AirBase Documentation

Release 0.7.0

John Paton

2023-04-25

CONTENTS

1 airbase package **1**
- 1.1 Submodules . . . 4
- 1.2 airbase.resources module . . . 4
- 1.3 airbase.util module . . . 4
- 1.4 airbase.fetch module . . . 5
- 1.5 airbase.summary module . . . 7

2 Installation **9**

3 Quickstart **11**

4 Key Concepts **13**
- 4.1 `AirbaseClient` . . . 13
- 4.2 `AirbaseRequest` . . . 13
- 4.3 `pl` and `shortpl` . . . 13

5 Indices and tables **15**

Python Module Index **17**

Index **19**

CHAPTER

ONE

AIRBASE PACKAGE

class airbase.**AirbaseClient**

Bases: `object`

The central point for requesting Airbase data.

Example

```
>>> client = AirbaseClient()
>>> r = client.request(["NL", "DE"], pl=["O3", "NO2"])
>>> r.download_to_directory("data/raw")
Generating CSV download links...
100%|| 4/4 [00:09<00:00,  2.64s/it]
Generated 5164 CSV links ready for downloading
Downloading CSVs to data/raw...
100%|| 5164/5164 [43:39<00:00,  1.95it/s]
>>> r.download_metadata("data/metadata.tsv")
Writing metadata to data/metadata.tsv...
```

property all_countries: list[str]

property all_pollutants: dict[str, str]

countries

All pollutants available from AirBase

static download_metadata(*filepath*, *verbose=True*)

Download the metadata file.

See http://discomap.eea.europa.eu/map/fme/AirQualityExport.htm.

Parameters

- **filepath** (*str | Path*) –
- **verbose** (*bool*) –

Return type
None

request(*country=None*, *pl=None*, *shortpl=None*, *year_from='2013'*, *year_to='2023'*, *source='All'*, *update_date=None*, *verbose=True*, *preload_csv_links=False*)

Initialize an AirbaseRequest for a query.

Pollutants can be specified either by name (*pl*) or by code (*shortpl*). If no pollutants are specified, data for all available pollutants will be requested. If a pollutant is not available for a country, then we simply do not try to download those CSVs.

Requests proceed in two steps: First, links to individual CSVs are requested from the Airbase server. Then these links are used to download the individual CSVs.

See http://discomap.eea.europa.eu/map/fme/AirQualityExport.htm.

Parameters

- **country** (*str | list[str] | None*) – (optional), 2-letter country code or a list of them. If a list, data will be requested for each country. Will raise ValueError if a country is not available on the server. If None, data for all countries will be requested. See *self.all_countries*.
- **pl** (*str | list[str] | None*) – (optional) The pollutant(s) to request data for. Must be one of the pollutants in *self.all_pollutants*. Cannot be used in conjunction with *shortpl*.
- **shortpl** (*str | list[str] | None*) – (optional). The pollutant code(s) to request data for. Will be applied to each country requested. Cannot be used in conjunction with *pl*. Deprecated, will be removed on v1.
- **year_from** (*str*) – (optional) The first year of data. Can not be earlier than 2013. Default 2013.
- **year_to** (*str*) – (optional) The last year of data. Can not be later than the current year. Default <current year>.
- **source** (*str*) – (optional) One of "E1a", "E2a" or "All". E2a (UTD) data are only available for years where E1a data have not yet been delivered (this will normally be the most recent year). Default "All".
- **update_date** (*str | datetime | None*) – (optional). Format "yyyy-mm-dd hh:mm:ss". To be used when only files created or updated after a certain date is of interest.
- **verbose** (*bool*) – (optional) print status messages to stderr. Default True.
- **preload_csv_links** (*bool*) – (optional) Request all the csv download links from the Airbase server at object initialization. Default False.

Return AirbaseRequest

The initialized AirbaseRequest.

