

---

# **bimmer\_connected Documentation**

**m1n3rva**

**Apr 30, 2024**



# README

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Usage</b>	<b>5</b>
2.1	Example in an asyncio event loop . . . . .	5
2.2	Example in non-async code . . . . .	5
<b>3</b>	<b>Compatibility</b>	<b>7</b>
<b>4</b>	<b>Data Contributions</b>	<b>9</b>
<b>5</b>	<b>Code Contributions</b>	<b>11</b>
<b>6</b>	<b>Thank you</b>	<b>13</b>
<b>7</b>	<b>License</b>	<b>15</b>
<b>8</b>	<b>Disclaimer</b>	<b>17</b>
8.1	bimmer_connected . . . . .	17
8.2	Installation . . . . .	17
8.3	Usage . . . . .	18
8.4	Compatibility . . . . .	19
8.5	Data Contributions . . . . .	19
8.6	Code Contributions . . . . .	20
8.7	Thank you . . . . .	20
8.8	License . . . . .	20
8.9	Disclaimer . . . . .	20
8.10	Using fingerprints in Home Assistant . . . . .	20
8.11	Reverse engineering the MyBMW API . . . . .	21
8.12	bimmerconnected . . . . .	24
8.13	bimmer_connected.account . . . . .	30
8.14	bimmer_connected.api . . . . .	32
8.15	bimmer_connected.const . . . . .	34
8.16	bimmer_connected.models . . . . .	35
8.17	bimmer_connected.utils . . . . .	38
8.18	bimmer_connected.vehicle . . . . .	38
<b>9</b>	<b>Indices and tables</b>	<b>51</b>
	<b>Python Module Index</b>	<b>53</b>
	<b>Index</b>	<b>55</b>



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



## INSTALLATION

bimmer\_connected is tested against **Python 3.8 or above**. Just install the latest release from [PyPI](#) using `pip3 install --upgrade bimmer_connected`.

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

---

**Note:** If you want to connect to a **chinese** server, you need to install the `[china]` extra, e.g. `pip3 install --upgrade bimmer_connected[china]`.

---





## USAGE

While this library is mainly written to be included in [Home Assistant](#), it can be use on its own.

After installation, execute `bimmerconnected` from command line for usage instruction or see the full [CLI documentation](#).

Please be aware that `bimmer_connected` is an `async` library when using it in Python code. The description of the modules can be found in the [module documentation](#).

### 2.1 Example in an asyncio event loop

```
import asyncio
from bimmer_connected.account import MyBMWAccount
from bimmer_connected.api.regions import Regions

async def main():
    account = MyBMWAccount(USERNAME, PASSWORD, Regions.REST_OF_WORLD)
    await account.get_vehicles()
    vehicle = account.get_vehicle(VIN)
    print(vehicle.brand, vehicle.name, vehicle.vin)

    result = await vehicle.remote_services.trigger_remote_light_flash()
    print(result.state)

asyncio.run(main())
```

### 2.2 Example in non-async code

```
import asyncio
from bimmer_connected.account import MyBMWAccount
from bimmer_connected.api.regions import Regions

account = MyBMWAccount(USERNAME, PASSWORD, Regions.REST_OF_WORLD)
asyncio.run(account.get_vehicles())
vehicle = account.get_vehicle(VIN)
print(vehicle.brand, vehicle.name, vehicle.vin)
```

(continues on next page)

(continued from previous page)

```
result = asyncio.run(vehicle.remote_services.trigger_remote_light_flash())  
print(result.state)
```

## COMPATIBILITY

This works with BMW (and Mini) vehicles with a MyBMW 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 MyBMW/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.



## DATA CONTRIBUTIONS

If some features do not work for your vehicle, we would need the data returned from 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.



## **CODE CONTRIBUTIONS**

Contributions are welcome! Please make sure that your code passes the checks in `.github/workflows/test.yml`. 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.





## 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. A special thanks to @HuChundong, @muxiachuixue, @vividmuse for figuring out how to solve login issues!

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>
- <https://github.com/TA2k/ioBroker.bmw>
- [https://gitee.com/ichuixue/bmw\\_shortcuts](https://gitee.com/ichuixue/bmw_shortcuts) / <https://www.icloud.com/shortcuts/eb064e89e6b647d2828a404227b91c4a>

Thank you for your great software!



## **LICENSE**

The `bimmer_connected` library is licensed under the Apache License 2.0.



## 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 MyBMW portal.

### 8.2 Installation

`bimmer_connected` is tested against **Python 3.8 or above**. Just install the latest release from [PyPI](#) using `pip3 install --upgrade bimmer_connected`.

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

---

**Note:** If you want to connect to a **chinese** server, you need to install the `[china]` extra, e.g. `pip3 install --upgrade bimmer_connected[china]`.

---

## 8.3 Usage

While this library is mainly written to be included in [Home Assistant](#), it can be use on its own.

After installation, execute `bimmerconnected` from command line for usage instruction or see the full [CLI documentation](#).

Please be aware that `bimmer_connected` is an async library when using it in Python code. The description of the modules can be found in the [module documentation](#).

### 8.3.1 Example in an asyncio event loop

```
import asyncio
from bimmer_connected.account import MyBMWAccount
from bimmer_connected.api.regions import Regions

async def main():
    account = MyBMWAccount(USERNAME, PASSWORD, Regions.REST_OF_WORLD)
    await account.get_vehicles()
    vehicle = account.get_vehicle(VIN)
    print(vehicle.brand, vehicle.name, vehicle.vin)

    result = await vehicle.remote_services.trigger_remote_light_flash()
    print(result.state)

asyncio.run(main())
```

### 8.3.2 Example in non-async code

```
import asyncio
from bimmer_connected.account import MyBMWAccount
from bimmer_connected.api.regions import Regions

account = MyBMWAccount(USERNAME, PASSWORD, Regions.REST_OF_WORLD)
asyncio.run(account.get_vehicles())
vehicle = account.get_vehicle(VIN)
print(vehicle.brand, vehicle.name, vehicle.vin)

result = asyncio.run(vehicle.remote_services.trigger_remote_light_flash())
print(result.state)
```

## 8.4 Compatibility

This works with BMW (and Mini) vehicles with a MyBMW 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 MyBMW/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 from 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 passes the checks in `.github/workflows/test.yml`. 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. A special thanks to @HuChundong, @muxiachuihue, @vividmuse for figuring out how to solve login issues!

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>
- <https://github.com/TA2k/ioBroker.bmw>
- [https://gitee.com/ichuihue/bmw\\_shortcuts](https://gitee.com/ichuihue/bmw_shortcuts) / <https://www.icloud.com/shortcuts/eb064e89e6b647d2828a404227b91c4a>

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 Using fingerprints in Home Assistant

Sometimes it can be useful to load the **fingerprints** used for our **pytest suite** in the development of the Home Assistant component. This enables debugging of the UI in Home Assistant which is not possible from `pytest` alone.

**Warning:** This is for the [Home Assistant development environment](#) only! Do not do this on your live instance!

Setup and start Home Assistant in the [development environment](#) at least once and let all python packages install (`hass -c ./config`). If not already done, set up the **BMW Connected Drive Integration**. You need to login a MyBMW account at least once. Shut down Homeassistant afterwards.



---

**Note:** The MyBMW account does not need to contain vehicles, a demo account without attached vehicles is sufficient.

---

Now, we have to “hack” our mocked backend calls into Home Assistant.

Edit `homeassistant/components/bmw_connected_drive/coordinator.py` and locate the function `def _async_update_data()`. We now have to replace `await self.account.get_vehicles()`. The `try .. except` block should look like this:

```
...
    try:
        from bimmer_connected.tests.conftest import MyBMWMockRouter, ALL_STATES, ALL_
↪CHARGING_SETTINGS
        with MyBMWMockRouter(["WBY000000000REXI01"], ALL_STATES, ALL_CHARGING_
↪SETTINGS):
            await self.account.get_vehicles()
    except:
...

```

As the first parameter, you can specify a list of VINs for debugging or leave it empty (`None` or `[]`) to load all vehicles of our test suite.

## 8.11 Reverse engineering the MyBMW API

This document should be seen as a help in setting up a working environment to intercept traffic of the MyBMW app. Not every step will be described fully, this guide is rather a summary and list for further reading. It will most likely need adjustments to your specific setup.

The MyBMW app is built with the Flutter framework and needs some additional persuasion to reveal the traffic.

### 8.11.1 Disclaimer

Note that we are actively disabling important security measures such as SSL/TLS encryption to understand which commands and messages are shared between the MyBMW app and the MyBMW servers.

Also note that there could always be changes to the API or the app itself made by BMW to stop us from understanding what is going on.

### 8.11.2 Acknowledgement

Most of this document would not exist without the amazing work of Jeroen Becker:

- [Intercepting traffic from Android Flutter applications \(ARMv7\)](#)
- [Intercepting Flutter traffic on Android \(ARMv8\)](#)
- [Intercepting Flutter traffic on iOS](#)

### 8.11.3 Software & hardware requirements

---

**Note:** This document is based on the MyBMW **Android** app. It should work similarly using **iPhones**. If possible, please create a PR with more details.

---

You will need:

- A proxy with MITM capabilities such as [mitmproxy](#)
- A **rooted** android phone with a version supported by MyBMW (currently Android 6.0 Marshmallow). It could also work using an [Android emulator](#).
- Access to your phone using ADB (via USB)
- [ProxyDroid](#) to forward all traffic to your proxy
- [Ghidra](#) to find the location to patch out SSL verification
- A python environment with [frida](#)
- [frida-android-helper](#) to help installing [frida](#) on your phone

### 8.11.4 Finding the location of SSL verification

The following steps are required if the location of the SSL verification function is not known. If it is, please continue with the [next section](#). For more details, please refer to [Jeroen Becker's work](#).

Get an APK/XAPK of the MyBMW app (from your phone or one of the many download sites). APK names include:

- `de.bmw.connected.mobile20.cn` (china)
- `de.bmw.connected.mobile20.na` (north america)
- `de.bmw.connected.mobile20.row` (rest of world)

Now extract `config.arm64-v8a.apk` or `config.armeabi-v7a.apk` from the APK package (depending of your phone's target architecture).

In Ghidra, load and analyze `lib/ARCH/libflutter.so`.

After analyze has finished, go to Search > For Scalar and search for value 390. Find `mov r3, #0x186` and jump to it.

Double click on function name on right side to get the hex address and first bytes of the function

Example: `2d e9 f0 4f a3 b0 81 46 50 20 10 70`

### 8.11.5 Preparations on phone

On your phone, add your custom CA certificates to the system store ([instructions for emulator](#), but works on **rooted** devies in similar fashion). This is required as the login screen is using the default Android WebView component, which again behaves differently from Flutter (or rather, behaves like expected).

Add your local proxy server to your Android system using ProxyDroid.

