), 5
`find_shared_properties()`
(*dcd.bucket.thing.Thing method*), 6

G

`get_token()` (*dcd.bucket.thing.ThingToken method*),
6

I

`is_connected()` (*dcd.bucket.thing.ThingHTTP*
method), 7

M

`merge()` (*dcd.bucket.properties.property.Property*
method), 8

P

`Property` (*class in dcd.bucket.properties.property*), 8

R

`read()` (*dcd.bucket.properties.property.Property*
method), 8
`read()` (*dcd.bucket.thing.ThingHTTP method*), 7
`read_property()` (*dcd.bucket.thing.Thing method*),
6
`read_property()` (*dcd.bucket.thing.ThingHTTP*
method), 7
`refresh()` (*dcd.bucket.thing.ThingToken method*), 7

S

`sync()` (*dcd.bucket.properties.property.Property*
method), 8

T

`Thing` (*class in dcd.bucket.thing*), 5
`ThingHTTP` (*class in dcd.bucket.thing*), 7
`ThingMQTT` (*class in dcd.bucket.thing*), 8
`ThingToken` (*class in dcd.bucket.thing*), 6

U

`update_property()` (*dcd.bucket.thing.Thing*
method), 6
`update_property()` (*dcd.bucket.thing.ThingHTTP*
method), 7
`update_property()` (*dcd.bucket.thing.ThingMQTT*
method), 8
`update_values()` (*dcd.bucket.properties.property.Property*
method), 8