Example

```
>>> client = AirbaseClient()
>>> r = client.request(["NL", "DE"], pl=["O3", "NO2"])
>>> r.download_to_directory("data/raw")
Generating CSV download links...
100%|| 4/4 [00:09<00:00,  2.64s/it]
Generated 5164 CSV links ready for downloading
Downloading CSVs to data/raw...
100%|| 5164/5164 [43:39<00:00,  1.95it/s]
>>> r.download_metadata("data/metadata.tsv")
Writing metadata to data/metadata.tsv...
```

Return type

AirbaseRequest

search_pollutant(*query*, *limit=None*)

Search for a pollutant's *shortpl* number based on its name.

Parameters

- **query** (*str*) – The pollutant to search for.
- **limit** (*int | None*) – (optional) Max number of results.

Returns
The best pollutant matches. Pollutants are dicts with keys "pl" and "shortpl".

Example

```
>>> AirbaseClient().search_pollutant("o3", limit=2)
>>> [{"pl": "O3", "shortpl": "7"}, {"pl": "NO3", "shortpl": "46"}]
```

Return type
list[airbase.airbase.PollutantDict]

class airbase.**AirbaseRequest**(*country=None*, *shortpl=None*, *year_from='2013'*, *year_to='2023'*, *source='All'*, *update_date=None*, *verbose=True*, *preload_csv_links=False*)

Bases: `object`

Handler for Airbase data requests.

Requests proceed in two steps: First, links to individual CSVs are requested from the Airbase server. Then these links are used to download the individual CSVs.

See http://discomap.eea.europa.eu/map/fme/AirQualityExport.htm.

Parameters

- **country** (*str | list[str] | None*) – 2-letter country code or a list of them. If a list, data will be requested for each country.
- **shortpl** (*str | list[str] | None*) – (optional). The pollutant code to request data for. Will be applied to each country requested. If None, all available pollutants will be requested. If a pollutant is not available for a country, then we simply do not try to download those CSVs.
- **year_from** (*str*) – (optional) The first year of data. Can not be earlier than 2013. Default 2013.
- **year_to** (*str*) – (optional) The last year of data. Can not be later than the current year. Default <current year>.
- **source** (*str*) – (optional) One of "E1a", "E2a" or "All". E2a (UTD) data are only available for years where E1a data have not yet been delivered (this will normally be the most recent year). Default "All".
- **update_date** (*str | datetime | None*) – (optional). Format "yyyy-mm-dd hh:mm:ss". To be used when only files created or updated after a certain date is of interest.
- **verbose** (*bool*) – (optional) print status messages to stderr. Default True.
- **preload_csv_links** (*bool*) – (optional) Request all the csv download links from the Airbase server at object initialization. Default False.

download_metadata(*filepath*)

Download the metadata TSV file.

See http://discomap.eea.europa.eu/map/fme/AirQualityExport.htm.

Parameters
filepath (*str | Path*) – Where to save the TSV

Return type
None

download_to_directory(*dir*, *skip_existing=True*, *raise_for_status=True*)

Download into a directory, preserving original file structure.

Parameters

- **dir** (*str | Path*) – The directory to save files in (must exist)
- **skip_existing** (*bool*) – (optional) Don't re-download files if they exist in *dir*. If False, existing files in *dir* may be overwritten. Default True.
- **raise_for_status** (*bool*) – (optional) Raise exceptions if download links return "bad" HTTP status codes. If False, a `warnings.warn()` will be issued instead. Default True.

Returns
self

Return type
AirbaseRequest

download_to_file(*filepath*, *raise_for_status=True*)

Download data into one large CSV.

Directory where the new CSV will be created must exist.

Parameters

- **filepath** (*str | Path*) – The path to the new CSV.
- **raise_for_status** (*bool*) – (optional) Raise exceptions if download links return "bad" HTTP status codes. If False, a `warnings.warn()` will be issued instead. Default True.

Returns
self

Return type
AirbaseRequest

1.1 Submodules

1.2 airbase.resources module

Global variables for URL templating

1.3 airbase.util module

Utility functions for processing the raw Portal responses, url templating, etc.

airbase.util.**link_list_url**(*country*, *shortpl=None*, *year_from='2013'*, *year_to='2023'*, *source='All'*, *update_date=None*)

Generate the URL where the download links for a query can be found.

Parameters

- **country** (*str | None*) – The 2-letter country code. See AirbaseClient.countries for options.
- **shortpl** (*str | None*) – (optional) The pollutant number. Leave blank to get all pollutants. See AirbaseClient.pollutants_per_country for options.