### 8.11.6 Disabling SSL verification with frida

Install & upgrade frida-tools & frida-android-helper (see [requirements](#)). Make sure that both are on the latest version.

Create a frida hook named hook\_flutter\_disable\_ssl.js with the following content. If needed, **replace the search pattern** and **disable adding 0x01 on ARMv8**.

```
function hook_ssl_verify_result(address)
{
  Interceptor.attach(address, {
    onEnter: function(args) {
      console.log("Disabling SSL validation")
    },
    onLeave: function(retval)
    {
      console.log("Retval: " + retval)
      retval.replace(0x1);
    }
  });
}

function disablePinning()
{
  var m = Process.findModuleByName("libflutter.so");
  var pattern = "2d e9 f0 4f a3 b0 81 46 50 20 10 70" // MyBMW 1.5.1 to 1.7.0 (all regions)

  var res = Memory.scan(m.base, m.size, pattern, {
    onMatch: function(address, size){
      console.log('[+] ssl_verify_result found at: ' + address.toString());

      // Add 0x01 because it's a THUMB function
      // Otherwise, we would get 'Error: unable to intercept function at 0x9906f8ac;
      ↪ please file a bug'
      // REQUIRED ON ARMv7 ONLY!!
      hook_ssl_verify_result(address.add(0x01));
    },
    onError: function(reason){
      console.log('[!] There was an error scanning memory');
    },
    onComplete: function()
    {
      console.log("All done")
    }
  });
}

setTimeout(disablePinning, 1000)
```

Connect to your phone via ADB with root permissions.

```
adb root && adb remount
```

Update & start frida server on the phone with frida-android-helper.

```
fah server update && fah server start
```

Start the MyBMW app from your computer via frida (adjust app identifier if needed).

```
frida -Uf de.bmw.connected.mobile20.row -l .\hook_flutter_disable_ssl.js --no-pause
```

Now you should be able to capture all traffic between your phone and the MyBMW API.

### 8.11.7 Using the information in bimmer\_connected

If you learn anything by capturing the traffic, please create [Issues/Feature Requests](#) or [Pull Requests](#) to our repository. Information that should be included contains:

- The URL of the endpoint
- HTTP headers of your request (**DO NOT** include **Cookie** or **Authentication** headers)
- The request payload (if available)
- The request response (if available)

If the data contains personal information, please do not delete it but replace it with random data.

**Warning:** Double check if all information is **sanitized** and no personal information or authentication data is included.

## 8.12 bimmerconnected

A simple executable to use and test the library.

```
usage: bimmerconnected [-h] [--debug]
                        {status,fingerprint,lightflash,horn,vehiclefinder,
↪ chargingsettings,chargingprofile,charge,image,sendpoi,sendpoi_from_address}
                        ...
```

### 8.12.1 Positional Arguments

<b>cmd</b>	Possible choices: status, fingerprint, lightflash, horn, vehiclefinder, chargingsettings, chargingprofile, charge, image, sendpoi, sendpoi_from_address
------------	---

### 8.12.2 Named Arguments

**--debug**                Print debug logs.  
Default: False

### 8.12.3 Sub-commands

#### status

Get the current status of the vehicle.

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

#### Positional Arguments

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

#### Named Arguments

<b>-j, --json</b>	Output as JSON only. Removes all other output. Default: False
<b>-v, --vin</b>	Output data for specified VIN only.

#### fingerprint

Save a vehicle fingerprint.

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

### Positional Arguments

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

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

### horn

Trigger the vehicle horn

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

### Positional Arguments

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

## vehiclefinder

Update the vehicle GPS location.

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

### Positional Arguments

<b>username</b>	Connected Drive username
<b>password</b>	Connected Drive password
<b>region</b>	Possible choices: north_america, china, rest_of_world Region of the Connected Drive account
<b>vin</b>	Vehicle Identification Number
<b>lat</b>	(optional) Your current GPS latitude (as float)
<b>lng</b>	(optional) Your current GPS longitude (as float)

## chargingsettings

Set vehicle charging settings.

```
bimmerconnected chargingsettings [-h] [--target-soc [TARGET_SOC]]
                                  [--ac-limit [AC_LIMIT]]
                                  username password
                                  {north_america,china,rest_of_world} vin
```

### Positional Arguments

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

### Named Arguments

<b>--target-soc</b>	Desired charging target SoC
<b>--ac-limit</b>	Maximum AC limit

### chargingprofile

Set vehicle charging profile.

```
bimmerconnected chargingprofile [-h]
                                [--charging-mode [{IMMEDIATE_CHARGING,DELAYED_CHARGING}]]
                                [--precondition-climate [PRECONDITION_CLIMATE]]
                                username password
                                {north_america,china,rest_of_world} vin
```

### Positional Arguments

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

### Named Arguments

<b>--charging-mode</b>	Possible choices: IMMEDIATE_CHARGING, DELAYED_CHARGING Desired charging mode
<b>--precondition-climate</b>	Precondition climate on charging windows

### charge

Start/stop charging on enabled vehicles.

```
bimmerconnected charge [-h]
                        username password {north_america,china,rest_of_world}
                        vin {start,stop}
```



## Positional Arguments

<b>username</b>	Connected Drive username
<b>password</b>	Connected Drive password
<b>region</b>	Possible choices: north_america, china, rest_of_world Region of the Connected Drive account
<b>vin</b>	Vehicle Identification Number
<b>action</b>	Possible choices: start, stop

## image

Download a vehicle image.

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

## Positional Arguments

<b>username</b>	Connected Drive username
<b>password</b>	Connected Drive password
<b>region</b>	Possible choices: north_america, china, rest_of_world Region of the Connected Drive account
<b>vin</b>	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

<b>username</b>	Connected Drive username
<b>password</b>	Connected Drive password
<b>region</b>	Possible choices: north_america, china, rest_of_world Region of the Connected Drive account
<b>vin</b>	Vehicle Identification Number
<b>latitude</b>	Latitude of the POI

<b>longitude</b>	Longitude of the POI
------------------	----------------------

### Named Arguments

<b>--name</b>	Name of the POI Default: “Sent with by bimmer_connected”
<b>--street</b>	(optional, display only) Street & House No. of the POI
<b>--city</b>	(optional, display only) City of the POI
<b>--postalcode</b>	(optional, display only) Postal code of the POI
<b>--country</b>	(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

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

### Named Arguments

<b>-n, --name</b>	(optional, display only) Name of the POI
<b>-a, --address</b>	Address (e.g. ‘Street 17, city, zip, country’)

## 8.13 bimmer\_connected.account

Access to a MyBMW account and all vehicles therein.

```
class bimmer_connected.account.MyBMWAccount(username: str, password: dataclasses.InitVar[str], region:  
                                             Regions, config: MyBMWClientConfiguration = None,  
                                             log_responses: dataclasses.InitVar[bool] = False,  
                                             observer_position: dataclasses.InitVar[GPSPosition] =  
                                             None, use_metric_units:  
                                             dataclasses.InitVar[Optional[bool]] = None)
```

Create a new connection to the MyBMW web service.

**async add\_vehicle**(*vehicle\_base: dict, fetched\_at: datetime | None = None*) → None

Add or update a vehicle from the API responses.

**config:** [MyBMWClientConfiguration](#) = None

Optional. If provided, username/password/region are ignored.

**property gcid:** str | None

Returns the current GCID.

**static get\_stored\_responses**() → List[[AnonymizedResponse](#)]

Return responses stored if log\_responses was set to True.

**get\_vehicle**(*vin: str*) → [MyBMWVehicle](#) | None

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 vehicle is found.

**async get\_vehicles**(*force\_init: bool = False*) → None

Retrieve vehicle data from BMW servers.

**log\_responses:** `dataclasses.InitVar[bool]` = False

Optional. If set, all responses from the server will be logged to this directory.

**observer\_position:** `dataclasses.InitVar[GPSPosition]` = None

Optional. Required for getting a position on older cars.

**password:** `dataclasses.InitVar[str]`

MyBMW password.

**property refresh\_token:** str | None

Returns the current refresh\_token.

**region:** [Regions](#)

Region of the account. See *api.Regions*.

**set\_observer\_position**(*latitude: float, longitude: float*) → None

Set the position of the observer for all vehicles.

**set\_refresh\_token**(*refresh\_token: str, gcid: str | None = None*) → None

Overwrite the current value of the MyBMW refresh token and GCID (if available).

**use\_metric\_units:** `dataclasses.InitVar[Optional[bool]]` = None

Deprecated. All returned values are metric units (km, l).

**username:** str

MyBMW user name (email) or 86-prefixed phone number (China only).

**vehicles:** List[[MyBMWVehicle](#)]

## 8.14 bimmer\_connected.api

The `bimmer_connected.api` module contains helper functions to communicate with the BMW APIs.

### 8.14.1 bimmer\_connected.api.authentication

Authentication management for BMW APIs.

```
class bimmer_connected.api.authentication.MyBMWAuthentication(username: str, password: str,  
                                                             region: Regions, access_token: str  
                                                             | None = None, expires_at:  
                                                             datetime | None = None,  
                                                             refresh_token: str | None = None,  
                                                             gcid: str | None = None)
```

Authentication and Retry Handler for MyBMW API.

**async** **async\_auth\_flow**(*request: Request*) → AsyncGenerator[Request, Response]

Execute the authentication flow asynchronously.

By default, this defers to `.auth_flow()`. You should override this method when the authentication scheme does I/O and/or uses concurrency primitives.

**async** **login**() → None

Get a valid OAuth token.

**property login\_lock: Lock**

Make sure that there is a lock in the current event loop.

**sync\_auth\_flow**(*request: Request*) → Generator[Request, Response, None]

Execute the authentication flow synchronously.

By default, this defers to `.auth_flow()`. You should override this method when the authentication scheme does I/O and/or uses concurrency primitives.

```
class bimmer_connected.api.authentication.MyBMWLoginClient(*args, **kwargs)
```

Async HTTP client based on `httpx.AsyncClient` with automated OAuth token refresh.

```
class bimmer_connected.api.authentication.MyBMWLoginRetry
```

`httpx.Auth` used as workaround to retry & sleep on 429 Too Many Requests.

**async** **async\_auth\_flow**(*request: Request*) → AsyncGenerator[Request, Response]

Execute the authentication flow asynchronously.

By default, this defers to `.auth_flow()`. You should override this method when the authentication scheme does I/O and/or uses concurrency primitives.

**sync\_auth\_flow**(*request: Request*) → Generator[Request, Response, None]

Execute the authentication flow synchronously.

By default, this defers to `.auth_flow()`. You should override this method when the authentication scheme does I/O and/or uses concurrency primitives.

**bimmer\_connected.api.authentication.get\_retry\_wait\_time**(*response: Response*) → int

Get the wait time for the next retry from the response and multiply by 2.

