# UNFCCC DI API Documentation

***Release 4.0.1***

**Mika Pflüger**

**Jan 08, 2024**

# CONTENTS:

CHAPTER

ONE

# UNFCCC DI API

Python wrapper around the Flexible Query API of the UNFCCC.

- Free software: Apache Software License 2.0
- Documentation: https://unfccc-di-api.readthedocs.io.

## 1.1 Warning

Due to a recent change in the UNFCCC's API, the UNFCCCApiReader class is **not functional** any more in standard environments. To continue to access the data, you have two options:

1. Use the new ZenodoReader. It provides access using the *query* function like the UNFCCCApiReader, but only supports querying for a full dataset with all data. It relies on our data package, which we update regularly; however, the data is naturally not as recent as querying from the API directly.
2. Run your functions in an environment which is not blocked by the UNFCCC DI API. According to our tests, Azure virtual machines work, as well as github hosted runners, with the exception of Mac OS runners.

## 1.2 Features

- High-level API to query all information for a given party.
- Low-level API to selectively query information with the same resolution as in the UNFCCC web query tool.

## 1.3 Citation

If you use this library and want to cite it, please cite it as:

Mika Pflüger, Daniel Huppmann & Johannes Gütschow. (2023-07-18). pik-primap/unfccc_di_api: Version 4.0.1. Zenodo. https://doi.org/10.5281/zenodo.8160056

## 1.4 Data package

If you just want all the data in CSV and parquet format (suitable for reading with pandas), look at our data package.

## 1.5 CI status and other links

CHAPTER

TWO

# INSTALLATION

## 2.1 Stable release

To install UNFCCC DI API, run this command in your terminal:

```
$ pip install unfccc_di_api
```

This is the preferred method to install UNFCCC DI API, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

## 2.2 From sources

The sources for UNFCCC DI API can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/pik-primap/unfccc_di_api
```

Or download the tarball:

```
$ curl -OJL https://github.com/pik-primap/unfccc_di_api/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

# THREE

# USAGE

To download the data from Zenodo, import the package and instantiate the reader, which will download the most recent dataset:

[1]:
```
import unfccc_di_api

reader = unfccc_di_api.ZenodoReader()
```

Check for which parties data is available:

[2]:
```
reader.parties
```

[2]:
```
['AUS',
 'AUT',
 'BEL',
 'BGR',
 'BLR',
 'CAN',
 'CHE',
 'CYP',
 'CZE',
 'DEU',
 'DNK',
 'ESP',
 'EST',
 'EUA',
 'FIN',
 'FRA',
 'GBR',
 'GRC',
 'HRV',
 'HUN',
 'IRL',
 'ISL',
 'ITA',
 'JPN',
 'KAZ',
 'LIE',
 'LTU',
 'LUX',
 'LVA',
 'MCO',
 'MLT',
 'NLD',
 'NOR',
 'NZL',
```

(continues on next page)

(continued from previous page)

```
'POL',
'PRT',
'ROU',
'RUS',
'SVK',
'SVN',
'SWE',
'TUR',
'UKR',
'USA',
'AFG',
'AGO',
'ALB',
'AND',
'ARE',
'ARG',
'ARM',
'ATG',
'AZE',
'BDI',
'BEN',
'BFA',
'BGD',
'BHR',
'BHS',
'BIH',
'BLZ',
'BOL',
'BRA',
'BRB',
'BRN',
'BTN',
'BWA',
'CAF',
'CHL',
'CHN',
'CIV',
'CMR',
'COD',
'COG',
'COK',
'COL',
'COM',
'CPV',
'CRI',
'CUB',
'DJI',
'DMA',
'DOM',
'DZA',
'ECU',
'EGY',
'ERI',
'ETH',
'FJI',
'FSM',
```

(continues on next page)

