apybiomart Documentation

Release 0.5.3

Roberto Preste

Nov 30, 2020

CONTENTS:

Async pythonic interface to BioMart.

- Free software: MIT license
- Documentation: https://apybiomart.readthedocs.io
- GitHub repo: https://github.com/robertopreste/apybiomart

CHAPTER

ONE

FEATURES

apybiomart is a Python module which provides a simple asynchronous interface to Ensembl BioMart. Users can exploit the async interface to schedule multiple queries using all the commodities offered by Python's asyncio library.

Depending on specific needs, apybiomart offers different entry points:

- an asynchronous `aquery()` function, to schedule multiple queries in the same event loop;
- a synchronous `query()` function, which can be used for exploratory queries, executed in real time;
- a set of synchronous `find_*()` functions, which can be used to retrieve the list of available marts (`find_marts()`), datasets for a specific mart (`find_datasets()`), attributes (`find_attributes()`) and filters (`find_filters()`) for a specific dataset.
  - a set of related CLI commands also exists to allow exploration of these data from the command line; these are, respectively, `apybiomart marts`, `apybiomart datasets`, `apybiomart attributes` and `apybiomart filters`. Run `apybiomart --help` for further details.

Please refer to the Usage section of the documentation for further information.

1.1 Background

apybiomart was originally born as a fork of the great pybiomart package.

I was working on a project that employed a series of async calls to several online resources, but I couldn't manage to perform asynchronous calls to BioMart using that package, so I decided to modify it to better suit my needs. However, it gradually evolved into a very different thing: the original implementation was rewritten and the structure of the package changed a bit, in a way that I found most useful for my purpose.

This said, all the credits go to jrderuiter, which created the original pybiomart package.

CHAPTER TWO

INSTALLATION

apybiomart only supports Python 3, and can be installed using pip:

```
pip install apybiomart
```

Please refer to the Installation section of the documentation for further information.

CHAPTER THREE

CREDITS

This package was created with Cookiecutter and the cc-pypackage project template.

CHAPTER

FOUR

TABLE OF CONTENTS

4.1 apybiomart

Async pythonic interface to BioMart.

- Free software: MIT license
- Documentation: https://apybiomart.readthedocs.io
- GitHub repo: https://github.com/robertopreste/apybiomart

4.1.1 Features

apybiomart is a Python module which provides a simple asynchronous interface to Ensembl BioMart. Users can exploit the async interface to schedule multiple queries using all the commodities offered by Python's asyncio library.

Depending on specific needs, apybiomart offers different entry points:

- an asynchronous `aquery()` function, to schedule multiple queries in the same event loop;
- a synchronous `query()` function, which can be used for exploratory queries, executed in real time;
- a set of synchronous `find_*()` functions, which can be used to retrieve the list of available marts (`find_marts()`), datasets for a specific mart (`find_datasets()`), attributes (`find_attributes()`) and filters (`find_filters()`) for a specific dataset.
  - a set of related CLI commands also exists to allow exploration of these data from the command line; these are, respectively, `apybiomart marts`, `apybiomart datasets`, `apybiomart attributes` and `apybiomart filters`. Run `apybiomart --help` for further details.

Please refer to the Usage section of the documentation for further information.

Background

apybiomart was originally born as a fork of the great pybiomart package.

I was working on a project that employed a series of async calls to several online resources, but I couldn't manage to perform asynchronous calls to BioMart using that package, so I decided to modify it to better suit my needs. However, it gradually evolved into a very different thing: the original implementation was rewritten and the structure of the package changed a bit, in a way that I found most useful for my purpose.

This said, all the credits go to jrderuiter, which created the original pybiomart package.

4.1.2 Installation

apybiomart only supports Python 3, and can be installed using pip:

```
pip install apybiomart
```

Please refer to the Installation section of the documentation for further information.

4.1.3 Credits

This package was created with Cookiecutter and the cc-pypackage project template.

4.2 Installation

PLEASE NOTE: apybiomart only supports Python 3!

4.2.1 Stable release

To install apybiomart, run this command in your terminal:

```
$ pip install apybiomart
```

This is the preferred method to install apybiomart, 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.