### 8.14.2 bimmer\_connected.api.client

Generic API management.

```
class bimmer_connected.api.client.MyBMWClient(config: MyBMWClientConfiguration, *args, brand: CarBrands | None = None, **kwargs)
```

Async HTTP client based on *httpx.AsyncClient* with automated OAuth token refresh.

```
generate_default_header(brand: CarBrands | None = None) → Dict[str, str]
```

Generate a header for HTTP requests to the server.

```
class bimmer_connected.api.client.MyBMWClientConfiguration(authentication: MyBMWAuthentication, log_responses: bool | None = False, observer_position: GPSPosition | None = None)
```

Stores global settings for MyBMWClient.

```
authentication: MyBMWAuthentication
```

```
log_responses: bool | None = False
```

```
observer_position: GPSPosition | None = None
```

```
set_log_responses(log_responses: bool) → None
```

Set if responses are logged and clear response store.

### 8.14.3 bimmer\_connected.api.regions

Get the right url for the different countries.

```
bimmer_connected.api.regions.get_app_version(region: Regions) → str
```

Get the app version & build number for the region.

```
bimmer_connected.api.regions.get_ocp_apim_key(region: Regions) → str
```

Get the authorization for OAuth settings.

```
bimmer_connected.api.regions.get_region_from_name(name: str) → Regions
```

Get a region for a string.

This function is not case-sensitive.

```
bimmer_connected.api.regions.get_server_url(region: Regions) → str
```

Get the url of the server for the region.

```
bimmer_connected.api.regions.get_user_agent(region: Regions) → str
```

Get the Dart user agent for the region.

```
bimmer_connected.api.regions.valid_regions() → List[str]
```

Get list of valid regions as strings.

### 8.14.4 `bimmer_connected.api.utils`

Utils for `bimmer_connected.api`.

`bimmer_connected.api.utils.anonymize_data(json_data: List | Dict) → List | Dict`

Replace parts of the logfiles containing personal information.

`bimmer_connected.api.utils.anonymize_response(response: Response) → AnonymizedResponse`

Anonymize a responses URL and content.

`bimmer_connected.api.utils.anonymize_vin(match: Match)`

Anonymize VINs but keep assignment.

`bimmer_connected.api.utils.create_s256_code_challenge(code_verifier: str) → str`

Create S256 code\_challenge with the given code\_verifier.

`bimmer_connected.api.utils.generate_cn_nonce(username: str) → str`

Generate a x-login-nonce string.

`bimmer_connected.api.utils.generate_random_base64_string(size: int) → str`

Generate a random base64 string with size.

`bimmer_connected.api.utils.generate_token(length: int = 30, chars: str = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_~') → str`

Generate a random token with given length and characters.

`bimmer_connected.api.utils.get_capture_position(base64_background_img: str) → str`

Get the position of the capture in the background image.

`bimmer_connected.api.utils.get_correlation_id() → Dict[str, str]`

Generate correlation headers.

`async bimmer_connected.api.utils.handle_httpstatuserror(ex: HTTPStatusError, module: str = 'API', log_handler: Logger | None = None, dont_raise: bool = False) → None`

Try to extract information from response and re-raise Exception.

`bimmer_connected.api.utils.try_import_pillow_image()`

Try to import PIL.Image and return if successful.

We only need to load PIL if we are in China, so we try to avoid a general dependency on Pillow for all users. Installing Pillow on Raspberry Pi (ARMv7) is painful.

## 8.15 `bimmer_connected.const`

URLs for different services and error code mapping.

`class bimmer_connected.const.CarBrands(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)`

Car brands supported by the MyBMW API.

`BMW = 'bmw'`

`MINI = 'mini'`

```
class bimmer_connected.const.Regions(value, names=None, *, module=None, qualname=None, type=None,
                                     start=1, boundary=None)
```

Regions of the world with separate servers.

```
CHINA = 'cn'
```

```
NORTH_AMERICA = 'na'
```

```
REST_OF_WORLD = 'row'
```

## 8.16 bimmer\_connected.models

Generals models used for bimmer\_connected.

```
class bimmer_connected.models.AnonymizedResponse(filename: str, content: List | Dict | str | None =
                                                  None)
```

An anonymized response.

```
content: List | Dict | str | None = None
```

```
filename: str
```

```
class bimmer_connected.models.ChargingSettings(chargingTarget: int | None, acLimitValue: int | None =
                                                  None)
```

Charging settings to control the vehicle.

```
acLimitValue: int | None = None
```

```
chargingTarget: int | None
```

```
dcLoudness = None
```

```
isUnlockCableActive = None
```

```
class bimmer_connected.models.GPSPosition(latitude: float | None, longitude: float | None)
```

GPS coordinates.

```
latitude: float | None
```

```
longitude: float | None
```

```
exception bimmer_connected.models.MyBMWAPIError
```

General BMW API error.

```
exception bimmer_connected.models.MyBMWAuthError
```

Auth-related error from BMW API (HTTP status codes 401 and 403).

```
exception bimmer_connected.models.MyBMWQuotaError
```

Quota exceeded on BMW API.

```
exception bimmer_connected.models.MyBMWRemoteServiceError
```

Error when executing remote services.

```
class bimmer_connected.models.PointOfInterest(lat: dataclasses.InitVar[float], lon:
                                             dataclasses.InitVar[float], name: str | None = 'Sent with
                                             by bimmer_connected', street: dataclasses.InitVar[str] =
                                             None, postal_code: dataclasses.InitVar[str] = None,
                                             city: dataclasses.InitVar[str] = None, country:
                                             dataclasses.InitVar[str] = None, formattedAddress: str |
                                             None = None, address: str | None = None,
                                             baseCategoryId: str | None = None, phoneNumber: str |
                                             None = None, provider: str | None = None, providerId:
                                             str | None = None, providerPoiId: str = '', sourceType:
                                             str | None = None, type: str | None = None,
                                             vehicleCategoryId: str | None = None)
```

A Point of Interest to be sent to the car.

```
address: str | None = None

baseCategoryId: str | None = None

city: dataclasses.InitVar[str] = None

coordinates: GPSPosition

country: dataclasses.InitVar[str] = None

entryPoints: List

formattedAddress: str | None = None

lat: dataclasses.InitVar[float]

locationAddress: PointOfInterestAddress | None

lon: dataclasses.InitVar[float]

name: str | None = 'Sent with by bimmer_connected'

phoneNumber: str | None = None

postal_code: dataclasses.InitVar[str] = None

provider: str | None = None

providerId: str | None = None

providerPoiId: str = ''

sourceType: str | None = None

street: dataclasses.InitVar[str] = None

type: str | None = None

vehicleCategoryId: str | None = None
```



```
class bimmer_connected.models.PointOfInterestAddress(street: str | None = None, postalCode: str |
None = None, city: str | None = None, country:
str | None = None, banchi: str | None = None,
chome: str | None = None, countryCode: str |
None = None, district: str | None = None, go:
str | None = None, houseNumber: str | None =
None, region: str | None = None, regionCode:
str | None = None, settlement: str | None =
None)
```

Address data of a PointOfInterest.

```
banchi: str | None = None
chome: str | None = None
city: str | None = None
country: str | None = None
countryCode: str | None = None
district: str | None = None
go: str | None = None
houseNumber: str | None = None
postalCode: str | None = None
region: str | None = None
regionCode: str | None = None
settlement: str | None = None
street: str | None = None
```

```
class bimmer_connected.models.StrEnum(value, names=None, *, module=None, qualname=None,
type=None, start=1, boundary=None)
```

A string enumeration of type (*str*; *Enum*). All members are compared via *upper()*. Defaults to UNKNOWN.

```
class bimmer_connected.models.ValueWithUnit(value: int | float | None, unit: str | None)
```

A value with a corresponding unit.

```
unit: str | None
    Alias for field number 1
value: int | float | None
    Alias for field number 0
```

```
class bimmer_connected.models.VehicleDataBase
```

A base class for parsing and storing complex vehicle data.

```
classmethod from_vehicle_data(vehicle_data: Dict)
```

Create the class based on vehicle data from API.

```
update_from_vehicle_data(vehicle_data: Dict)
```

Update the attributes based on vehicle data from API.

## 8.17 bimmer\_connected.utils

General utils and base classes used in the library.

```
class bimmer_connected.utils.MyBMWJSONEncoder(*, skipkeys=False, ensure_ascii=True,  
                                              check_circular=True, allow_nan=True,  
                                              sort_keys=False, indent=None, separators=None,  
                                              default=None)
```

JSON Encoder that handles data classes, properties and additional data types.

**default**(o) → str | dict

Implement this method in a subclass such that it returns a serializable object for o, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):  
    try:  
        iterable = iter(o)  
    except TypeError:  
        pass  
    else:  
        return list(iterable)  
    # Let the base class default method raise the TypeError  
    return JSONEncoder.default(self, o)
```

**bimmer\_connected.utils.get\_class\_property\_names**(obj: object)

Return the names of all properties of a class.

**bimmer\_connected.utils.log\_response\_store\_to\_file**(response\_store: List[AnonymizedResponse],  
 logfile\_path: Path) → None

Log all responses to files.

**bimmer\_connected.utils.parse\_datetime**(date\_str: str) → datetime | None

Convert a time string into datetime.

**bimmer\_connected.utils.to\_camel\_case**(input\_str: str) → str

Convert SNAKE\_CASE or snake\_case to camelCase.

## 8.18 bimmer\_connected.vehicle

The `bimmer_connected.vehicle` module contains all data & parsers for a vehicle.

### 8.18.1 bimmer\_connected.vehicle.vehicle

Models state and remote services of one vehicle.

