bimmer_connected Documentation

Release 0.7.15.dev2

m1n3rva

Feb 25, 2021

Readme

This is a simple library to query and control the status of your BMW or Mini vehicle from the ConnectedDrive portal.

CHAPTER 1

Installation

`bimmer_connected` requires **Python 3.6 or above** but should also run with Python 3.5. Just install the latest release from PyPI using `pip3 install --upgrade bimmer_connected`.

Alteratively, clone the project and execute `pip install -e .` to install the current `master` branch.

CHAPTER 2

Usage

After installation, execute `bimmerconnected` from command line for usage instruction or see the full CLI documentation.

The description of the `modules` can be found in the module documentation.

This library is written to be included in Home Assistant.

CHAPTER 3

Compatibility

This works with BMW (and Mini) vehicles with a ConnectedDrive account. So far it is tested on vehicles with a 'MGU', 'NBTEvo', 'EntryEvo', 'NBT', or 'EntryNav' navigation system. If you have any trouble with other navigation systems, please create an issue with your server responses (see next section).

To use this library, your BMW (or Mini) must have the remote services enabled for your vehicle. You might need to book this in the ConnectedDrive/Mini Connected portal and this might cost some money. In addition to that you need to enable the Remote Services in your infotainment system in the vehicle.

Different models of vehicles and infotainment systems result in different types of attributes provided by the server. So the experience with the library will certainly vary across the different vehicle models.

CHAPTER 4

Data Contributions

If some features do not work for your vehicle, we would need the data returned form the server to analyse this and potentially extend the code. Different models and head unit generations lead to different responses from the server.

If you want to contribute your data, perform the following steps:

```
# get the latest version of the library
pip3 install --upgrade bimmer_connected

# run the fingerprint function
bimmerconnected fingerprint <username> <password> <region>
```

This will create a set of log files in the "vehicle_fingerprint" folder. Before sending the data to anyone please **check for any personal data** such as **dealer name** or **country**.

The following attributes are by default replaced with anonymized values:

- `vin` (Vehicle Identification Number)
- `lat` and `lon` (GPS position)
- `licensePlate`
- `information of dealer`

Create a new fingerprint data contribution and add the files as attachment to the discussion.

Please add your model and year to the title of the issue, to make it easier to organize. If you know the "chassis code" of your car, you can include that too. (For example, googling "2017 BMW X5" will show a Wikipedia article entitled "BMW X5 (F15)". F15 is therefore the chassis code of the car.)

Note: We will then use this data as additional test cases. So we will publish (parts of) it (after checking for personal information again) and use this as test cases for our library. If you do not want this, please let us know in advance.

CHAPTER 5

Code Contributions

Contributions are welcome! Please make sure that your code passed the `tox` checks. We currently test against `flake8`, `pylint` and our own `pytest` suite. And please add tests where it makes sense. The more the better.

See the contributing guidelines for more details.

CHAPTER 6

Thank you

Thank you to all contributors for your research and contributions! And thanks to everyone who shares the fingerprint data of their vehicles which we use to test the code.

This library is basically a best-of of other similar solutions, yet none of them provided a ready to use library with a matching interface to be used in Home Assistant and is available on pypi.

- https://github.com/edent/BMW-i-Remote
- https://github.com/jupe76/bmwcdapi
- https://github.com/frankjoke/iobroker.bmw

Thank you for your great software!

CHAPTER 7

License

The bimmer_connected library is licensed under the Apache License 2.0.

CHAPTER 8

Disclaimer

This library is not affiliated with or endorsed by BMW Group.

8.1 bimmer_connected

This is a simple library to query and control the status of your BMW or Mini vehicle from the ConnectedDrive portal.

8.2 Installation

`bimmer_connected` requires **Python 3.6 or above** but should also run with Python 3.5. Just install the latest release from PyPI using `pip3 install --upgrade bimmer_connected`.

Alteratively, clone the project and execute `pip install -e .` to install the current `master` branch.

8.3 Usage

After installation, execute `bimmerconnected` from command line for usage instruction or see the full CLI documentation.

The description of the `modules` can be found in the module documentation.

This library is written to be included in Home Assistant.

8.4 Compatibility

This works with BMW (and Mini) vehicles with a ConnectedDrive account. So far it is tested on vehicles with a 'MGU', 'NBTEvo', 'EntryEvo', 'NBT', or 'EntryNav' navigation system. If you have any trouble with other navigation systems, please create an issue with your server responses (see next section).