- **year_from** (*str*) – (optional) The first year of data. Can not be earlier than 2013. Default 2013.
- **year_to** (*str*) – (optional) The last year of data. Can not be later than the current year. Default <current year>.
- **source** (*str*) – (optional) One of "E1a", "E2a" or "All". E2a (UTD) data are only available for years where E1a data have not yet been delivered (this will normally be the most recent year). Default "All".
- **update_date** (*str | datetime | None*) – (optional). Format "yyyy-mm-dd hh:mm:ss". To be used when only files created or updated after a certain date is of interest.

Returns
The URL which will yield the list of relevant CSV download links.

Return type
str

airbase.util.**string_safe_list**(*obj: None*) → list[None]

airbase.util.**string_safe_list**(*obj: str | Iterable[str]*) → list[str]

Turn an (iterable) object into a list. If it is a string or not iterable, put the whole object into a list of length 1.

Parameters
obj –

Return list

1.4 airbase.fetch module

Helper functions encapsulating async HTTP request and file IO

airbase.fetch.**fetch_json**(*url*, *, *timeout=None*, *encoding=None*)

Request url and read response's body as JSON

Parameters

- **url** (*str*) – requested url
- **timeout** (*float | None*) – maximum time to complete request (seconds)
- **encoding** (*str | None*) – text encoding used for decoding the response's body

Returns
decoded text from response's body as JSON

Return type
list[dict[str, str]]

airbase.fetch.**fetch_text**(*url*, *, *timeout=None*, *encoding=None*)

Request url and read response's body

Parameters

- **url** (*str*) – requested url
- **timeout** (*float | None*) – maximum time to complete request (seconds)
- **encoding** (*str | None*) – text encoding used for decoding the response's body

Returns
decoded text from response's body

Return type
str

airbase.fetch.**fetch_to_directory**(*urls*, *root*, ***, *skip_existing=True*, *encoding=None*, *progress=False*, *raise_for_status=True*, *max_concurrent=10*)

Request a list of url write each response to different file

Parameters

- **urls** (*list[str]*) – requested urls
- **root** (*Path*) – directory to write all responses
- **skip_existing** (*bool*) – Do not re-download url if the corresponding file is found in *root*
- **encoding** (*str | None*) – text encoding used for decoding each response's body
- **progress** (*bool*) – show progress bar
- **raise_for_status** (*bool*) – Raise exceptions if download links return "bad" HTTP status codes. If False, a `warnings.warn()` will be issued instead.
- **max_concurrent** (*int*) – maximum concurrent requests

Return type
None

airbase.fetch.**fetch_to_file**(*urls*, *path*, ***, *encoding=None*, *progress=False*, *raise_for_status=True*, *max_concurrent=10*)

Request a list of url write out all responses into a single text file

Parameters

- **urls** (*list[str]*) – requested urls
- **path** (*Path*) – text file for all combined responses
- **encoding** (*str | None*) – text encoding used for decoding each response's body
- **progress** (*bool*) – show progress bar
- **raise_for_status** (*bool*) – Raise exceptions if download links return "bad" HTTP status codes. If False, a `warnings.warn()` will be issued instead.
- **max_concurrent** (*int*) – maximum concurrent requests

Return type
None

airbase.fetch.**fetch_unique_lines**(*urls*, ***, *encoding=None*, *progress=False*, *raise_for_status=True*, *max_concurrent=10*)

Request a list of url and return only the unique lines among all the responses

Parameters

- **urls** (*list[str]*) – requested urls
- **encoding** (*str | None*) – text encoding used for decoding each response's body
- **progress** (*bool*) – show progress bar
- **raise_for_status** (*bool*) – Raise exceptions if download links return "bad" HTTP status codes. If False, a `warnings.warn()` will be issued instead.
- **max_concurrent** (*int*) – maximum concurrent requests

Returns
unique lines among from all the responses

Return type
set[str]

async airbase.fetch.**fetcher**(*urls: list[str]*, ***, *encoding: str | None = None*, *progress: bool = DEFAULT.progress*, *raise_for_status: bool = DEFAULT.raise_for_status*, *max_concurrent: int = DEFAULT.max_concurrent*) → AsyncIterator[str]

async airbase.fetch.**fetcher**(*urls: dict[str, pathlib.Path]*, ***, *encoding: str | None = None*, *progress: bool = DEFAULT.progress*, *raise_for_status: bool = DEFAULT.raise_for_status*, *max_concurrent: int = DEFAULT.max_concurrent*) → AsyncIterator[Path]

Request multiple urls and write request text into individual paths it a *dict[url, path]* is provided, or return the decoded text from each request if only a *list[url]* is provided.

Parameters

- **urls** – requested urls
- **encoding** – text encoding used for decoding each response's body
- **progress** – show progress bar
- **raise_for_status** – Raise exceptions if download links return "bad" HTTP status codes. If False, a `warnings.warn()` will be issued instead.
- **max_concurrent** – maximum concurrent requests

Returns
url text or path to downloaded text, one by one as the requests are completed

1.5 airbase.summary module

class airbase.summary.**DB**

Bases: `object`

In DB containing the available country and pollutants

classmethod **countries**()

Get the list of unique countries from the summary.

Returns
list of available country codes

Return type
list[str]

classmethod **cursor**()

db cursor as a "self closing" context manager

Return type
Iterator[*Cursor*]

db = <sqlite3.Connection object>

classmethod **pollutants**()

Get the list of unique pollutants from the summary.

Parameters
summary – The E1a summary.

Returns
The available pollutants, as a dictionary with

Return type
dict[str, str]

with name as keys with name as values, e.g. {"NO": "38", ...}

`classmethod pollutants_per_country()`

Get the available pollutants per country from the summary.

Returns
All available pollutants per country, as a dictionary with

Return type
dict[str, dict[str, int]]

with country code as keys and a dictionary of pollutant/ids (e.g. {"NO": 38, ...}) as values.

`classmethod search_pollutant(query, *, limit=None)`

Search for a pollutant's ID number based on its name.

Parameters

- **query** (*str*) – The pollutant to search for.
- **limit** (*int | None*) – (optional) Max number of results.

Returns
The best pollutant matches, as a dictionary with

Return type
dict[str, int]

with name as keys with name as values, e.g. {"NO": 38, ...}

An easy downloader for the AirBase air quality data.

AirBase is an air quality database provided by the European Environment Agency (EEA). The data is available for download at the Portal, but the interface makes it a bit time consuming to do bulk downloads. Hence, an easy Python-based interface.

CHAPTER

TWO

INSTALLATION

To install `airbase`, simply run

```
$ pip install airbase
```

`airbase` has been tested on Python 3.7 and higher.

CHAPTER

THREE

QUICKSTART

Get info about available countries and pollutants:

```
>>> import airbase
>>> client = airbase.AirbaseClient()
>>> client.all_countries
['GR', 'ES', 'IS', 'CY', 'NL', 'AT', 'LV', 'BE', 'CH', 'EE', 'FR', 'DE', ...

>>> client.all_pollutants
{'k': 412, 'CO': 10, 'NO': 38, 'O3': 7, 'As': 2018, 'Cd': 2014, ...

>>> client.pollutants_per_country
{'AD': [{'pl': 'CO', 'shortpl': 10}, {'pl': 'NO', 'shortpl': 38}, ...

>>> client.search_pollutant("O3")
[{'pl': 'O3', 'shortpl': 7}, {'pl': 'NO3', 'shortpl': 46}, ...
```

Request download links from the server and save the resulting CSVs into a directory:

```
>>> r = client.request(country=["NL", "DE"], pl="NO3", year_from=2015)
>>> r.download_to_directory(dir="data", skip_existing=True)
Generating CSV download links...
100%|| 2/2 [00:03<00:00,  2.03s/it]
Generated 12 CSV links ready for downloading
Downloading CSVs to data...
100%|| 12/12 [00:01<00:00,  8.44it/s]
```