```
class bimmer_connected.vehicle.vehicle.LscType(value, names=None, *, module=None,  
                                              qualname=None, type=None, start=1,  
                                              boundary=None)
```

Known Values for `lsc_type` field.

Not really sure, what this value really contains.

ACTIVATED = 'ACTIVATED'

NOT\_CAPABLE = 'NOT\_CAPABLE'

NOT\_SUPPORTED = 'NOT\_SUPPORTED'

UNKNOWN = 'UNKNOWN'

**class** bimmer\_connected.vehicle.vehicle.**MyBMWVehicle**(*account: MyBMWAccount, vehicle\_base: dict, fetched\_at: datetime | None = None*)

Models state and remote services of one vehicle.

#### Parameters

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

**property available\_attributes: List[str]**

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

**property brand: CarBrands**

Get the car brand.

**combine\_data**(*data: Dict | List[Dict], fetched\_at: datetime | None = None*) → Dict

Combine API responses and additional information to a single dictionary.

**property drive\_train: DriveTrainType**

Get the type of drive train of the vehicle.

**property drive\_train\_attributes: List[str]**

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.

**async get\_vehicle\_image**(*direction: VehicleViewDirection*) → bytes

Get a rendered image of the vehicle.

:returns bytes containing the image in PNG format.

**async get\_vehicle\_state**() → None

Retrieve vehicle data from BMW servers.

**property has\_combustion\_drivetrain: bool**

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

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

**property has\_electric\_drivetrain: bool**

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.

**property is\_charging\_plan\_supported: bool**

Return True if charging profile is available and can be set via API.

**property is\_charging\_settings\_supported: bool**

Return True if charging settings can be set via API.

**property is\_lsc\_enabled: bool**

Return True if LastStateCall is enabled (vehicle automatically updates API).

**property is\_remote\_charge\_start\_enabled: bool**

Return True if charging can be started via the API.

**property is\_remote\_charge\_stop\_enabled: bool**

Return True if charging can be stop via the API.

**property is\_remote\_climate\_start\_enabled: bool**

Return True if AC/ventilation can be started via the API.

**property is\_remote\_climate\_stop\_enabled: bool**

Return True if AC/ventilation can be stopped via the API.

**property is\_remote\_horn\_enabled: bool**

Return True if the horn can be activated via the API.

**property is\_remote\_lights\_enabled: bool**

Return True if the lights can be activated via the API.

**property is\_remote\_lock\_enabled: bool**

Return True if vehicle can be locked via the API.

**property is\_remote\_sendpoi\_enabled: bool**

Return True if POIs can be set via the API.

**property is\_remote\_set\_ac\_limit\_enabled: bool**

Return True if AC limit can be set via the API.

**property is\_remote\_set\_target\_soc\_enabled: bool**

Return True if Target SoC can be set via the API.

**property is\_remote\_unlock\_enabled: bool**

Return True if POIs can be unlocked via the API.

**property is\_vehicle\_active: bool**

Deprecated, always returns False.

Check if the vehicle is active/moving.

If the vehicle was active/moving at the time of the last status update, current position is not available.

**property is\_vehicle\_tracking\_enabled: bool**

Return True if vehicle finder is enabled in vehicle.

**property lsc\_type: *LscType***

Get the lscType of the vehicle.

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

**property mileage: *ValueWithUnit***

Get the mileage of the vehicle.

**property name: str**

Get the name of the vehicle.

**property timestamp: datetime | None**

Get the timestamp when the data was recorded.

**update\_state(data: Dict | List[Dict], fetched\_at: datetime | None = None) → None**

Update the state of a vehicle.

**property vin: str**

Get the VIN (vehicle identification number) of the vehicle.

**class bimmer\_connected.vehicle.vehicle.VehicleViewDirection**(value, names=None, \*, module=None, qualname=None, type=None, start=1, boundary=None)

Viewing angles for the vehicle.

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

**FRONT = 'FrontView'**

**FRONTSIDE = 'AngleSideViewForty'**

**SIDE = 'SideViewLeft'**

**UNKNOWN = 'UNKNOWN'**

## 8.18.2 bimmer\_connected.vehicle.remote\_services

Trigger remote services on a vehicle.

**class bimmer\_connected.vehicle.remote\_services.ExecutionState**(value, names=None, \*, module=None, qualname=None, type=None, start=1, boundary=None)

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

**DELIVERED = 'DELIVERED'**

**ERROR = 'ERROR'**

**EXECUTED = 'EXECUTED'**

**IGNORED = 'IGNORED'**

**INITIATED = 'INITIATED'**

**PENDING = 'PENDING'**

**UNKNOWN = 'UNKNOWN'**

**class bimmer\_connected.vehicle.remote\_services.RemoteServiceStatus**(response: dict, event\_id: str | None = None)

Wraps the status of the execution of a remote service.

**class bimmer\_connected.vehicle.remote\_services.RemoteServices**(vehicle: [MyBMWVehicle](#))

Trigger remote services on a vehicle.

**async trigger\_charge\_start()** → [RemoteServiceStatus](#)

Trigger the vehicle to start charging.

**async trigger\_charge\_stop()** → *RemoteServiceStatus*

Trigger the vehicle to stop charging.

**async trigger\_charging\_profile\_update**(charging\_mode: *ChargingMode* | *None* = *None*,  
precondition\_climate: *bool* | *None* = *None*) →  
*RemoteServiceStatus*

Update the charging profile on the vehicle.

**async trigger\_charging\_settings\_update**(target\_soc: *int* | *None* = *None*, ac\_limit: *int* | *None* = *None*)  
→ *RemoteServiceStatus*

Update the charging settings on the vehicle.

**async trigger\_remote\_air\_conditioning()** → *RemoteServiceStatus*

Trigger the air conditioning to start.

**async trigger\_remote\_air\_conditioning\_stop()** → *RemoteServiceStatus*

Trigger the air conditioning to stop.

**async trigger\_remote\_door\_lock()** → *RemoteServiceStatus*

Trigger the vehicle to lock its doors.

**async trigger\_remote\_door\_unlock()** → *RemoteServiceStatus*

Trigger the vehicle to unlock its doors.

**async trigger\_remote\_horn()** → *RemoteServiceStatus*

Trigger the vehicle to sound its horn.

**async trigger\_remote\_light\_flash()** → *RemoteServiceStatus*

Trigger the vehicle to flash its headlights.

**async trigger\_remote\_service**(service\_id: *Services*, params: *Dict* | *None* = *None*, data: *Any* = *None*,  
refresh: *bool* = *False*) → *RemoteServiceStatus*

Trigger a remote service and wait for the result.

**async trigger\_remote\_vehicle\_finder()** → *RemoteServiceStatus*

Trigger the vehicle finder.

**async trigger\_send\_poi**(poi: *PointOfInterest* | *Dict*) → *RemoteServiceStatus*

Send a PointOfInterest to the vehicle.

#### Parameters

**poi** – A PointOfInterest containing at least ‘lat’ and ‘lon’ and optionally ‘name’, ‘street’,  
‘city’, ‘postalCode’, ‘country’

**class bimmer\_connected.vehicle.remote\_services.Services**(value, names=*None*, \*, module=*None*,  
qualname=*None*, type=*None*, start=*1*,  
boundary=*None*)

Enumeration of possible services to be executed.

**AIR\_CONDITIONING** = 'climate-now'

**CHARGE\_START** = 'start-charging'

**CHARGE\_STOP** = 'stop-charging'

**CHARGING\_PROFILE** = 'CHARGING\_PROFILE'

**CHARGING\_SETTINGS** = 'CHARGING\_SETTINGS'

```

DOOR_LOCK = 'door-lock'

DOOR_UNLOCK = 'door-unlock'

HORN = 'horn-blow'

LIGHT_FLASH = 'light-flash'

SEND_POI = 'SEND_POI'

VEHICLE_FINDER = 'vehicle-finder'

```

### 8.18.3 bimmer\_connected.vehicle.charging\_profile

Models the charging profiles of a vehicle.

```

class bimmer_connected.vehicle.charging_profile.ChargingMode(value, names=None, *,
                                                             module=None, qualname=None,
                                                             type=None, start=1,
                                                             boundary=None)

```

Charging mode of electric vehicle.

```

DELAYED_CHARGING = 'DELAYED_CHARGING'

IMMEDIATE_CHARGING = 'IMMEDIATE_CHARGING'

UNKNOWN = 'UNKNOWN'

```

```

class bimmer_connected.vehicle.charging_profile.ChargingPreferences(value, names=None, *,
                                                                    module=None,
                                                                    qualname=None,
                                                                    type=None, start=1,
                                                                    boundary=None)

```

Charging preferences of electric vehicle.

```

CHARGING_WINDOW = 'CHARGING_WINDOW'

NO_PRESELECTION = 'NO_PRESELECTION'

UNKNOWN = 'UNKNOWN'

```

```

class bimmer_connected.vehicle.charging_profile.ChargingProfile(is_pre_entry_climatization_enabled:
                                                                bool, timer_type: TimerTypes,
                                                                departure_times:
                                                                List[DepartureTimer],
                                                                preferred_charging_window:
                                                                ChargingWindow,
                                                                charging_preferences:
                                                                ChargingPreferences,
                                                                charging_mode: ChargingMode,
                                                                ac_current_limit: int | None =
                                                                None, ac_available_limits: list |
                                                                None = None, charging_preferences_service_pack:
                                                                str | None = None)

```

Models the charging profile of a vehicle.

**ac\_available\_limits:** list | None = None

Available AC limits to be selected.

**ac\_current\_limit:** int | None = None

Returns the ac current limit.

**charging\_mode:** [ChargingMode](#)

Returns the preferred charging mode.

**charging\_preferences:** [ChargingPreferences](#)

Returns the preferred charging preferences.

**charging\_preferences\_service\_pack:** str | None = None

Service Pack required for remote service format.

**departure\_times:** List[[DepartureTimer](#)]

List of timers.

**format\_for\_remote\_service()** → dict

Format current charging profile as base to be sent to remote service.

**is\_pre\_entry\_climatization\_enabled:** bool

Get status of pre-entry climatization.

**preferred\_charging\_window:** [ChargingWindow](#)

Returns the preferred charging window.

**timer\_type:** [TimerTypes](#)

Returns the current timer plan type.

**class** bimmer\_connected.vehicle.charging\_profile.[ChargingWindow](#)(window\_dict: dict)

A charging window.

**property** end\_time: time

End of the charging window.

**property** start\_time: time

Start of the charging window.

**class** bimmer\_connected.vehicle.charging\_profile.[DepartureTimer](#)(timer\_dict: dict)

A departure timer.

**property** action: str | None

What does the timer do.

**property** start\_time: time | None

Departure time for this timer.

**property** timer\_id: int | None

ID of this timer.

**property** weekdays: List[str]

Active weekdays for this timer.

**class** bimmer\_connected.vehicle.charging\_profile.[TimerTypes](#)(value, names=None, \*, module=None, qualname=None, type=None, start=1, boundary=None)

Different timer types.



```
TWO_TIMES_TIMER = 'TWO_TIMES_TIMER'
```

```
UNKNOWN = 'UNKNOWN'
```

```
WEEKLY_PLANNER = 'WEEKLY_PLANNER'
```

#### 8.18.4 bimmer\_connected.vehicle.doors\_windows

Models the state of a vehicle.

```
class bimmer_connected.vehicle.doors_windows.DoorsAndWindows(door_lock_state: ~bimmer_connected.vehicle.doors_windows.LockState
    = LockState.UNKNOWN, lids: ~typing.List[~bimmer_connected.vehicle.doors_windows.Lid]
    = <factory>, windows: ~typing.List[~bimmer_connected.vehicle.doors_windows.Window]
    = <factory>)
```

Provides an accessible version of *properties.doorsAndWindows*.

```
property all_lids_closed: bool
```

Check if all lids are closed.

```
property all_windows_closed: bool
```

Check if all windows are closed.

```
door_lock_state: LockState = 'UNKNOWN'
```

Get state of the door locks.

```
lids: List[Lid]
```

All lids (doors+hood+trunk) of the car.

```
property open_lids: List[Lid]
```

Get all open lids of the car.

```
property open_windows: List[Window]
```

Get all open windows of the car.

```
windows: List[Window]
```

All windows (doors+sunroof) of the car.