To use this library, your BMW (or Mini) must have the remote services enabled for your vehicle. You might need to book this in the ConnectedDrive/Mini Connected portal and this might cost some money. In addition to that you need to enable the Remote Services in your infotainment system in the vehicle.

Different models of vehicles and infotainment systems result in different types of attributes provided by the server. So the experience with the library will certainly vary across the different vehicle models.

8.5 Data Contributions

If some features do not work for your vehicle, we would need the data returned form the server to analyse this and potentially extend the code. Different models and head unit generations lead to different responses from the server.

If you want to contribute your data, perform the following steps:

```
# get the latest version of the library
pip3 install --upgrade bimmer_connected

# run the fingerprint function
bimmerconnected fingerprint <username> <password> <region>
```

This will create a set of log files in the "vehicle_fingerprint" folder. Before sending the data to anyone please **check for any personal data** such as **dealer name** or **country**.

The following attributes are by default replaced with anonymized values:

- `vin` (Vehicle Identification Number)
- `lat` and `lon` (GPS position)
- `licensePlate`
- `information of dealer`

Create a new fingerprint data contribution and add the files as attachment to the discussion.

Please add your model and year to the title of the issue, to make it easier to organize. If you know the "chassis code" of your car, you can include that too. (For example, googling "2017 BMW X5" will show a Wikipedia article entitled "BMW X5 (F15)". F15 is therefore the chassis code of the car.)

Note: We will then use this data as additional test cases. So we will publish (parts of) it (after checking for personal information again) and use this as test cases for our library. If you do not want this, please let us know in advance.

8.6 Code Contributions

Contributions are welcome! Please make sure that your code passed the `tox` checks. We currently test against `flake8`, `pylint` and our own `pytest` suite. And please add tests where it makes sense. The more the better.

See the contributing guidelines for more details.

8.7 Thank you

Thank you to all contributors for your research and contributions! And thanks to everyone who shares the fingerprint data of their vehicles which we use to test the code.

This library is basically a best-of of other similar solutions, yet none of them provided a ready to use library with a matching interface to be used in Home Assistant and is available on pypi.

- https://github.com/edent/BMW-i-Remote
- https://github.com/jupe76/bmwcdapi
- https://github.com/frankjoke/iobroker.bmw

Thank you for your great software!

8.8 License

The bimmer_connected library is licensed under the Apache License 2.0.

8.9 Disclaimer

This library is not affiliated with or endorsed by BMW Group.

8.10 `bimmerconnected`

A simple executable to use and test the library.

```
usage: bimmerconnected [-h]
                       {status,fingerprint,lightflash,vehiclefinder,image,sendpoi,
↪sendpoi_from_address,sendmessage}
                       ...
```

8.10.1 Positional Arguments

cmd Possible choices: status, fingerprint, lightflash, vehiclefinder, image, sendpoi, sendpoi_from_address, sendmessage

8.10.2 Sub-commands:

status

Get the current status of the vehicle.

```
bimmerconnected status [-h]
                       username password {north_america,china,rest_of_world}
                       [lat] [lng]
```

Positional Arguments

username	Connected Drive username
password	Connected Drive password
region	Possible choices: north_america, china, rest_of_world Region of the Connected Drive account
lat	(optional) Your current GPS latitude (as float)
lng	(optional) Your current GPS longitude (as float)

fingerprint

Save a vehicle fingerprint.

```
bimmerconnected fingerprint [-h]
                            username password
                            {north_america,china,rest_of_world} [lat] [lng]
```

Positional Arguments

username	Connected Drive username
password	Connected Drive password
region	Possible choices: north_america, china, rest_of_world Region of the Connected Drive account
lat	(optional) Your current GPS latitude (as float)
lng	(optional) Your current GPS longitude (as float)

lightflash

Flash the vehicle lights.

```
bimmerconnected lightflash [-h]
                           username password
                           {north_america,china,rest_of_world} vin
```

Positional Arguments

username	Connected Drive username
password	Connected Drive password

region	Possible choices: north_america, china, rest_of_world
	Region of the Connected Drive account
vin	Vehicle Identification Number

vehiclefinder

Update the vehicle GPS location.

```
bimmerconnected vehiclefinder [-h]
                              username password
                              {north_america,china,rest_of_world} vin
```

Positional Arguments

username	Connected Drive username
password	Connected Drive password
region	Possible choices: north_america, china, rest_of_world
	Region of the Connected Drive account
vin	Vehicle Identification Number

image

Download a vehicle image.

```
bimmerconnected image [-h]
                      username password {north_america,china,rest_of_world}
                      vin
```