(continued from previous page)

```
'GAB',
'GEO',
'GHA',
'GIN',
'GMB',
'GNB',
'GNQ',
'GRD',
'GTM',
'GUY',
'HND',
'HTI',
'IDN',
'IND',
'IRN',
'IRQ',
'ISR',
'JAM',
'JOR',
'KEN',
'KGZ',
'KHM',
'KIR',
'KNA',
'KOR',
'KWT',
'LAO',
'LBN',
'LBR',
'LBY',
'LCA',
'LKA',
'LSO',
'MAR',
'MDA',
'MDG',
'MDV',
'MEX',
'MHL',
'MKD',
'MLI',
'MMR',
'MNE',
'MNG',
'MOZ',
'MRT',
'MUS',
'MWI',
'MYS',
'NAM',
'NER',
'NGA',
'NIC',
'NIU',
'NPL',
'NRU',
```

(continues on next page)

(continued from previous page)

```
 'OMN',
 'PAK',
 'PAN',
 'PER',
 'PHL',
 'PLW',
 'PNG',
 'PRK',
 'PRY',
 'PSE',
 'QAT',
 'RWA',
 'SAU',
 'SDN',
 'SEN',
 'SGP',
 'SLB',
 'SLE',
 'SLV',
 'SMR',
 'SOM',
 'SRB',
 'SSD',
 'STP',
 'SUR',
 'SWZ',
 'SYC',
 'SYR',
 'TCD',
 'TGO',
 'THA',
 'TJK',
 'TKM',
 'TLS',
 'TON',
 'TTO',
 'TUN',
 'TUV',
 'TZA',
 'UGA',
 'URY',
 'UZB',
 'VCT',
 'VEN',
 'VNM',
 'VUT',
 'WSM',
 'YEM',
 'ZAF',
 'ZMB',
 'ZWE']
```

Access all data for a specific party:

```
[3]: reader.query(party_code="AFG")
```

```
[3]:    party                    category      classification  \
```

(continues on next page)

(continued from previous page)

```
0     AFG                  1.  Energy  Total for category
1     AFG                  1.  Energy  Total for category
2     AFG                  1.  Energy  Total for category
3     AFG                  1.  Energy  Total for category
4     AFG                  1.  Energy  Total for category
..    ...                        ...                 ...
383   AFG  unknown category nr. 10504  Total for category
384   AFG  unknown category nr. 10504  Total for category
385   AFG  unknown category nr. 10504  Total for category
386   AFG  unknown category nr. 10504  Total for category
387   AFG  unknown category nr. 10504  Total for category

                    measure             gas               unit  year  \
0    Net emissions/removals  Aggregate GHGs  Gg CO2 equivalent  2005
1    Net emissions/removals  Aggregate GHGs  Gg CO2 equivalent  2013
2    Net emissions/removals             CH4                 Gg  2005
3    Net emissions/removals             CH4                 Gg  2013
4    Net emissions/removals              CO                 Gg  2005
..                      ...             ...                ...   ...
383        Total population          No gas          thousands  2017
384        Total population          No gas          thousands  2018
385        Total population          No gas          thousands  2019
386        Total population          No gas          thousands  2020
387        Total population          No gas          thousands  2021

     numberValue stringValue
0     3776.15946        None
1    10343.00000        None
2       35.06076        None
3        4.00000        None
4      865.20000        None
..           ...         ...
383  28224.32300        None
384  30075.01800        None
385  30725.56000        None
386  31390.17100        None
387  32069.16000        None

[388 rows x 9 columns]
```

You can also request data directly form the UNFCCC DI API if you have a way to get access. Check the API docs section (next section), or the docstrings of `UNFCCCApiReader` for details.

[ ]:

CHAPTER

# FOUR

# API

## 4.1 unfccc_di_api

The API provided by the UNFCCC distinguishes between parties listed in Annex I and the other parties, likely because the reporting requirements for Annex I parties and non-Annex I parties differ substantially. This library provides a wrapper class *UNFCCCApiReader* which unifies both APIs so that you don't have to worry about the status of a particular party. However, if you want to filter for specific variables and only query a subset of the data, you have to use the individual API objects for Annex I and non-Annex I parties, which are available at *UNFCCCApiReader.annex_one_reader* and *UNFCCCApiReader.non_annex_one_reader*, respectively.

**exception** unfccc_di_api.**NoDataError**(*party_codes: Sequence[str]*, *category_ids: Sequence[int] | None = None*, *classifications: Sequence[str] | None = None*, *measure_ids: Sequence[int] | None = None*, *gases: Sequence[str] | None = None*)

Query returned no data.

**class** unfccc_di_api.**UNFCCCApiReader**(*, *base_url: str = 'https://di.unfccc.int/api/'*)

Provides simplified unified access to the Flexible Query API of the UNFCCC data access for all parties.

Essentially encapsulates https://di.unfccc.int/flex_non_annex1 and https://di.unfccc.int/flex_annex1 .

**parties**

All parties, with their ID, code, and full name.

**Type**
pandas.DataFrame

**gases**

The available gases and their IDs.

**Type**
pandas.DataFrame

**annex_one_reader**

The API reader object for Annex I parties.

**Type**
*UNFCCCSingleCategoryApiReader*

**non_annex_one_reader**

The API reader object for non-Annex I parties.

**Type**
*UNFCCCSingleCategoryApiReader*

**query**(*, *party_code: str*, *gases: Sequence[str] | None = None*, *progress: bool = False*, *normalize_gas_names: bool = True*) → DataFrame

Query the UNFCCC for data.

**Parameters**

- **party_code** (*str*) – ISO code of a party for which to query. For possible values, see *parties*.
- **gases** (*list of str, optional*) – Limit the query to these gases. For possible values, see *gases*. Accepts subscripts ("$N_2O$") as well as ASCII-strings ("N2O"). Default: query for all gases.
- **progress** (*bool*) – Display a progress bar. Requires the `tqdm` library. Default: false.
- **normalize_gas_names** (*bool, optional*) – If `True`, return gases as ASCII strings ("N2O"). Else, return native UNFCCC notation ("$N_2O$"). Default: true.

**Return type**
pandas.DataFrame

### Notes

If you need more fine-grained control over which variables to query for, including restricting the query to specific measures, categories, or classifications or to query for multiple parties at once, please see the corresponding methods `UNFCCCApiReader.annex_one_reader.query()` and `UNFCCCApiReader.non_annex_one_reader.query()`.

**class** unfccc_di_api.**UNFCCCSingleCategoryApiReader**(*, *party_category: str*, *base_url: str* = *'https://di.unfccc.int/api/'*)

Provides access to the Flexible Query API of the UNFCCC data access for a single category, either annexOne or nonAnnexOne.

Use this class if you want to do fine-grained queries for specific measures, categories, years, or classifications.

Essentially encapsulates https://di.unfccc.int/flex_non_annex1 or https://di.unfccc.int/flex_annex1 .

**parties**

All parties in this category, with their ID, code, and full name.

**Type**
pandas.DataFrame

**years**

All years for which data is available, mapping the ID to the year.

**Type**
pandas.DataFrame

**category_tree**

The available categories and their relationships. Use *show_category_hierarchy()* for displaying the category tree.

**Type**
treelib.Tree

**classifications**

All classifications and their IDs.

**Type**
pandas.DataFrame

**measure_tree**

The available measures and their relationsips. Use *show_measure_hierarchy()* for displaying the measure tree.

**Type**
treelib.Tree

**gases**

The available gases and their IDs.

**Type**
pandas.DataFrame

**units**

The available units and their IDs.

**Type**
pandas.DataFrame

**conversion_factors**

Conversion factors between units for the specified gases.

**Type**
pandas.DataFrame

**variables**

The available variables with the corresponding category, classification, measure, gas, and unit.