```
class bimmer_connected.vehicle.doors_windows.Lid(name: str, state: str)
```

A lid of the vehicle.

Lids are: Doors + Trunk + Hatch

```
property is_closed: bool
```

Check if the lid is closed.

```
name
```

name of the lid

```
class bimmer_connected.vehicle.doors_windows.LidState(value, names=None, *, module=None,
    qualname=None, type=None, start=1,
    boundary=None)
```

Possible states of the hatch, trunk, doors, windows, sun roof.

```
CLOSED = 'CLOSED'
INTERMEDIATE = 'INTERMEDIATE'
INVALID = 'INVALID'
OPEN = 'OPEN'
OPEN_TILT = 'OPEN_TILT'
UNKNOWN = 'UNKNOWN'
```

```
class bimmer_connected.vehicle.doors_windows.LockState(value, names=None, *, module=None,
                                                         qualname=None, type=None, start=1,
                                                         boundary=None)
```

Possible states of the door locks.

```
LOCKED = 'LOCKED'
PARTIALLY_LOCKED = 'PARTIALLY_LOCKED'
SECURED = 'SECURED'
SELECTIVE_LOCKED = 'SELECTIVE_LOCKED'
UNKNOWN = 'UNKNOWN'
UNLOCKED = 'UNLOCKED'
```

```
class bimmer_connected.vehicle.doors_windows.Window(name: str, state: str)
```

A window of the vehicle.

A window can be a normal window of the car or the sun roof.

### 8.18.5 bimmer\_connected.vehicle.fuel\_and\_battery

Generals models used for bimmer\_connected.

```
class bimmer_connected.vehicle.fuel_and_battery.ChargingState(value, names=None, *,
                                                                module=None, qualname=None,
                                                                type=None, start=1,
                                                                boundary=None)
```

Charging state of electric vehicle.

```
CHARGING = 'CHARGING'
COMPLETE = 'COMPLETE'
DEFAULT = 'DEFAULT'
ERROR = 'ERROR'
FINISHED_FULLY_CHARGED = 'FINISHED_FULLY_CHARGED'
FINISHED_NOT_FULL = 'FINISHED_NOT_FULL'
FULLY_CHARGED = 'FULLY_CHARGED'
```

```

INVALID = 'INVALID'

NOT_CHARGING = 'NOT_CHARGING'

PLUGGED_IN = 'PLUGGED_IN'

TARGET_REACHED = 'TARGET_REACHED'

UNKNOWN = 'UNKNOWN'

WAITING_FOR_CHARGING = 'WAITING_FOR_CHARGING'

```

```

class bimmer_connected.vehicle.fuel_and_battery.FuelAndBattery(remaining_range_fuel:
    ValueWithUnit | None = (None,
    None), remaining_range_electric:
    ValueWithUnit | None = (None,
    None), remaining_range_total:
    ValueWithUnit | None = (None,
    None), remaining_fuel:
    ValueWithUnit | None = (None,
    None), remaining_fuel_percent:
    int | None = None,
    remaining_battery_percent: int |
    None = None, charging_status:
    ChargingState | None = None,
    charging_start_time: datetime |
    None = None, charging_end_time:
    datetime | None = None,
    is_charger_connected: bool =
    False, charging_target: int | None
    = None)

```

Provides an accessible version of *status.FuelAndBattery*.

**charging\_end\_time:** `datetime | None = None`

The estimated time the vehicle will have finished charging.

**charging\_start\_time:** `datetime | None = None`

The planned time the vehicle will start charging in UTC.

**charging\_status:** `ChargingState | None = None`

Charging state of the vehicle.

**charging\_target:** `int | None = None`

State of charging target in percent.

**classmethod from\_vehicle\_data**(*vehicle\_data: Dict*)

Create the class based on vehicle data from API.

**is\_charger\_connected:** `bool = False`

Get status of the connection

**remaining\_battery\_percent:** `int | None = None`

State of charge of the high voltage battery in percent.

**remaining\_fuel:** `ValueWithUnit | None = (None, None)`

Get the remaining fuel of the vehicle.

**remaining\_fuel\_percent:** `int | None = None`

State of charge of the high voltage battery in percent.

**remaining\_range\_electric:** `ValueWithUnit | None = (None, None)`

Get the remaining range of the vehicle on electricity.

**remaining\_range\_fuel:** `ValueWithUnit | None = (None, None)`

Get the remaining range of the vehicle on fuel.

**remaining\_range\_total:** `ValueWithUnit | None = (None, None)`

Get the total remaining range of the vehicle (fuel + electricity, if available).

### 8.18.6 `bimmer_connected.vehicle.location`

Generals models used for `bimmer_connected`.

```
class bimmer_connected.vehicle.location.VehicleLocation(location: GPSPosition | None = None,
                                                         heading: int | None = None,
                                                         vehicle_update_timestamp: datetime | None
                                                         = None, account_region: Regions | None =
                                                         None, remote_service_position: Dict |
                                                         None = None)
```

The current position of a vehicle.

**account\_region:** `Regions | None = None`

**classmethod from\_vehicle\_data**(*vehicle\_data: Dict*)

Create the class based on vehicle data from API.

**heading:** `int | None = None`

The last known heading/direction of the vehicle.

**location:** `GPSPosition | None = None`

The last known position of the vehicle.

**remote\_service\_position:** `Dict | None = None`

**set\_remote\_service\_position**(*remote\_service\_dict: Dict*)

Store remote service position returned from vehicle finder service.

**vehicle\_update\_timestamp:** `datetime | None = None`

### 8.18.7 `bimmer_connected.vehicle.reports`

Models the state of a vehicle.

```
class bimmer_connected.vehicle.reports.CheckControlMessage(description_short: str,
                                                            description_long: str | None, state:
                                                            CheckControlStatus)
```

Check control message sent from the server.

**description\_long:** `str | None`

**description\_short:** `str`

**classmethod from\_api\_entry**(*type: str, severity: str, longDescription: str | None = None, \*\*kwargs*)

Parse a check control entry from the API format to *CheckControlMessage*.

**state:** *CheckControlStatus*

**class** *bimmer\_connected.vehicle.reports.CheckControlMessageReport*(*messages: ~typing.List[~bimmer\_connected.vehicle.reports.CheckControlMessage] = <factory>, has\_check\_control\_messages: bool = False*)

Parse and summarizes check control messages (e.g. low tire pressure).

**has\_check\_control\_messages:** *bool = False*

Indicate if check control messages are present.

**messages:** *List[CheckControlMessage]*

List of check control messages.

**class** *bimmer\_connected.vehicle.reports.CheckControlStatus*(*value, names=None, \*, module=None, qualname=None, type=None, start=1, boundary=None*)

Status of the condition based services.

**CRITICAL** = 'CRITICAL'

**HIGH** = 'HIGH'

**LOW** = 'LOW'

**MEDIUM** = 'MEDIUM'

**OK** = 'OK'

**UNKNOWN** = 'UNKNOWN'

**class** *bimmer\_connected.vehicle.reports.ConditionBasedService*(*service\_type: str, state: ConditionBasedServiceStatus, due\_date: datetime | None, due\_distance: ValueWithUnit*)

Entry in the list of condition based services.

**due\_date:** *datetime | None*

**due\_distance:** *ValueWithUnit*

**classmethod from\_api\_entry**(*type: str, status: str, dateTime: str | None = None, mileage: int | None = None, \*\*kwargs*)

Parse a condition based service entry from the API format to *ConditionBasedService*.

**service\_type:** *str*

**state:** *ConditionBasedServiceStatus*

**class** *bimmer\_connected.vehicle.reports.ConditionBasedServiceReport*(*messages: ~typing.List[~bimmer\_connected.vehicle.reports.ConditionBasedService] = <factory>, is\_service\_required: bool = False*)

Parse and summarizes condition based services (e.g. next oil service).

**is\_service\_required:** **bool** = **False**

Indicate if a service is required.

**messages:** **List**[[\*ConditionBasedService\*](#)]

List of the condition based services.

```
class bimmer_connected.vehicle.reports.ConditionBasedServiceStatus(value, names=None, *,
                                                                    module=None,
                                                                    qualname=None,
                                                                    type=None, start=1,
                                                                    boundary=None)
```

Status of the condition based services.

**OK** = 'OK'

**OVERDUE** = 'OVERDUE'

**PENDING** = 'PENDING'

**UNKNOWN** = 'UNKNOWN'