4.2.2 From sources

The sources for apybiomart can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/robertopreste/apybiomart
```

Or download the tarball:

```
$ curl  -OL https://github.com/robertopreste/apybiomart/tarball/master
```

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

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

4.3 Usage

apybiomart can be used in a project with a simple import:

```
import apybiomart
```

The main purpose of the package is to perform queries on BioMart (either synchronously or asynchronously), however users may first need to explore the available marts, datasets, attributes and filters.

In addition to interactively inspect these results, users can also save them to a CSV file, using the `--save` flag on the CLI and the `save=True` argument in Python, and optionally specify a filename using the `--output <filename.csv>` option on the CLI and the `output="filename.csv"` argument in Python.

4.3.1 Marts, datasets, attributes and filters

BioMart contains different databases, called *marts*, each of which in turn contains several *datasets*, each related to a specific species. These datasets can be queried and it is possible to restrict the amount of data returned to one or more particular types of information, namely *attributes*, and using *filters* that only retain data satisfying one or more specific criteria.

For more information, please refer to BioMart's help page.

Marts

In order to view the marts available on BioMart, the `find_marts()` function can be used:

```
from apybiomart import find_marts
find_marts()
```

A dataframe with the available marts is returned, with their proper `name` and `display_name`:

```
                 Mart_ID              Mart_name
0   ENSEMBL_MART_ENSEMBL       Ensembl Genes 96
1     ENSEMBL_MART_MOUSE       Mouse strains 96
2  ENSEMBL_MART_SEQUENCE               Sequence
3  ENSEMBL_MART_ONTOLOGY               Ontology
4   ENSEMBL_MART_GENOMIC    Genomic features 96
5       ENSEMBL_MART_SNP   Ensembl Variation 96
6   ENSEMBL_MART_FUNCGEN  Ensembl Regulation 96
```

A CLI command is also available to retrieve the same information: `apybiomart marts`.

Datasets

Available datasets for a specific mart can be retrieved using the `find_datasets()` function:

```
from apybiomart import find_datasets
find_datasets(mart="ENSEMBL_MART_ENSEMBL")
# same as above, using the default mart
find_datasets()
```

The `find_datasets()` function accepts an optional `mart` argument, which defaults to "ENSEMBL_MART_ENSEMBL". The returned dataframe contains all the available datasets in the given mart, with their `name`, `display_name` and the `mart` to which they belong:

```
                     Dataset_ID                                  Dataset_name   ␣
↪            Mart_ID
0       rroxellana_gene_ensembl          Golden snub-nosed monkey genes (Rrox_v1) ␣
↪ENSEMBL_MART_ENSEMBL
1          ggallus_gene_ensembl                          Chicken genes (GRCg6a) ␣
↪ENSEMBL_MART_ENSEMBL
2     dmelanogaster_gene_ensembl          Drosophila melanogaster genes (BDGP6.22) ␣
↪ENSEMBL_MART_ENSEMBL
..                          ...                                            ...   ␣
↪                  ...
181       sdorsalis_gene_ensembl               Yellowtail amberjack genes (Sedor1) ␣
↪ENSEMBL_MART_ENSEMBL
182            ohni_gene_ensembl           Japanese medaka HNI genes (ASM223471v1) ␣
↪ENSEMBL_MART_ENSEMBL
183        pmarinus_gene_ensembl                      Lamprey genes (Pmarinus_7.0) ␣
↪ENSEMBL_MART_ENSEMBL
```

A CLI command is also available to retrieve the same information: `apybiomart datasets`, whose `--mart` option can be used to specify which mart will be used (default is "ENSEMBL_MART_ENSEMBL").

Attributes

When querying a dataset, users may want to retrieve specific attributes; the `find_attributes()` function accepts an optional `dataset` (defaulting to "hsapiens_gene_ensembl") and gathers all the available attributes for the given dataset:

```
from apybiomart import find_attributes
find_attributes(dataset="hsapiens_gene_ensembl")
# same as above, using the default dataset
find_attributes()
```

The dataframe returned contains each attribute's `name`, `display_name`, `description` (where available), and the `dataset` to which it belongs:

```
              Attribute_ID          Attribute_name          Attribute_description ␣
↪            Dataset_ID
0          ensembl_gene_id          Gene stable ID            Stable ID of the Gene ␣
↪hsapiens_gene_ensembl
1  ensembl_gene_id_version  Gene stable ID version  Versionned stable ID of the Gene ␣
↪hsapiens_gene_ensembl
2    ensembl_transcript_id    Transcript stable ID       Stable ID of the Transcript ␣
↪hsapiens_gene_ensembl
..                     ...                     ...                              ... ␣
↪                  ...
```

(continues on next page)

(continued from previous page)