**Type**
pandas.DataFrame

**query**(*, *party_codes: Sequence[str]*, *category_ids: Sequence[int] | None = None*, *classifications: Sequence[str] | None = None*, *measure_ids: Sequence[int] | None = None*, *gases: Sequence[str] | None = None*, *batch_size: int = 1000*, *progress: bool = False*, *normalize_gas_names: bool = True*) → DataFrame

Query the UNFCCC for data.

**Parameters**

- **party_codes** (*list of str*) – List of ISO codes of parties for which to query. For possible values, see *parties*.
- **category_ids** (*list of int, optional*) – List of category IDs to query. For possible values, see *show_category_hierarchy()*. Default: query for all categories.
- **classifications** (*list of str, optional*) – List of classifications to query. For possible values, see *classifications*. Default: query for all classifications.
- **measure_ids** (*list of int, optional*) – List of measure IDs to query. For possible values, see *show_measure_hierarchy()*. Default: query for all measures.
- **gases** (*list of str, optional*) – Limit the query to these gases. For possible values, see *gases*. Accepts subscripts ("$N_2O$") as well as ASCII-strings ("N2O"). Default: query for all gases.
- **batch_size** (*int, optional*) – Number of variables to query in a single API query in the same batch to avoid internal server errors. Larger queries are split automatically. The default is 1000, which seems to work fine.
- **progress** (*bool*) – Display a progress bar. Requires the `tqdm` library. Default: false.
- **normalize_gas_names** (*bool, optional*) – If `True`, return gases as ASCII strings ("N2O"). Else, return native UNFCCC notation ("$N_2O$"). Default: true.

**Return type**
pandas.DataFrame

#### Notes

Further documentation about the meaning of parties, categories, classifications, measures and gases is available at the UNFCCC documentation.

**show_category_hierarchy**() → None

Print the hierarchy of categories and their IDs.

**show_measure_hierarchy**() → None

Print the hierarchy of measures and their IDs.

**class** unfccc_di_api.**ZenodoReader**(*, *url: str = 'doi:10.5281/zenodo.10470862/parquet-only.zip'*, *known_hash: str = 'md5:52dd6cc26f1c2eb3f8204c6a78d2e7ba'*)

Provides simplified unified access to the data provided by the Flexible Query API of the UNFCCC data access, via the dataset stored at zenodo.

Essentially gives you the same API as the UNFCCCApiReader, but without complications due to the protection measures of the DI API. The advantage of using the ZenodoReader is that it works reliably without special measures, the disadvantage is that the data might be a bit older.

**parties**

All parties as a 3-letter iso code.

**Type**
list[str]

**query**(*, *party_code: str*, *gases: Sequence[str] | None = None*, *normalize_gas_names: bool = True*) → DataFrame

Query the dataset for party data.

**Parameters**

- **party_code** (*str*) – ISO code of a party for which to query. For possible values, see *parties*.
- **gases** (*list of str, optional*) – Limit the query to these gases. Accepts subscripts ("$N_2O$") as well as ASCII-strings ("N2O"). Default: query for all gases. Note that anything else than the default is not yet implemented and raises an error. Just request the whole dataset and filter using pandas' normal functionality.
- **normalize_gas_names** (*bool, optional*) – If True, return gases as ASCII strings ("N2O"). Else, return native UNFCCC notation ("$N_2O$"). Default: true. Note that anything else than the default is not implemented and raises an error. If you require unnormalized gas names, open an issue in the issue tracker at github so we can understand your use case.

**Return type**
pandas.DataFrame

## 4.2 unfccc_di_api.tests

Unit test package for unfccc_di_api.

Code coverage metrics:

CHAPTER

FIVE

# CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

## 5.1 Types of Contributions

### 5.1.1 Report Bugs

Report bugs at https://github.com/pik-primap/unfccc_di_api/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

### 5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

### 5.1.3 Write Documentation

UNFCCC DI API could always use more documentation, whether as part of the official UNFCCC DI API docs, in docstrings, or even on the web in blog posts, articles, and such.