```
class bimmer_connected.vehicle.reports.Headunit(idrive_version: str = "", headunit_type: str = "",
                                                  software_version: str = "")
```

Parse and summarizes headunit hard/software versions.

**headunit\_type:** **str** = ''

Type of headunit.

**idrive\_version:** **str** = ''

IDRIVE generation.

**software\_version:** **str** = ''

Current software revision of vehicle

## INDICES AND TABLES

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





## PYTHON MODULE INDEX

### b

- `bimmer_connected.account`, [30](#)
- `bimmer_connected.api.authentication`, [32](#)
- `bimmer_connected.api.client`, [33](#)
- `bimmer_connected.api.regions`, [33](#)
- `bimmer_connected.api.utils`, [34](#)
- `bimmer_connected.const`, [34](#)
- `bimmer_connected.models`, [35](#)
- `bimmer_connected.utils`, [38](#)
- `bimmer_connected.vehicle.charging_profile`, [43](#)
- `bimmer_connected.vehicle.doors_windows`, [45](#)
- `bimmer_connected.vehicle.fuel_and_battery`, [46](#)
- `bimmer_connected.vehicle.location`, [48](#)
- `bimmer_connected.vehicle.remote_services`, [41](#)
- `bimmer_connected.vehicle.reports`, [48](#)
- `bimmer_connected.vehicle.vehicle`, [38](#)



## INDEX

### A

**ac\_available\_limits** (bimmer\_connected.vehicle.charging\_profile.ChargingProfile attribute), 43  
**ac\_current\_limit** (bimmer\_connected.vehicle.charging\_profile.ChargingProfile attribute), 44  
**account\_region** (bimmer\_connected.vehicle.location.VehicleLocation attribute), 48  
**acLimitValue** (bimmer\_connected.models.ChargingSettings attribute), 35  
**action** (bimmer\_connected.vehicle.charging\_profile.DepartureTimer attribute), 44  
**ACTIVATED** (bimmer\_connected.vehicle.vehicle.LscType attribute), 38  
**add\_vehicle()** (bimmer\_connected.account.MyBMWAccount method), 30  
**address** (bimmer\_connected.models.PointOfInterest attribute), 36  
**AIR\_CONDITIONING** (bimmer\_connected.vehicle.remote\_services.Services attribute), 42  
**all\_lids\_closed** (bimmer\_connected.vehicle.doors\_windows.DoorsAndWindows attribute), 45  
**all\_windows\_closed** (bimmer\_connected.vehicle.doors\_windows.DoorsAndWindows attribute), 45  
**anonymize\_data()** (in module bimmer\_connected.api.utils), 34  
**anonymize\_response()** (in module bimmer\_connected.api.utils), 34  
**anonymize\_vin()** (in module bimmer\_connected.api.utils), 34  
**AnonymizedResponse** (class in bimmer\_connected.models), 35  
**async\_auth\_flow()** (bimmer\_connected.api.authentication.MyBMWAuthentication method), 32  
**async\_auth\_flow()** (bimmer\_connected.api.authentication.MyBMWLoginRetry method), 32

method), 32

authentication

(bim-

mer\_connected.api.client.MyBMWClientConfiguration attribute), 33

available\_attributes

(bim-

mer\_connected.vehicle.vehicle.MyBMWVehicle property), 39

### B

**banchi** (bimmer\_connected.models.PointOfInterestAddress attribute), 37

**baseCategoryId**

(bim-

mer\_connected.models.PointOfInterest attribute), 36

**bimmer\_connected.account**  
module, 30

**bimmer\_connected.api.authentication**  
module, 32

**bimmer\_connected.api.client**  
module, 33

**bimmer\_connected.api.regions**  
module, 33

**bimmer\_connected.api.utils**  
module, 34

**bimmer\_connected.const**  
module, 34

**bimmer\_connected.models**  
module, 35

**bimmer\_connected.utils**  
module, 38

**bimmer\_connected.vehicle.charging\_profile**  
module, 43

**bimmer\_connected.vehicle.doors\_windows**  
module, 45

**bimmer\_connected.vehicle.fuel\_and\_battery**  
module, 46

**bimmer\_connected.vehicle.location**  
module, 48

**bimmer\_connected.vehicle.remote\_services**  
module, 41

**bimmer\_connected.vehicle.reports**  
module, 48

bimmer_connected.vehicle.vehicle module, 38	ChargingState (class in bimmer_connected.vehicle.fuel_and_battery), 46
BMW (bimmer_connected.const.CarBrands attribute), 34	chargingTarget (bimmer_connected.models.ChargingSettings attribute), 35
brand (bimmer_connected.vehicle.vehicle.MyBMWVehicle property), 39	ChargingWindow (class in bimmer_connected.vehicle.charging_profile), 44
<b>C</b>	CheckControlMessage (class in bimmer_connected.vehicle.reports), 48
CarBrands (class in bimmer_connected.const), 34	CheckControlMessageReport (class in bimmer_connected.vehicle.reports), 49
CHARGE_START (bimmer_connected.vehicle.remote_services.Services attribute), 42	CheckControlStatus (class in bimmer_connected.vehicle.reports), 49
CHARGE_STOP (bimmer_connected.vehicle.remote_services.Services attribute), 42	CHINA (bimmer_connected.const.Regions attribute), 35
CHARGING (bimmer_connected.vehicle.fuel_and_battery.ChargingState attribute), 46	chome (bimmer_connected.models.PointOfInterestAddress attribute), 37
charging_end_time (bimmer_connected.vehicle.fuel_and_battery.FuelAndBattery attribute), 47	city (bimmer_connected.models.PointOfInterest attribute), 36
charging_mode (bimmer_connected.vehicle.charging_profile.ChargingProfile attribute), 44	city (bimmer_connected.models.PointOfInterestAddress attribute), 37
charging_preferences (bimmer_connected.vehicle.charging_profile.ChargingProfile attribute), 44	CLOSED (bimmer_connected.vehicle.doors_windows.LidState attribute), 45
charging_preferences_service_pack (bimmer_connected.vehicle.charging_profile.ChargingProfile attribute), 44	combine_data() (bimmer_connected.vehicle.vehicle.MyBMWVehicle method), 39
CHARGING_PROFILE (bimmer_connected.vehicle.remote_services.Services attribute), 42	COMPLETE (bimmer_connected.vehicle.fuel_and_battery.ChargingState attribute), 46
CHARGING_SETTINGS (bimmer_connected.vehicle.remote_services.Services attribute), 42	ConditionBasedService (class in bimmer_connected.vehicle.reports), 49
charging_start_time (bimmer_connected.vehicle.fuel_and_battery.FuelAndBattery attribute), 47	ConditionBasedServiceReport (class in bimmer_connected.vehicle.reports), 49
charging_status (bimmer_connected.vehicle.fuel_and_battery.FuelAndBattery attribute), 47	ConditionBasedServiceStatus (class in bimmer_connected.vehicle.reports), 50
charging_target (bimmer_connected.vehicle.fuel_and_battery.FuelAndBattery attribute), 47	config (bimmer_connected.account.MyBMWAccount attribute), 31
CHARGING_WINDOW (bimmer_connected.vehicle.charging_profile.ChargingProfile attribute), 43	content (bimmer_connected.models.AnonymizedResponse attribute), 35
ChargingMode (class in bimmer_connected.vehicle.charging_profile), 43	coordinates (bimmer_connected.models.PointOfInterest attribute), 36
ChargingPreferences (class in bimmer_connected.vehicle.charging_profile), 43	country (bimmer_connected.models.PointOfInterest attribute), 36
ChargingProfile (class in bimmer_connected.vehicle.charging_profile), 43	country (bimmer_connected.models.PointOfInterestAddress attribute), 37
ChargingSettings (class in bimmer_connected.models), 35	countryCode (bimmer_connected.models.PointOfInterestAddress attribute), 37
	create_s256_code_challenge() (in module bimmer_connected.api.utils), 34
	CRITICAL (bimmer_connected.vehicle.reports.CheckControlStatus attribute), 49
	<b>D</b>
	dcLoudness (bimmer_connected.models.ChargingSettings attribute), 35

attribute), 35  
 DEFAULT (bimmer\_connected.vehicle.fuel\_and\_battery.ChargingState attribute), 41  
 attribute), 46  
 default() (bimmer\_connected.utils.MyBMWJSONEncoder method), 38  
 DELAYED\_CHARGING (bimmer\_connected.vehicle.charging\_profile.ChargingMode attribute), 43  
 DELIVERED (bimmer\_connected.vehicle.remote\_services.ExecutionState attribute), 35  
 attribute), 41  
 departure\_times (bimmer\_connected.vehicle.charging\_profile.ChargingProfile attribute), 44  
 DepartureTimer (class in bimmer\_connected.vehicle.charging\_profile), 44  
 description\_long (bimmer\_connected.vehicle.reports.CheckControlMessage attribute), 48  
 description\_short (bimmer\_connected.vehicle.reports.CheckControlMessage attribute), 48  
 district (bimmer\_connected.models.PointOfInterestAddress attribute), 37  
 DOOR\_LOCK (bimmer\_connected.vehicle.remote\_services.Services attribute), 42  
 door\_lock\_state (bimmer\_connected.vehicle.doors\_windows.DoorsAndWindows attribute), 45  
 DOOR\_UNLOCK (bimmer\_connected.vehicle.remote\_services.Services attribute), 43  
 DoorsAndWindows (class in bimmer\_connected.vehicle.doors\_windows), 45  
 drive\_train (bimmer\_connected.vehicle.vehicle.MyBMWVehicle property), 39  
 drive\_train\_attributes (bimmer\_connected.vehicle.vehicle.MyBMWVehicle property), 39  
 due\_date (bimmer\_connected.vehicle.reports.ConditionBasedService attribute), 49  
 due\_distance (bimmer\_connected.vehicle.reports.ConditionBasedService attribute), 49  
 E  
 end\_time (bimmer\_connected.vehicle.charging\_profile.ChargingWindow property), 44  
 entryPoints (bimmer\_connected.models.PointOfInterest attribute), 36  
 ERROR (bimmer\_connected.vehicle.fuel\_and\_battery.ChargingState attribute), 46  
 ERROR (bimmer\_connected.vehicle.remote\_services.ExecutionState attribute), 41  
 EXECUTED (bimmer\_connected.vehicle.remote\_services.ExecutionState attribute), 41  
 ExecutionState (class in bimmer\_connected.vehicle.remote\_services), 41  
 F  
 filename (bimmer\_connected.models.AnonymizedResponse attribute), 35  
 FINISHED\_FULLY\_CHARGED (bimmer\_connected.vehicle.fuel\_and\_battery.ChargingState attribute), 46  
 FINISHED\_NOT\_FULL (bimmer\_connected.vehicle.fuel\_and\_battery.ChargingState attribute), 46  
 format\_for\_remote\_service() (bimmer\_connected.vehicle.charging\_profile.ChargingProfile method), 44  
 formattedAddress (bimmer\_connected.models.PointOfInterest attribute), 36  
 from\_api\_entry() (bimmer\_connected.vehicle.reports.CheckControlMessage class method), 48  
 from\_api\_entry() (bimmer\_connected.vehicle.reports.ConditionBasedService class method), 49  
 from\_vehicle\_data() (bimmer\_connected.models.VehicleDataBase class method), 37  
 from\_vehicle\_data() (bimmer\_connected.vehicle.fuel\_and\_battery.FuelAndBattery class method), 47  
 from\_vehicle\_data() (bimmer\_connected.vehicle.location.VehicleLocation class method), 48  
 FRONT (bimmer\_connected.vehicle.vehicle.VehicleViewDirection attribute), 41  