Or concatenate them into one big file:

```
>>> r = client.request(country="FR", pl=["O3", "PM10"], year_to=2014)
>>> r.download_to_file("data/raw.csv")
Generating CSV download links...
100%|| 2/2 [00:12<00:00,  7.40s/it]
Generated 2,029 CSV links ready for downloading
Writing data to data/raw.csv...
100%|| 2029/2029 [31:23<00:00,  1.04it/s]
```

Download the entire dataset (not for the faint of heart):

```
>>> r = client.request()
>>> r.download_to_directory("data")
```

(continues on next page)

(continued from previous page)

```
Generating CSV download links...
100%|| 40/40 [03:38<00:00,  2.29s/it]
Generated 146,993 CSV links ready for downloading
Downloading CSVs to data...
  0%|          | 299/146993 [01:50<17:15:06,  2.36it/s]
```

Don't forget to get the metadata about the measurement stations:

```
>>> client.download_metadata("data/metadata.tsv")
Writing metadata to data/metadata.tsv...
```

CHAPTER

FOUR

KEY CONCEPTS

The `airbase` package is centered around two key objects: the `AirbaseClient` and the `AirbaseRequest`.

4.1 `AirbaseClient`

The client is responsible for generating and validating requests. It does this by gathering information from the AirBase Portal when it is initialized, allowing it to know which countries and pollutants are currently available.

4.2 `AirbaseRequest`

The request is an object that is generally created using the `AirbaseClient.request` method. The request automatically handles the 2-step process of generating CSV links for your query using the AirBase Portal, and downloading the resulting list of CSVs. All that the user needs to do is choose where the downloaded CSVs should be saved, and whether they should stay seperate or get concatenated into one big file.

By default, the request will request the entire dataset, which will take most of a day to download. Its arguments can be used to filter to only specific dates, countries, pollutants, etc.

4.3 `pl` and `shortpl`

The common abbreviations for pollutants ("O3", "NOX", "PM10", etc.) are referred to in the `airbase` package as `pl`. The AirBase Portal internally makes use of a numeric system for labelling pollutants, which we refer to as the `shortpl`. The client is built in such a way as to only require knowing the familiar `pl` you are looking for, but the pollutant lists and search functionality provided by the client will always return both the `pl` and the `shortpl` for every pollutant, as these are required for constructing the requests and communicating with the AirBase Portal.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

airbase, 1
airbase.fetch, 5
airbase.resources, 4
airbase.summary, 7
airbase.util, 4

INDEX

A

airbase
    module, 1
airbase.fetch
    module, 5
airbase.resources
    module, 4
airbase.summary
    module, 7
airbase.util
    module, 4
AirbaseClient (*class in airbase*), 1
AirbaseRequest (*class in airbase*), 3
all_countries (*airbase.AirbaseClient property*), 1
all_pollutants (*airbase.AirbaseClient property*), 1

C

countries (*airbase.AirbaseClient attribute*), 1
countries() (*airbase.summary.DB class method*), 7
cursor() (*airbase.summary.DB class method*), 7

D

db (*airbase.summary.DB attribute*), 7
DB (*class in airbase.summary*), 7
download_metadata() (*airbase.AirbaseClient static method*), 1
download_metadata() (*airbase.AirbaseRequest method*), 3
download_to_directory() (*airbase.AirbaseRequest method*), 3
download_to_file() (*airbase.AirbaseRequest method*), 4

F

fetch_json() (*in module airbase.fetch*), 5
fetch_text() (*in module airbase.fetch*), 5
fetch_to_directory() (*in module airbase.fetch*), 6
fetch_to_file() (*in module airbase.fetch*), 6
fetch_unique_lines() (*in module airbase.fetch*), 6
fetcher() (*in module airbase.fetch*), 7

L

link_list_url() (*in module airbase.util*), 4

M

module
    airbase, 1
    airbase.fetch, 5
    airbase.resources, 4
    airbase.summary, 7
    airbase.util, 4

P

pollutants() (*airbase.summary.DB class method*), 7
pollutants_per_country() (*airbase.summary.DB class method*), 8

R

request() (*airbase.AirbaseClient method*), 1

S

search_pollutant() (*airbase.AirbaseClient method*), 2
search_pollutant() (*airbase.summary.DB class method*), 8
string_safe_list() (*in module airbase.util*), 5