### 5.1.4 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/pik-primap/unfccc_di_api/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that contributions are welcome :)

## 5.2 Get Started!

Ready to contribute? Here's how to set up *unfccc_di_api* for local development.

1. Fork the *unfccc_di_api* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/unfccc_di_api.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ cd unfccc_di_api/
$ make virtual-environment
$ make install-pre-commit
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass our tests and automatically format everything according to our rules:

```
$ make lint

Often, the linters can fix errors themselves, so if you get failures, run
``make lint`` again to see if any errors need human intervention.
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## 5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.9, 3.10, 3.11, and 3.12.

## 5.4 Deploying

A reminder for the maintainers on how to deploy.

1. Commit all your changes.
2. Replace the unreleased entry in CHANGELOG.rst with your target version number.
3. Run `tbump X.Y.Z`.
4. Go to github and make a release from the tag. Use "Version x.y.z" as the release title, and the changelog entries as the release description. Creating the github release will automatically trigger a release on zenodo.
5. Run `make update-citation` to update the citation information in the README.
6. Upload the release to pyPI: `make release`
7. To prepare for future development, add a new "unreleased" section to CHANGELOG.rst, and commit the result.

CHAPTER

SIX

# CREDITS

## 6.1 Developers

- Mika Pflüger mailto:mika.pflueger@climate-resource.com

## 6.2 Contributors

- Florian Dierickx floriandierickx.github.io
- Daniel Huppmann https://twitter.com/daniel_huppmann
- Johannes Gütschow <johannes.guetschow@climate-resource.com>

## 6.3 Libraries

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

SEVEN

# CHANGELOG

## 7.1 4.0.1 (2024-01-08)

- Use data released until 2023-01-08 when using the ZenodoReader.
- Support python 3.12.

## 7.2 4.0.0 (2023-07-18)

- Breaking: the UNFCCC restricted API access, likely you have to change your code to use the new ZenodoReader instead.
- Add ZenodoReader which doesn't rely on API access.
- Use data released until 2023-07-18 when using the ZenodoReader.
- Build the documentation on ReadTheDocs using newer Python and Sphinx versions.

## 7.3 3.0.2 (2022-12-13)

- Support python 3.11.
- Drop support for python 3.6.

## 7.4 3.0.1 (2022-03-15)

- Fix handling of unspecified measure IDs. The DI API started returning measure IDs without a name or description. We now call them `unknown measure nr. {measureId}` instead of erroring out.

## 7.5 3.0.0 (2021-12-03)

- Support python 3.10.
- Fix handling of duplicate variable IDs. **Note**: This entails changes to the public API! In particular, UNFCCCSingleCategoryApiReader.variables now has a generic index instead of using the `variableId` as index. Also, the `query` function now correctly restricts queries if `category_ids` are provided and correctly fills all categories with data for a multi-category variable.
- Fix pre-commit config for newer mypy type checking versions.
- Raise a more informative NoDataError (subclass of KeyError) instead of a generic KeyError when a query result is empty.

## 7.6 2.0.1 (2021-04-23)

- Change build system.

## 7.7 2.0.0 (2021-02-09)

- Accept ASCII format for `gases` when querying data and return gases & units normalized to ASCII (optional), thanks to Daniel Huppmann. Note that gases and units are normalized to ASCII by default, if you need the old behaviour for compatibility reasons, pass `normalize_gas_names=False` to your `query()` calls.

## 7.8 1.1.1 (2021-02-08)

- Include ipython notebooks and CHANGELOG in release tarballs.

## 7.9 1.1.0 (2021-01-25)

- Add a useful error message when querying for unknown parties, thanks to Daniel Huppmann.

## 7.10 1.0.0 (2021-01-22)

- Add continuous integration using GitHub actions.
- Add tests.
- Add usage documentation in notebook format.
- Documentation fixes.

## 7.11 0.1.0 (2021-01-22)

- First release on PyPI.
- Convert API wrapper into standalone Python package.

CHAPTER

EIGHT

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## u

# INDEX