 FRONTSIDE (bimmer\_connected.vehicle.vehicle.VehicleViewDirection attribute), 41  
 FuelAndBattery (class in bimmer\_connected.vehicle.fuel\_and\_battery), 47  
 FULLY\_CHARGED (bimmer\_connected.vehicle.fuel\_and\_battery.ChargingState attribute), 46  
 G  
 gcid (bimmer\_connected.account.MyBMWAccount property), 31  
 generate\_state\_cn\_nonce() (in module bimmer\_connected.api.utils), 34  
 generate\_default\_header() (bimmer\_connected.api.client.MyBMWClient method), 33

`generate_random_base64_string()` (in module `bimmer_connected.api.utils`), 34

`generate_token()` (in module `bimmer_connected.api.utils`), 34

`get_app_version()` (in module `bimmer_connected.api.regions`), 33

`get_capture_position()` (in module `bimmer_connected.api.utils`), 34

`get_class_property_names()` (in module `bimmer_connected.utils`), 38

`get_correlation_id()` (in module `bimmer_connected.api.utils`), 34

`get_ocp_apim_key()` (in module `bimmer_connected.api.regions`), 33

`get_region_from_name()` (in module `bimmer_connected.api.regions`), 33

`get_retry_wait_time()` (in module `bimmer_connected.api.authentication`), 32

`get_server_url()` (in module `bimmer_connected.api.regions`), 33

`get_stored_responses()` (`bimmer_connected.account.MyBMWAccount` static method), 31

`get_user_agent()` (in module `bimmer_connected.api.regions`), 33

`get_vehicle()` (`bimmer_connected.account.MyBMWAccount` method), 31

`get_vehicle_image()` (`bimmer_connected.vehicle.vehicle.MyBMWVehicle` method), 39

`get_vehicle_state()` (`bimmer_connected.vehicle.vehicle.MyBMWVehicle` method), 39

`get_vehicles()` (`bimmer_connected.account.MyBMWAccount` method), 31

`go` (`bimmer_connected.models.PointOfInterestAddress` attribute), 37

`GPSPosition` (class in `bimmer_connected.models`), 35

## H

`handle_httpstatuserror()` (in module `bimmer_connected.api.utils`), 34

`has_check_control_messages` (`bimmer_connected.vehicle.reports.CheckControlMessageReport` attribute), 49

`has_combustion_drivetrain` (`bimmer_connected.vehicle.vehicle.MyBMWVehicle` property), 39

`has_electric_drivetrain` (`bimmer_connected.vehicle.vehicle.MyBMWVehicle` property), 39

`heading` (`bimmer_connected.vehicle.location.VehicleLocation` attribute), 48

`Headunit` (class in `bimmer_connected.vehicle.reports`), 50

`headunit_type` (`bimmer_connected.vehicle.reports.Headunit` attribute), 50

`HIGH` (`bimmer_connected.vehicle.reports.CheckControlStatus` attribute), 49

`HORN` (`bimmer_connected.vehicle.remote_services.Services` attribute), 43

`houseNumber` (`bimmer_connected.models.PointOfInterestAddress` attribute), 37

|

`idrive_version` (`bimmer_connected.vehicle.reports.Headunit` attribute), 50

`IGNORED` (`bimmer_connected.vehicle.remote_services.ExecutionState` attribute), 41

`IMMEDIATE_CHARGING` (`bimmer_connected.vehicle.charging_profile.ChargingMode` attribute), 43

`INITIATED` (`bimmer_connected.vehicle.remote_services.ExecutionState` attribute), 41

`INTERMEDIATE` (`bimmer_connected.vehicle.doors_windows.LidState` attribute), 46

`INVALID` (`bimmer_connected.vehicle.doors_windows.LidState` attribute), 46

`INVALID` (`bimmer_connected.vehicle.fuel_and_battery.ChargingState` attribute), 46

`is_charger_connected` (`bimmer_connected.vehicle.fuel_and_battery.FuelAndBattery` attribute), 47

`is_charging_plan_supported` (`bimmer_connected.vehicle.vehicle.MyBMWVehicle` property), 39

`is_charging_settings_supported` (`bimmer_connected.vehicle.vehicle.MyBMWVehicle` property), 39

`is_closed` (`bimmer_connected.vehicle.doors_windows.Lid` property), 45

`is_lsc_enabled` (`bimmer_connected.vehicle.vehicle.MyBMWVehicle` property), 39

`is_pre_entry_climatization_enabled` (`bimmer_connected.vehicle.charging_profile.ChargingProfile` attribute), 44

`is_remote_charge_start_enabled` (`bimmer_connected.vehicle.vehicle.MyBMWVehicle` property), 40

`is_remote_charge_stop_enabled` (`bimmer_connected.vehicle.vehicle.MyBMWVehicle` property), 40

`is_remote_climate_start_enabled` (`bimmer_connected.vehicle.vehicle.MyBMWVehicle` property), 40

[is\\_remote\\_climate\\_stop\\_enabled](#) (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
[is\\_remote\\_horn\\_enabled](#) (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
[is\\_remote\\_lights\\_enabled](#) (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
[is\\_remote\\_lock\\_enabled](#) (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
[is\\_remote\\_sendpoi\\_enabled](#) (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
[is\\_remote\\_set\\_ac\\_limit\\_enabled](#) (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
[is\\_remote\\_set\\_target\\_soc\\_enabled](#) (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
[is\\_remote\\_unlock\\_enabled](#) (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
[is\\_service\\_required](#) (*bimmer\_connected.vehicle.reports.ConditionBasedServiceReport* attribute), 50  
[is\\_vehicle\\_active](#) (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
[is\\_vehicle\\_tracking\\_enabled](#) (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
[isUnlockCableActive](#) (*bimmer\_connected.models.ChargingSettings* attribute), 35

## L

[lat](#) (*bimmer\_connected.models.PointOfInterest* attribute), 36  
[latitude](#) (*bimmer\_connected.models.GPSPosition* attribute), 35  
[Lid](#) (class in *bimmer\_connected.vehicle.doors\_windows*), 45  
[lids](#) (*bimmer\_connected.vehicle.doors\_windows.DoorsAndWindows* attribute), 45  
[LidState](#) (class in *bimmer\_connected.vehicle.doors\_windows*), 45  
[LIGHT\\_FLASH](#) (*bimmer\_connected.vehicle.remote\_services.Service* attribute), 43  
[location](#) (*bimmer\_connected.vehicle.location.VehicleLocation* attribute), 48

[locationAddress](#) (*bimmer\_connected.models.PointOfInterest* attribute), 36  
[LOCKED](#) (*bimmer\_connected.vehicle.doors\_windows.LockState* attribute), 46  
[LockState](#) (class in *bimmer\_connected.vehicle.doors\_windows*), 46  
[log\\_response\\_store\\_to\\_file\(\)](#) (in module *bimmer\_connected.utils*), 38  
[log\\_responses](#) (*bimmer\_connected.account.MyBMWAccount* attribute), 31  
[log\\_responses](#) (*bimmer\_connected.api.client.MyBMWClientConfiguration* attribute), 33  
[login\(\)](#) (*bimmer\_connected.api.authentication.MyBMWAuthentication* method), 32  
[login\\_lock](#) (*bimmer\_connected.api.authentication.MyBMWAuthentication* property), 32  
[lon](#) (*bimmer\_connected.models.PointOfInterest* attribute), 36  
[longitude](#) (*bimmer\_connected.models.GPSPosition* attribute), 35  
[LOW](#) (*bimmer\_connected.vehicle.reports.CheckControlStatus* attribute), 49  
[lsc\\_type](#) (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
[LscType](#) (class in *bimmer\_connected.vehicle.vehicle*), 38

## M

[MEDIUM](#) (*bimmer\_connected.vehicle.reports.CheckControlStatus* attribute), 49  
[messages](#) (*bimmer\_connected.vehicle.reports.CheckControlMessageReport* attribute), 49  
[messages](#) (*bimmer\_connected.vehicle.reports.ConditionBasedServiceReport* attribute), 50  
[mileage](#) (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
[MINI](#) (*bimmer\_connected.const.CarBrands* attribute), 34

## module

[bimmer\\_connected.account](#), 30  
[bimmer\\_connected.api.authentication](#), 32  
[bimmer\\_connected.api.client](#), 33  
[bimmer\\_connected.api.regions](#), 33  
[bimmer\\_connected.api.utils](#), 34  
[bimmer\\_connected.const](#), 34  
[bimmer\\_connected.models](#), 35  
[bimmer\\_connected.utils](#), 38  
[bimmer\\_connected.vehicle.charging\\_profile](#), 43  
[bimmer\\_connected.vehicle.doors\\_windows](#), 45  
[bimmer\\_connected.vehicle.fuel\\_and\\_battery](#), 46  
[bimmer\\_connected.vehicle.location](#), 48



**bimmer\_connected.vehicle.remote\_services**, 41  
**bimmer\_connected.vehicle.reports**, 48  
**bimmer\_connected.vehicle.vehicle**, 38  
**MyBMWAccount** (class in *bimmer\_connected.account*), 30  
**MyBMWAPIError**, 35  
**MyBMWAuthentication** (class in *bimmer\_connected.api.authentication*), 32  
**MyBMWAuthError**, 35  
**MyBMWClient** (class in *bimmer\_connected.api.client*), 33  
**MyBMWClientConfiguration** (class in *bimmer\_connected.api.client*), 33  
**MyBMWJSONEncoder** (class in *bimmer\_connected.utils*), 38  
**MyBMWLoginClient** (class in *bimmer\_connected.api.authentication*), 32  
**MyBMWLoginRetry** (class in *bimmer\_connected.api.authentication*), 32  
**MyBMWQuotaError**, 35  
**MyBMWRemoteServiceError**, 35  
**MyBMWVehicle** (class in *bimmer\_connected.vehicle.vehicle*), 39

**N**

**name** (*bimmer\_connected.models.PointOfInterest* attribute), 36  
**name** (*bimmer\_connected.vehicle.doors\_windows.Lid* attribute), 45  
**name** (*bimmer\_connected.vehicle.vehicle.MyBMWVehicle* property), 40  
**NO\_PRESELECTION** (*bimmer\_connected.vehicle.charging\_profile.ChargingProfile* attribute), 43  
**NORTH\_AMERICA** (*bimmer\_connected.const.Regions* attribute), 35  
**NOT\_CAPABLE** (*bimmer\_connected.vehicle.vehicle.LscType* attribute), 39  
**NOT\_CHARGING** (*bimmer\_connected.vehicle.fuel\_and\_battery.ChargingState* attribute), 47  
**NOT\_SUPPORTED** (*bimmer\_connected.vehicle.vehicle.LscType* attribute), 39

**O**

**observer\_position** (*bimmer\_connected.account.MyBMWAccount* attribute), 31  
**observer\_position** (*bimmer\_connected.api.client.MyBMWClientConfiguration* attribute), 33  
**OK** (*bimmer\_connected.vehicle.reports.CheckControlStatus* attribute), 49  
**OK** (*bimmer\_connected.vehicle.reports.ConditionBasedServiceStatus* attribute), 50

**OPEN** (*bimmer\_connected.vehicle.doors\_windows.LidState* attribute), 46  
**open\_lids** (*bimmer\_connected.vehicle.doors\_windows.DoorsAndWindows* property), 45  
**OPEN\_TILT** (*bimmer\_connected.vehicle.doors\_windows.LidState* attribute), 46  
**open\_windows** (*bimmer\_connected.vehicle.doors\_windows.DoorsAndWindows* property), 45  
**OVERDUE** (*bimmer\_connected.vehicle.reports.ConditionBasedServiceStatus* attribute), 50

**P**

**parse\_datetime()** (in module *bimmer\_connected.utils*), 38  
**PARTIALLY\_LOCKED** (*bimmer\_connected.vehicle.doors\_windows.LockState* attribute), 46  
**password** (*bimmer\_connected.account.MyBMWAccount* attribute), 31  
**PENDING** (*bimmer\_connected.vehicle.remote\_services.ExecutionState* attribute), 41  
**PENDING** (*bimmer\_connected.vehicle.reports.ConditionBasedServiceStatus* attribute), 50  
**phoneNumber** (*bimmer\_connected.models.PointOfInterest* attribute), 36  
**PLUGGED\_IN** (*bimmer\_connected.vehicle.fuel\_and\_battery.ChargingState* attribute), 47  
**PointOfInterest** (class in *bimmer\_connected.models*), 35  
**PointOfInterestAddress** (class in *bimmer\_connected.models*), 36  
**postalCode** (*bimmer\_connected.models.PointOfInterest* attribute), 36  
**postalCode** (*bimmer\_connected.models.PointOfInterestAddress* attribute), 37  
**preferred\_charging\_window** (*bimmer\_connected.vehicle.charging\_profile.ChargingProfile* attribute), 44  
**provider** (*bimmer\_connected.models.PointOfInterest* attribute), 36  
**providerId** (*bimmer\_connected.models.PointOfInterest* attribute), 36  
**providerPoiId** (*bimmer\_connected.models.PointOfInterest* attribute), 36

**R**

**refresh\_token** (*bimmer\_connected.account.MyBMWAccount* property), 31  
**region** (*bimmer\_connected.account.MyBMWAccount* attribute), 31  
**region** (*bimmer\_connected.models.PointOfInterestAddress* attribute), 37  
**regionCode** (*bimmer\_connected.models.PointOfInterestAddress* attribute), 37



Regions (class in <i>bimmer_connected.const</i> ), 34	set_remote_service_position() (bimmer_connected.vehicle.location.VehicleLocation method), 48
remaining_battery_percent (bimmer_connected.vehicle.fuel_and_battery.FuelAndBattery attribute), 47	settlement (bimmer_connected.models.PointOfInterestAddress attribute), 37
remaining_fuel (bimmer_connected.vehicle.fuel_and_battery.FuelAndBattery attribute), 47	SIDE (bimmer_connected.vehicle.vehicle.VehicleViewDirection attribute), 41
remaining_fuel_percent (bimmer_connected.vehicle.fuel_and_battery.FuelAndBattery attribute), 47	software_version (bimmer_connected.vehicle.reports.Headunit attribute), 50
remaining_range_electric (bimmer_connected.vehicle.fuel_and_battery.FuelAndBattery attribute), 48	sourceType (bimmer_connected.models.PointOfInterest attribute), 36
remaining_range_fuel (bimmer_connected.vehicle.fuel_and_battery.FuelAndBattery attribute), 48	start_time (bimmer_connected.vehicle.charging_profile.ChargingWindow property), 44
remaining_range_total (bimmer_connected.vehicle.fuel_and_battery.FuelAndBattery attribute), 48	Start_time (bimmer_connected.vehicle.charging_profile.DepartureTimer property), 44
remote_service_position (bimmer_connected.vehicle.location.VehicleLocation attribute), 48	state (bimmer_connected.vehicle.reports.CheckControlMessage attribute), 49
RemoteServices (class in bimmer_connected.vehicle.remote_services), 41	state (bimmer_connected.vehicle.reports.ConditionBasedService attribute), 49
RemoteServiceStatus (class in bimmer_connected.vehicle.remote_services), 41	street (bimmer_connected.models.PointOfInterest attribute), 36
REST_OF_WORLD (bimmer_connected.const.Regions attribute), 35	street (bimmer_connected.models.PointOfInterestAddress attribute), 37
	StrEnum (class in bimmer_connected.models), 37
	sync_auth_flow() (bimmer_connected.api.authentication.MyBMWAuthentication method), 32
	sync_auth_flow() (bimmer_connected.api.authentication.MyBMWLoginRetry method), 32
<b>S</b>	
<b>T</b>	
SECURED (bimmer_connected.vehicle.doors_windows.LockState attribute), 46	TARGET_REACHED (bimmer_connected.vehicle.fuel_and_battery.ChargingState attribute), 47
SELECTIVE_LOCKED (bimmer_connected.vehicle.doors_windows.LockState attribute), 46	timer_id (bimmer_connected.vehicle.charging_profile.DepartureTimer property), 44
SEND_POI (bimmer_connected.vehicle.remote_services.Services attribute), 43	timer_type (bimmer_connected.vehicle.charging_profile.ChargingProfile attribute), 44
service_type (bimmer_connected.vehicle.reports.ConditionBasedService attribute), 49	TimerTypes (class in bimmer_connected.vehicle.charging_profile), 44
Services (class in bimmer_connected.vehicle.remote_services), 42	timestamp (bimmer_connected.vehicle.vehicle.MyBMWVehicle property), 40
set_log_responses() (bimmer_connected.api.client.MyBMWClientConfiguration method), 33	to_camel_case() (in module bimmer_connected.utils), 38
set_observer_position() (bimmer_connected.account.MyBMWAccount method), 31	trigger_charge_start() (bimmer_connected.vehicle.remote_services.RemoteServices method), 41
set_refresh_token() (bimmer_connected.account.MyBMWAccount method), 31	trigger_charge_stop() (bimmer_connected.vehicle.remote_services.RemoteServices method), 41

trigger\_charging\_profile\_update() (bimmer\_connected.vehicle.remote\_services.RemoteServices attribute), 47  
 trigger\_charging\_profile\_update() (bimmer\_connected.vehicle.remote\_services.RemoteServices method), 42  
 trigger\_charging\_settings\_update() (bimmer\_connected.vehicle.remote\_services.RemoteServices attribute), 41  
 trigger\_charging\_settings\_update() (bimmer\_connected.vehicle.remote\_services.RemoteServices method), 42  
 trigger\_remote\_air\_conditioning() (bimmer\_connected.vehicle.remote\_services.RemoteServices attribute), 50  
 trigger\_remote\_air\_conditioning() (bimmer\_connected.vehicle.remote\_services.RemoteServices method), 42  
 trigger\_remote\_air\_conditioning\_stop() (bimmer\_connected.vehicle.remote\_services.RemoteServices attribute), 39  
 trigger\_remote\_air\_conditioning\_stop() (bimmer\_connected.vehicle.remote\_services.RemoteServices method), 42  
 trigger\_remote\_door\_lock() (bimmer\_connected.vehicle.remote\_services.RemoteServices attribute), 46  
 trigger\_remote\_door\_lock() (bimmer\_connected.vehicle.remote\_services.RemoteServices method), 42  
 trigger\_remote\_door\_unlock() (bimmer\_connected.vehicle.remote\_services.RemoteServices attribute), 37  
 trigger\_remote\_door\_unlock() (bimmer\_connected.vehicle.remote\_services.RemoteServices method), 42  
 trigger\_remote\_horn() (bimmer\_connected.vehicle.remote\_services.RemoteServices attribute), 41  
 trigger\_remote\_horn() (bimmer\_connected.vehicle.remote\_services.RemoteServices method), 42  
 trigger\_remote\_light\_flash() (bimmer\_connected.vehicle.remote\_services.RemoteServices attribute), 31  
 trigger\_remote\_light\_flash() (bimmer\_connected.vehicle.remote\_services.RemoteServices method), 42  
 trigger\_remote\_service() (bimmer\_connected.vehicle.remote\_services.RemoteServices attribute), 31  
 trigger\_remote\_service() (bimmer\_connected.vehicle.remote\_services.RemoteServices method), 42  
 trigger\_remote\_vehicle\_finder() (bimmer\_connected.vehicle.remote\_services.RemoteServices attribute), 33  
 trigger\_remote\_vehicle\_finder() (bimmer\_connected.vehicle.remote\_services.RemoteServices method), 42  
 trigger\_send\_poi() (bimmer\_connected.vehicle.remote\_services.RemoteServices attribute), 37  
 trigger\_send\_poi() (bimmer\_connected.vehicle.remote\_services.RemoteServices method), 42  
 try\_import\_pillow\_image() (in module bimmer\_connected.api.utils), 34  
 TWO\_TIMES\_TIMER (bimmer\_connected.vehicle.charging\_profile.TimerTypes attribute), 44  
 type (bimmer\_connected.models.PointOfInterest attribute), 36  
 U  
 unit (bimmer\_connected.models.ValueWithUnit attribute), 37  
 UNKNOWN (bimmer\_connected.vehicle.charging\_profile.ChargingMode attribute), 43  
 UNKNOWN (bimmer\_connected.vehicle.charging\_profile.ChargingPreferences attribute), 31  
 UNKNOWN (bimmer\_connected.vehicle.charging\_profile.ChargingPreferences attribute), 43  
 UNKNOWN (bimmer\_connected.vehicle.charging\_profile.TimerTypes attribute), 45  
 UNKNOWN (bimmer\_connected.vehicle.doors\_windows.LidState attribute), 46  
 UNKNOWN (bimmer\_connected.vehicle.doors\_windows.LockState attribute), 46  
 UNKNOWN (bimmer\_connected.vehicle.fuel\_and\_battery.ChargingState attribute), 47  
 UNKNOWN (bimmer\_connected.vehicle.remote\_services.ExecutionState attribute), 41  
 UNKNOWN (bimmer\_connected.vehicle.reports.CheckControlStatus attribute), 49  
 UNKNOWN (bimmer\_connected.vehicle.reports.ConditionBasedServiceStatus attribute), 50  
 UNKNOWN (bimmer\_connected.vehicle.vehicle.LscType attribute), 39  
 UNKNOWN (bimmer\_connected.vehicle.vehicle.VehicleViewDirection attribute), 41  
 UNLOCKED (bimmer\_connected.vehicle.doors\_windows.LockState attribute), 46  
 update\_from\_vehicle\_data() (bimmer\_connected.models.VehicleDataBase attribute), 37  
 update\_state() (bimmer\_connected.vehicle.vehicle.MyBMWVehicle attribute), 41  
 use\_metric\_units (bimmer\_connected.account.MyBMWAccount attribute), 31  
 username (bimmer\_connected.account.MyBMWAccount attribute), 31  
 V  
 valid\_regions() (in module bimmer\_connected.api.regions), 33  
 value (bimmer\_connected.models.ValueWithUnit attribute), 37  
 ValueWithUnit (class in bimmer\_connected.models), 37  
 VEHICLE\_FINDER (bimmer\_connected.vehicle.remote\_services.Services attribute), 43  
 vehicle\_update\_timestamp (bimmer\_connected.vehicle.location.VehicleLocation attribute), 48  
 vehicleCategoryId (bimmer\_connected.models.PointOfInterest attribute), 36  
 VehicleDataBase (class in bimmer\_connected.models), 37  
 VehicleLocation (class in bimmer\_connected.vehicle.location), 48  
 vehicles (bimmer\_connected.account.MyBMWAccount attribute), 31  
 VehicleViewDirection (class in bimmer\_connected.vehicle.vehicle), 41  
 vin (bimmer\_connected.vehicle.vehicle.MyBMWVehicle property), 41  
 WAITING\_FOR\_CHARGING (bimmer\_connected.vehicle.charging\_profile.ChargingMode attribute), 43

*mer\_connected.vehicle.fuel\_and\_battery.ChargingState*  
attribute), [47](#)

**weekdays** (*bimmer\_connected.vehicle.charging\_profile.DepartureTimer*  
property), [44](#)

**WEEKLY\_PLANNER** (*bim-*  
*mer\_connected.vehicle.charging\_profile.TimerTypes*  
attribute), [45](#)

**Window** (class in *bim-*  
*mer\_connected.vehicle.doors\_windows*),  
[46](#)

**windows** (*bimmer\_connected.vehicle.doors\_windows.DoorsAndWindows*  
attribute), [45](#)