```
3348            cds_length              CDS Length                          ␣
↪hsapiens_gene_ensembl
3349             cds_start               CDS start                          ␣
↪hsapiens_gene_ensembl
3350               cds_end                 CDS end                          ␣
↪hsapiens_gene_ensembl
```

A CLI command is also available to retrieve the same information: `apybiomart attributes`, whose `--dataset` option can be used to specify which dataset will be used (default is "hsapiens_gene_ensembl").

Filters

Datasets can be queried using filters that restrict the returned information to some specific subset of interest (e.g. chromosome, start position, etc.). In order to retrieve the list of filters available for a given dataset, the `find_filters()` function can be used:

```
from apybiomart import find_filters
find_filters("hsapiens_gene_ensembl")
# same as above, using the default dataset
find_filters()
```

This function accepts an optional `dataset` argument, which defaults to "hsapiens_gene_ensembl", and returns a dataframe with the `name`, `type`, `description` (where available) of each filter, as well as the `dataset` to which it belongs:

```
                           Filter_ID  Filter_type Filter_description          ␣
↪Dataset_ID
0               link_so_mini_closure         list                    hsapiens_gene_
↪ensembl
1                    link_go_closure         text                    hsapiens_gene_
↪ensembl
2  link_ensembl_transcript_stable_id         text                    hsapiens_gene_
↪ensembl
..                               ...          ...                ...              ␣
↪   ...
39        germ_line_variation_source         list                    hsapiens_gene_
↪ensembl
40          somatic_variation_source         list                    hsapiens_gene_
↪ensembl
42                so_consequence_name         list                    hsapiens_gene_
↪ensembl
```

A CLI command is also available to retrieve the same information: `apybiomart filters`, whose `--dataset` option can be used to specify which dataset will be used (default is "hsapiens_gene_ensembl").

4.3.2 Queries

Once the desired mart, dataset, attributes and filters have been explored (or if they were known beforehand), it is possible to query BioMart to retrieve the actual data; queries can be performed synchronously or asynchronously.

Exploring the difference between these two approaches is out of the scope of this document, but basically while in synchronous calls the client has to wait for a request to be complete before moving to the next one, in asynchronous calls the client can perform another request while the first one is idle, and so on until all the requests have been performed and a response was returned.

Simply put, apybiomart allows to perform synchronous queries to explore the data, and asynchronous queries to group multiple queries and run them efficiently.

Synchronous Queries

Synchronous queries can be performed using the `query()` function, which accepts `attributes` and `filters` arguments, and an optional `dataset` argument (which defaults to "hsapiens_gene_ensembl"):

```
from apybiomart import query
query(attributes=["ensembl_gene_id", "external_gene_name"],
      filters={"chromosome_name": "1"},
      dataset="hsapiens_gene_ensembl")
```

The `attributes` are provided as a list of properties, while `filters` are represented by a filter name : filter value dictionary. The returned dataframe contains the result of the query, restricted according to the provided filters and attributes.

Asynchronous Queries

Asynchronous queries can be performed using the `aquery()` function, which works just like `query()`, with the only difference that this is an async coroutine, so it needs to be handled properly taking advantage of the `asyncio` event loop:

```
import asyncio
from apybiomart import aquery
loop = asyncio.get_event_loop()
loop.run_until_complete(
    aquery(attributes=["ensembl_gene_id", "external_gene_name"],
           filters={"chromosome_name": "1"},
           dataset="hsapiens_gene_ensembl")
)
```

This allows to group multiple queries together, and the event loop will take care of scheduling them for execution:

```
import asyncio
from apybiomart import aquery
loop = asyncio.get_event_loop()
tasks = [aquery(attributes=["ensembl_gene_id", "external_gene_name"],
                filters={"chromosome_name": str(i)},
                dataset="hsapiens_gene_ensembl") for i in range(3)]
loop.run_until_complete(asyncio.gather(*tasks))
```

It is of course possible to assign the query results to one or more specific variables, for future usage:

```
# replacing last line of the previous code snippet
single_result = loop.run_until_complete(asyncio.gather(*tasks))
# or using multiple variables
chrom1, chrom2, chrom3 = loop.run_until_complete(asyncio.gather(*tasks))
```

Please refer to the asyncio documentation for more information.

4.4 API

4.4.1 Python Module

Entry points

These functions are available after you `import apybiomart` and should be used as the main entry points for apybiomart. If you want more control, you can use the internal classes described below.

async `apybiomart.apybiomart.`**`aquery`**(*attributes: List[str], filters: Dict[str, Union[str, int, list, tuple, bool]], dataset: str = 'hsapiens_gene_ensembl', save: bool = False, output: str = 'apybiomart_aquery.csv'*) → pandas.core.frame.DataFrame

Launch asynchronous query using the given attributes, filters and dataset.

Parameters

- **`attributes`** – list of attributes to include
- **`filters`** – dict of filter name : value to filter results
- **`dataset`** – BioMart dataset name (default: "hsapiens_gene_ensembl")
- **`save`** – save results to a CSV file [default: False]
- **`output`** – output filename if saving results [default: 'apybiomart_aquery.csv']

`apybiomart.apybiomart.`**`find_attributes`**(*dataset: str = 'hsapiens_gene_ensembl', save: bool = False, output: str = 'apybiomart_attributes.csv'*) → pandas.core.frame.DataFrame

Retrieve and list available attributes for a given mart.

Parameters

- **`dataset`** – BioMart dataset name (default: "hsapiens_gene_ensembl")
- **`save`** – save results to a CSV file [default: False]
- **`output`** – output filename if saving results [default: 'apybiomart_attributes.csv']

`apybiomart.apybiomart.`**`find_datasets`**(*mart: str = 'ENSEMBL_MART_ENSEMBL', save: bool = False, output: str = 'apybiomart_datasets.csv'*) → pandas.core.frame.DataFrame

Retrieve and list available datasets for a given mart.

Parameters

- **`mart`** – BioMart mart name (default: "ENSEMBL_MART_ENSEMBL")
- **`save`** – save results to a CSV file [default: False]
- **`output`** – output filename if saving results [default: 'apybiomart_datasets.csv']

`apybiomart.apybiomart.`**`find_filters`**(*dataset: str = 'hsapiens_gene_ensembl'*, *save: bool = False*, *output: str = 'apybiomart_filters.csv'*) → pandas.core.frame.DataFrame

Retrieve and list available filters for a given mart.

Parameters

- **`dataset`** – BioMart dataset name (default: "hsapiens_gene_ensembl")
- **`save`** – save results to a CSV file [default: False]
- **`output`** – output filename if saving results [default: 'apybiomart_filters.csv']

`apybiomart.apybiomart.`**`find_marts`**(*save: bool = False*, *output: str = 'apybiomart_marts.csv'*) → pandas.core.frame.DataFrame

Retrieve and list available marts.

Parameters

- **`save`** – save results to a CSV file [default: False]
- **`output`** – output filename if saving results [default: 'apybiomart_marts.csv']

`apybiomart.apybiomart.`**`query`**(*attributes: List[str]*, *filters: Dict[str, Union[str, int, list, tuple, bool]]*, *dataset: str = 'hsapiens_gene_ensembl'*, *save: bool = False*, *output: str = 'apybiomart_query.csv'*) → pandas.core.frame.DataFrame

Launch synchronous query using the given attributes, filters and dataset.

Parameters

- **`attributes`** – list of attributes to include
- **`filters`** – dict of filter name : value to filter results
- **`dataset`** – BioMart dataset name (default: "hsapiens_gene_ensembl")
- **`save`** – save results to a CSV file [default: False]
- **`output`** – output filename if saving results [default: 'apybiomart_query.csv']

Internal classes

These are the internal classes used by apybiomart, and can be imported with `from apybiomart.classes import <ClassName>`. Use them if you want more control over the application.

`class` `apybiomart.classes.`**`AttributesServer`**(*dataset: str*, *save: bool = False*, *output: str = 'apybiomart_attributes.csv'*)

Class used to retrieve and list available attributes for a dataset.

`dataset`

BioMart dataset name

`find_attributes``()` → pandas.core.frame.DataFrame

Return the list of available attributes for a specific dataset as a dataframe.

`class` `apybiomart.classes.`**`DatasetServer`**(*mart: str*, *save: bool = False*, *output: str = 'apybiomart_datasets.csv'*)

Class used to retrieve and list available datasets for a mart.

`mart`

BioMart mart name

`find_datasets``()` → pandas.core.frame.DataFrame

Return the list of available datasets for a specific mart as a dataframe.

class `apybiomart.classes.`**`FiltersServer`**(*dataset: str*, *save: bool = False*, *output: str = 'apybiomart_filters.csv'*)

Class used to retrieve and list available filters for a dataset.

`dataset`
BioMart dataset name

`find_filters()` → pandas.core.frame.DataFrame
Return the list of available filters for a specific dataset as a dataframe.

class `apybiomart.classes.`**`MartServer`**(*save: bool = False*, *output: str = 'apybiomart_marts.csv'*)

Class used to retrieve and list available marts.

`find_marts()` → pandas.core.frame.DataFrame
Return the list of available marts as a dataframe.

Returns pd.DataFrame

class `apybiomart.classes.`**`Query`**(*attributes: List[str]*, *filters: Dict[str, Union[str, int, list, tuple, bool]]*, *dataset: str*, *save: bool = False*, *output: str = 'apybiomart_query.csv'*)

Class used to perform either synchronous or asynchronous queries on BioMart.

`attributes`
list of attributes to include

`filters`
dict of filter name : value to filter results

`dataset`
BioMart dataset name

`async aquery()` → pandas.core.frame.DataFrame
Perform asynchronous query.

Return the result of the query based on the given attributes, filters and optional dataset using Server.get_async(), as a pandas DataFrame.

`query()` → pandas.core.frame.DataFrame
Perform synchronous query.

Return the result of the query based on the given attributes, filters and optional dataset using Server.get_sync(), as a pandas DataFrame.

4.4.2 Command Line Interface

4.5 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:

4.5.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/robertopreste/apybiomart/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.

Fix Bugs

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

Implement Features

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

Write Documentation

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

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/robertopreste/apybiomart/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 this is a volunteer-driven project, and that contributions are welcome :)

4.5.2 Get Started!

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

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

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