Positional Arguments

username	Connected Drive username
password	Connected Drive password
region	Possible choices: north_america, china, rest_of_world
	Region of the Connected Drive account
vin	Vehicle Identification Number

sendpoi

Send a point of interest to the vehicle.

```
bimmerconnected sendpoi [-h] [--name [NAME]] [--street [STREET]]
                        [--city [CITY]] [--postalcode [POSTALCODE]]
                        [--country [COUNTRY]]
                        username password {north_america,china,rest_of_world}
                        vin latitude longitude
```

Positional Arguments

username	Connected Drive username
password	Connected Drive password
region	Possible choices: north_america, china, rest_of_world Region of the Connected Drive account
vin	Vehicle Identification Number
latitude	Latitude of the POI
longitude	Longitude of the POI

Named Arguments

--name	(optional, display only) Name of the POI
--street	(optional, display only) Street & House No. of the POI
--city	(optional, display only) City of the POI
--postalcode	(optional, display only) Postal code of the POI
--country	(optional, display only) Country of the POI

sendpoi_from_address

Send a point of interest parsed from a street address to the vehicle.

```
bimmerconnected sendpoi_from_address [-h] [-n [NAME]]
                                     [-a ADDRESS [ADDRESS ...]]
                                     username password
                                     {north_america,china,rest_of_world} vin
```

Positional Arguments

username	Connected Drive username
password	Connected Drive password
region	Possible choices: north_america, china, rest_of_world Region of the Connected Drive account
vin	Vehicle Identification Number

Named Arguments

-n, --name	(optional, display only) Name of the POI
-a, --address	Address (e.g. 'Street 17, city, zip, country')

sendmessage

Send a text message to the vehicle.

```
bimmerconnected sendmessage [-h]
                            username password
                            {north_america,china,rest_of_world} vin text
                            [subject]
```

Positional Arguments

username	Connected Drive username
password	Connected Drive password
region	Possible choices: north_america, china, rest_of_world Region of the Connected Drive account
vin	Vehicle Identification Number
text	Text to be sent.
subject	(optional) Message subject

8.11 bimmer_connected.account

Library to read data from the BMW Connected Drive portal.

The library bimmer_connected provides a Python interface to interact with the BMW Connected Drive web service. It allows you to read the current state of the vehicle and also trigger remote services.

Disclaimer: This library is not affiliated with or endorsed by BMW Group.

class bimmer_connected.account.**ConnectedDriveAccount**(*username: str*, *password: str*, *region: bimmer_connected.country_selector.Regions*, *log_responses: pathlib.Path = None*, *retries_on_500_error: int = 5*)

Create a new connection to the BMW Connected Drive web service.

Parameters

- **username** – Connected drive user name
- **password** – Connected drive password
- **country** – Country for which the account was created. For a list of valid countries, check https://www.bmw-connecteddrive.com . Use the name of the countries exactly as on the website.
- **log_responses** – If log_responses is set, all responses from the server will be loged into this directory. This can be used for later analysis of the different responses for different vehicles.
- **retries_on_500_error** – If retries_on_500_error is set, a communication with the Connected Drive server will automatically be retried the number of times specified in the

event the error code received was 500. This sometimes occurs (presumably) due to bugs in the server implementation.

add_update_listener(*listener: Callable*) → None
Add a listener for state updates.

get_vehicle(*vin: str*) → bimmer_connected.vehicle.ConnectedDriveVehicle
Get vehicle with given VIN.

The search is NOT case sensitive. :param vin: VIN of the vehicle you want to get. :return: Returns None if no such vehicle is found.

request_header
Generate a header for HTTP requests to the server.

send_request(*url: str*, *data=None*, *headers=None*, *expected_response=200*, *post=False*, *allow_redirects=True*, *logfilename: str = None*, *params: dict = None*)
Send an http request to the server.

If the http headers are not set, default headers are generated. You can choose if you want a GET or POST request.

server_url
Get the url of the server for this country.

set_observer_position(*latitude: float*, *longitude: float*) → None
Set the position of the observer for all vehicles.

see VehicleViewDirection.set_observer_position() for more details.

update_vehicle_states() → None
Update the state of all vehicles.

Notify all listeners of the vehicle state update.

vehicles
Get list of vehicle of this account

8.12 bimmer_connected.country_selector

Get the right url for the different countries.

class bimmer_connected.country_selector.**Regions**
Regions of the world with separate servers.

CHINA = 1

NORTH_AMERICA = 0

REST_OF_WORLD = 2

bimmer_connected.country_selector.**get_gcdm_oauth_authorization**(*region: bimmer_connected.country_selector.Regions*) → str
Get the url of the server for the region.

bimmer_connected.country_selector.**get_gcdm_oauth_endpoint**(*region: bimmer_connected.country_selector.Regions*) → str
Get the url of the server for the region.

bimmer_connected.country_selector.**get_region_from_name**(*name: str*) → bimmer_connected.country_selector.Regions

Get a region for a string.

This function is not case-sensitive.

bimmer_connected.country_selector.**get_server_url**(*region: bimmer_connected.country_selector.Regions*) → str

Get the url of the server for the region.

bimmer_connected.country_selector.**valid_regions**() → List[str]

Get list of valid regions as strings.

8.13 bimmer_connected.remote_services

Trigger remote services on a vehicle.

class bimmer_connected.remote_services.**ExecutionState**

Enumeration of possible states of the execution of a remote service.

DELIVERED = 'DELIVERED'

EXECUTED = 'EXECUTED'

INITIATED = 'INITIATED'

PENDING = 'PENDING'

UNKNOWN = 'UNKNOWN'

class bimmer_connected.remote_services.**Message**(*data: dict*)

Text message or PointOfInterst to be sent to the vehicle.

as_server_request

Convert to a dictionary so that it can be sent to the server.

classmethod from_poi(*poi: bimmer_connected.remote_services.PointOfInterest*)

Create a message from a PointOfInterest

classmethod from_text(*text: str*, *subject: str = None*)

Create a text message

class bimmer_connected.remote_services.**PointOfInterest**(*lat: float*, *lon: float*, *name: str = None*, *additional_info: str = None*, *street: str = None*, *city: str = None*, *postal_code: str = None*, *country: str = None*, *website: str = None*, *phone_numbers: [<class 'str'>] = None*)

Point of interest to be sent to the vehicle.

The latitude/longitude of a POI are mandatory, all other attributes are optional. CamelCase attribute names are used here so that we do not have to convert the names between the attributes and the keys as expected on the server.

class bimmer_connected.remote_services.**RemoteServiceStatus**(*response: dict*)

Wraps the status of the execution of a remote service.

class bimmer_connected.remote_services.**RemoteServices**(*account*, *vehicle*)
Trigger remote services on a vehicle.

trigger_remote_air_conditioning() → bimmer_connected.remote_services.RemoteServiceStatus
Trigger the air conditioning to start.

A state update is NOT triggered after this, as the vehicle state is unchanged.

trigger_remote_door_lock() → bimmer_connected.remote_services.RemoteServiceStatus
Trigger the vehicle to lock its doors.

A state update is triggered after this, as the lock state of the vehicle changes.

trigger_remote_door_unlock() → bimmer_connected.remote_services.RemoteServiceStatus
Trigger the vehicle to unlock its doors.

A state update is triggered after this, as the lock state of the vehicle changes.

trigger_remote_horn() → bimmer_connected.remote_services.RemoteServiceStatus
Trigger the vehicle to sound its horn.

A state update is NOT triggered after this, as the vehicle state is unchanged.

trigger_remote_light_flash() → bimmer_connected.remote_services.RemoteServiceStatus
Trigger the vehicle to flash its headlights.

A state update is NOT triggered after this, as the vehicle state is unchanged.

trigger_remote_vehicle_finder() → bimmer_connected.remote_services.RemoteServiceStatus
Trigger the vehicle finder.

A state update is triggered after this, as the location state of the vehicle changes.

trigger_send_message(*data: dict*) → bimmer_connected.remote_services.RemoteServiceStatus
Send a message to the vehicle.

Parameters **data** (*dict*) – A dictonary containing a 'text' and an optional 'subject'

A state update is NOT triggered after this, as the vehicle state is unchanged.

trigger_send_poi(*data: dict*) → bimmer_connected.remote_services.RemoteServiceStatus
Send a PointOfInterest to the vehicle.

Parameters **data** (*dict*) – A dictonary containing at least 'lat' and 'lon' and optionally 'name', 'additionalInfo', 'street', 'city', 'postalCode', 'country', 'website' or 'phoneNumbers'

A state update is NOT triggered after this, as the vehicle state is unchanged.

8.14 bimmer_connected.state

Models the state of a vehicle.

class bimmer_connected.state.**VehicleState**(*account*, *vehicle*)
Models the state of a vehicle.

all_lids_closed

all_windows_closed

are_all_cbs_ok

are_parking_lights_on

attributes

charging_level_hv

charging_status

charging_time_remaining

check_control_messages

condition_based_services

connection_status

door_lock_state

gps_position

has_check_control_messages

is_vehicle_tracking_enabled

last_charging_end_result

last_update_reason

lids

max_range_electric

mileage

open_lids

open_windows

parking_lights

remaining_fuel

remaining_range_electric

remaining_range_fuel

remaining_range_total

timestamp

update_data() → None
Read new status data from the server.

windows

bimmer_connected.state.**backend_parameter**(*func*)
Decorator for parameters reading data from the backend.

Errors are handled in a default way.

8.15 bimmer_connected.vehicle

Models state and remote services of one vehicle.

bimmer_connected.vehicle.**COMBUSTION_ENGINE_DRIVE_TRAINS = {<DriveTrainType.CONVENTIONAL: 'C**
Set of drive trains that have a combustion engine

class bimmer_connected.vehicle.**ConnectedDriveVehicle**(*account*, *attributes: dict*)
Models state and remote services of one vehicle.

Parameters

- **account** – ConnectedDrive account this vehicle belongs to
- **attributes** – attributes of the vehicle as provided by the server

available_attributes
Get the list of non-drivetrain attributes available for this vehicle.

available_state_services
Get the list of all available state services for this vehicle.

drive_train
Get the type of drive train of the vehicle.

drive_train_attributes
Get list of attributes available for the drive train of the vehicle.

The list of available attributes depends if on the type of drive train. Some attributes only exist for electric/hybrid vehicles, others only if you have a combustion engine. Depending on the state of the vehicle, some of the attributes might still be None.

get_vehicle_image(*width: int*, *height: int*, *direction: bimmer_connected.vehicle.VehicleViewDirection*) → bytes
Get a rendered image of the vehicle.

:returns bytes containing the image in PNG format.

has_destination_service
Return True if destinations are available.

has_hv_battery
Return True if vehicle is equipped with a high voltage battery.

In this case we can get the state of the battery in the state attributes.

has_internal_combustion_engine
Return True if vehicle is equipped with an internal combustion engine.

In this case we can get the state of the gas tank.

has_range_extender
Return True if vehicle is equipped with a range extender.

In this case we can get the state of the gas tank.

has_rangemap_service
Return True if rangemap (range circle) is available.

has_statistics_service
Return True if statistics are available.

has_weekly_planner_service
Return True if charging control (weekly planner) is available.

lsc_type
Get the lscType of the vehicle.

Not really sure what that value really means. If it is NOT_SUPPORTED, that probably means that the vehicle state will not contain much data.

name
Get the name of the vehicle.

rangemap_service_type
Returns the vehicles rangemap service type if available.

set_observer_position(*latitude: float*, *longitude: float*) → None
Set the position of the observer, who requests the vehicle state.

Some vehicle require you to send your position to the server before you get the vehicle state. Your position must be within some range (2km?) of the vehicle to get you a proper answer.

update_state() → None
Update the state of a vehicle.

class bimmer_connected.vehicle.**DriveTrainType**
Different types of drive trains.

BEV = 'BEV'

BEV_REX = 'BEV_REX'

CONVENTIONAL = 'CONV'

PHEV = 'PHEV'

bimmer_connected.vehicle.**HV_BATTERY_DRIVE_TRAINS = {<DriveTrainType.BEV: 'BEV'>, <DriveTra**
set of drive trains that have a high voltage battery

class bimmer_connected.vehicle.**LscType**
Known Values for lsc_type field.

Not really sure, what this value really contains.

I_LSC_IMM = 'I_LSC_IMM'

LSC_BASIS = 'LSC_BASIS'

LSC_PHEV = 'LSC_PHEV'

NOT_SUPPORTED = 'NOT_SUPPORTED'

UNKNOWN = 'UNKNOWN'

bimmer_connected.vehicle.**RANGE_EXTENDER_DRIVE_TRAINS = {<DriveTrainType.BEV_REX: 'BEV_REX'>**
Set of drive trains that have a range extender

class bimmer_connected.vehicle.**VehicleViewDirection**
Viewing angles for the vehicle.

This is used to get a rendered image of the vehicle.

DASHBOARD = 'DASHBOARD'

DRIVERDOOR = 'DRIVERDOOR'

FRONT = 'FRONT'

FRONTSIDE = 'FRONTSIDE'

REAR = 'REAR'

REARBIRDSEYE = 'REARBIRDSEYE'

REARSIDE = 'REARSIDE'

SIDE = 'SIDE'

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

U

V

W