```
$ mkvirtualenv apybiomart
$ cd apybiomart/
$ python setup.py develop
```

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 flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 apybiomart tests
   $ python setup.py test  # or pytest
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

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.

4.5.3 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 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/robertopreste/apybiomart/pull_requests and make sure that the tests pass for all supported Python versions.

4.5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_apybiomart
```

4.5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch  # possible: major / minor / patch
$ git push
$ git push --tags
```

WIP: Travis will then deploy to PyPI if tests pass.

4.6 Credits

4.6.1 Development Lead

- Roberto Preste <robertopreste@gmail.com>

4.6.2 Contributors

None yet. Why not be the first?

4.7 History

4.7.1 0.1.0 (2019-03-26)

- First development release.

0.1.1 (2019-03-27)

- Requests are converted to async calls;
- Code style is clean and Python 3 compatible.

0.1.2 (2019-03-27)

- Add basic tests.

4.7.2 0.2.0 (2019-03-31)

- New version with different organisation of classes and functions;
- Sync `query` and async `aquery` functions to query Biomart;
- Sync `list_*` functions to retrieve available `marts`, `datasets`, `filters` and `attributes`.

0.2.1 (2019-04-01)

- Add tests.

0.2.2 (2019-04-01)

- Basic functions working and tested;
- Fix documentation;
- Update requirements.

0.2.3 (2019-04-02)

- Update requirements;
- Fix type hints for query functions;
- Reorganise query classes into a single class;
- Update documentation.

0.2.4 (2019-04-04)

- Fix type hints;
- Fix docstrings in classes;
- Add docstrings to main entry points.

0.2.5 (2019-04-09)

- Fix test files with new BioMart versions;
- Add script to create test files automatically.

0.2.6 (2019-04-29)

- Update test files;
- Fix and update documentation.

4.7.3 0.3.0 (2019-05-05)

- Change `list_*` functions names to `find_*` for better compliance;
- Update documentation.

0.3.1 (2019-05-11)

- Fix requirements handling;
- Add function to check internet connection.

0.3.2 (2019-05-29)

- Correct minor typos;
- Update documentation and testfiles.

0.3.3 (2019-07-29)

- Fix #37 - issue with the requests module not installed.

0.3.4 (2019-08-23)

- Better handling of filters arguments for `query()` and `aquery()` functions;
- Convert docstrings to Google style;
- Fix documentation.

0.3.5 (2019-08-25)

- Relax requirement versions.

4.7.4 0.4.0 (2020-01-26)

- Add CLI commands for finding marts, datasets, attributes and filters;
- Change output dataframe column names.

4.7.5 0.5.0 (2020-03-22)

- Add CLI and Python module options to save outputs to CSV file.

0.5.1 (2020-04-04)

- Add option to specify the output CSV filename.

0.5.2 (2020-06-06)

- Update tests and test files;
- Clean code;
- Add CI module.

0.5.3 (2020-11-30)

- Remove deprecated `pd.np` occurrencies;
- Update test files.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX