Roarquery

Adam Richie-Halford

May 01, 2024

CONTENTS

1 Features 3

2 Requirements 5

3 Installation 7

4 Usage 9

5 Contributing 11

6 License 13

7 Issues 15

8 Credits 17

Python Module Index 29

Index 31

CHAPTER

ONE

FEATURES

- Query ROAR runs
- Download ROAR runs and trials
- List ROAR Firestore collections

CHAPTER

TWO

REQUIREMENTS

- Python 3.9+
- fuego

CHAPTER

THREE

INSTALLATION

You can install *Roarquery* via pip from PyPI:

```
pip install roarquery
```

Roarquery also requires you to install *fuego*, a command line firestore client. Please see the fuego documentation for complete installation instructions.

On a Mac, follow these steps:

1. Ensure you have a working go installation. If

```
go version
```

returns something, then you are good to go. If not, install go with homebrew:

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/
↪install.sh)"
brew install go
```

2. Then install *fuego*

```
git clone https://github.com/sgarciac/fuego.git
cd fuego
go build .
go install .
```

3. Finally, modify your PATH variable to include the go installation directory, which can be done with the following incantation:

```
echo $HOME/go/bin | sudo tee -a /private/etc/paths.d/go
```

4. You may need to open a new terminal window or tab for these changes to take effect.

CHAPTER

FOUR

USAGE

4.1 Authentication

Before you can use *Roarquery*, you need to provide authentication details:

Roarquery works with both the current and legacy ROAR assessment databases. For example, the *roarquery runs* subcommand accepts a *–legacy* parameter to access the legacy database. If you would like to use roarquery with both databases, you will need to follow the steps below in both the legacy and current assessment Firebase projects.

1. Retrieve or generate a Service Account key file.
    a. go to your Firebase project console,
    b. go to "Project settings" (in the little gear menu next to "Project Overview"),
    c. click on the "Service accounts" tab,
    d. click on the "Generate new private key" button.
2. Save these files to somewhere on your computer. For example, presuming the previous commands downloaded the files to "$HOME/Downloads/private_key.json" and "$HOME/Downloads/legacy_private_key.json"

```
mkdir -p "$HOME/.firebaseconfig"
mv "$HOME/Downloads/private_key.json" "$HOME/.firebaseconfig/private_key.json"
mv "$HOME/Downloads/legacy_private_key.json" "$HOME/.firebaseconfig/legacy_private_
↪key.json"
```

3. Set the environment variable *ROAR_QUERY_CREDENTIALS* (or *ROAR_QUERY_LEGACY_CREDENTIALS* for the legacy database) to point to these files.

```
echo "export ROAR_QUERY_CREDENTIALS=\"$HOME/.firebaseconfig/private_key.json\"" >> ~
↪/.zprofile
echo "export ROAR_QUERY_CREDENTIALS=\"$HOME/.firebaseconfig/private_key.json\"" >> ~
↪/.bash_profile
echo "export ROAR_QUERY_LEGACY_CREDENTIALS=\"$HOME/.firebaseconfig/legacy_private_
↪key.json\"" >> ~/.zprofile
echo "export ROAR_QUERY_LEGACY_CREDENTIALS=\"$HOME/.firebaseconfig/legacy_private_
↪key.json\"" >> ~/.bash_profile
```

4.2 Command-line Usage

Please see the Command-line Reference for details.

CHAPTER

FIVE

CONTRIBUTING

Contributions are very welcome. To learn more, see the Contributor Guide.

CHAPTER

SIX

LICENSE

Distributed under the terms of the MIT license, *Roarquery* is free and open source software.

CHAPTER

SEVEN

ISSUES

If you encounter any problems, please file an issue along with a detailed description.

CHAPTER

EIGHT

CREDITS

This project was generated from @cjolowicz's Hypermodern Python Cookiecutter template.

8.1 Usage

8.1.1 roarquery

Roarquery.

Roarquery is a command-line interface for querying ROAR data in Google Cloud Firestore. It has several subcommands, which are listed below.

```
roarquery [OPTIONS] COMMAND [ARGS]...
```

Options

--version

Show the version and exit.

Useful definitions:
- trial: a single stimulus/response pair
- run: globally unique collection of successive trials.
  This constitute "running" through the task one time.
- corpus: A named and immutable collection of stimuli
- block: A portion of a run whose stimuli are drawn from only one corpus.
- task: the activity or assessment that was performed in a run (e.g. SWR, PA, SRE, etc.)
- task-variant: a specification of the task, e.g. adaptive vs. random; 1 vs 3 blocks
- school, district, class: all assume the standard meaning
- study: a collection of runs associated with a research project

runs

Return ROAR runs matching certain query parameters.

The options described below can be combined to return runs that match all of the specified query parameters.

b Arguments:

OUTPUT FILENAME Path to the output file to which to save runs/trials.

```
roarquery runs [OPTIONS] OUTPUT_FILENAME
```

Options

--legacy

Return trials from the legacy database

Default
`False`

--roar-uid <roar_uid>

Return only runs for the user with this ROAR UID.

--pid-prefix <pid_prefix>

Return only runs for users with this prefix.

--task-id <task_id>

Return only runs for this task.

--study-id <study_id>

Return only runs for this study. Only supported in the legacy database.

--variant-id <variant_id>

Return only runs for this variant.

--district-id <district_id>

Return only runs for this district.

--school-id <school_id>

Return only runs for this school.

--class-id <class_id>

Return only runs with this class.

--group-id <group_id>

Return only runs with this group. Only supported in the current database.

--require-completed

Require all runs to be completed.

Default
`False`

--started-before <started_before>

Return only runs started before this date. Format: YYYY-MM-DD.

--started-after <started_after>

Return only runs started after this date. Format: YYYY-MM-DD.

--return-trials

Return the trials for each run as well.

--root-doc <root_doc>

The Firestore root document. Returned runs will all be under this document.

Arguments

OUTPUT_FILENAME

Required argument

Examples:

Return trials for the "swr" task in the "validation" study.

```
roarquery runs --task-id=swr --study-id=validation --return-trials trials.csv
```

Return runs for the "sre" task in the "sd" district that started after 2021-05-10.

```
roarquery runs --task-id=sre --district-id=sd --started-after=2021-05-10 runs.
csv
```

8.2 Reference

8.2.1 roarquery.runs

Query and return ROAR runs.

roarquery.runs.**filter_run_dates**(*runs*, *started_before=None*, *started_after=None*)

Filter runs by date.

Parameters

- **runs** (*List[Dict[str, Any]]*) – The runs to filter.
- **started_before** (*date, optional, default=None*) – Return only runs started before this date.
- **started_after** (*date, optional, default=None*) – Return only runs started after this date.

Returns

The filtered runs.

Return type

List[Dict[str, Any]]

Examples

```
>>> runs = [
...     {
...         "CreateTime": "2020-01-01T00:00:00.000Z",
...         "Data": {
...             "name": "run-1",
...             "timeStarted": "2020-01-01T00:00:00.000Z",
...             "classId": "class-1",
...             "completed": "true",
...         },
...         "ID": "run-1",
...         "Path": "prod/roar-prod/runs/run-1",
...         "ReadTime": "2020-01-01T00:00:00.000Z",
...         "UpdateTime": "2020-01-01T00:00:00.000Z",
...     },
...     {
...         "CreateTime": "2020-02-01T00:00:00.000Z",
...         "Data": {
...             "name": "run-2",
...             "timeStarted": "2020-02-01T00:00:00.000Z",
...             "classId": "class-2",
...             "completed": "true",
...         },
...         "ID": "run-2",
...         "Path": "prod/roar-prod/runs/run-1",
...         "ReadTime": "2020-02-01T00:00:00.000Z",
...         "UpdateTime": "2020-02-01T00:00:00.000Z",
...     },
... ]
>>> filtered = filter_run_dates(runs, started_before=date(2020, 1, 15))
>>> print(filtered == [runs[0]])
True
```

roarquery.runs.**get_runs**(*return_trials=False*, *query_kwargs=None*, *started_before=None*, *started_after=None*, *user_type='users'*, *merge_user_info=True*)

Get all runs that satisfy a specific query.

Parameters

- **return_trials** (*bool, optional, default=False*) – If True, return the trials for each run as well.
- **query_kwargs** (*dict, optional, default=None*) – The query to run. If None, all runs will be returned.
- **started_before** (*date, optional, default=None*) – Return only runs started before this date.
- **started_after** (*date, optional, default=None*) – Return only runs started after this date.
- **user_type** (*str, optional, default="users"*) – The user type to query. Either "users" or "guests".
- **merge_user_info** (*bool, optional, default=True*) – If True, merge the user doc info into the run data.

Returns
The runs that satisfy the query.

Return type
List[dict]

roarquery.runs.**get_runs_compat**(*root_doc='prod/roar-prod'*, *return_trials=False*, *query_kwargs=None*, *started_before=None*, *started_after=None*, *merge_user_info=False*)

Get all runs that satisfy a specific query.

Parameters

- **root_doc** (*str, optional, default="prod/roar-prod"*) – The Firestore root document. The returned runs will all be under this document.
- **return_trials** (*bool, optional, default=False*) – If True, return the trials for each run as well.
- **query_kwargs** (*dict, optional, default=None*) – The query to run. If None, all runs will be returned.
- **started_before** (*date, optional, default=None*) – Return only runs started before this date.
- **started_after** (*date, optional, default=None*) – Return only runs started after this date.
- **merge_user_info** (*bool, optional, default=False*) – If True, merge the user doc info into the run data.

Returns
The runs that satisfy the query.

Return type
List[dict]

roarquery.runs.**get_trials_from_run**(*run_path*)

Get all trials from a run.

Parameters
run_path (*str*) – The Firestore path to the run.

Returns
The trials from the run.

Return type
List[Dict[str, str]]

roarquery.runs.**get_user_from_run**(*run_path*, *legacy=False*)

Get the user that owns a run.

Parameters

- **run_path** (*str*) – The Firestore path to the run.
- **legacy** (*bool, optional*) – If True, the returned user will be identified by PID, otherwise the user will be identified by roarUid. Default: False.

Returns
The user that owns the run.

Return type
List[Dict[str, str]]

roarquery.runs.**merge_data_with_metadata**(*fuego_response*, *metadata_params*)

Merge trial data with metadata.

We often want to merge the run/trial data with some of the metadata returned by Firestore. In Python 3.9 we could use the | operator but we want to be backward compatible so we iterate over the data and metadata and merge them together.

Parameters

- **fuego_response** (*List[Dict[str, Any]]*) – The trial data.
- **metadata_params** (*Dict[str, str]*) – The metadata fields that will be merged into the data. The keys are the desired keys in the merged data and the values are the metadata keys to use.

Returns
The merged data.

Return type
List[Dict[str, Any]]

8.2.2 roarquery.collections

Query Firestore collections.

roarquery.collections.**get_collections**()

Get collections from a database.

Return type
List[str]

8.2.3 roarquery.utils

Utilities functions.

roarquery.utils.**bytes2json**(*bytes*)

Convert bytes to json.

Parameters
bytes (*bytes*) – The bytes to convert.

Returns
The converted json.

Return type
List[_FuegoResponse]

Examples

An empty string will return an empty list: >>> bytes2json(b"") []

```
>>> json_out = bytes2json(
...     b'[{"ID": "1", "Data": {"a": "b"}, "Path": "prod/roar-prod", '
...     b'"CreateTime": "2020-04-01T00:00:00Z", '
...     b'"ReadTime": "2020-04-01T00:00:00Z", '
...     b'"UpdateTime": "2020-04-01T00:00:00Z"}]'
```

(continues on next page)

(continued from previous page)

```
... )
>>> print(json_out == [{
...     'ID': '1',
...     'Data': {'a': 'b'},
...     'Path': 'prod/roar-prod',
...     'CreateTime': '2020-04-01T00:00:00Z',
...     'ReadTime': '2020-04-01T00:00:00Z',
...     'UpdateTime': '2020-04-01T00:00:00Z'
... }])
True
```

roarquery.utils.**camel_case**(*string*)

Convert a string to camel case.

Parameters
string (*str*) – The string to convert.

Returns
The camel case string.

Return type
str

Examples

```
>>> camel_case("an_example_string")
anExampleString
```

```
>>> camel_case("an-example-string-with-dashes")
anExampleStringWithDashes
```

roarquery.utils.**drop_empty**(*iterable*)

Drop empty strings from a list.

Parameters
iterable (*list*) – The list to drop empty strings from.

Returns
The list with empty strings dropped.

Return type
list

Examples

```
>>> drop_empty(["", "a", "", "b"])
['a', 'b']
```

roarquery.utils.**page_results**(*query*, *limit=None*)

Page through results from a query.

Parameters

- **query** (*List[str]*) – The query to run. This is a list of strings that will be passed to subprocess.check_output.
- **limit** (*int, optional, default=100*) – The number of results to return per page.

Returns
The results of the query.

Return type
List[_FuegoResponse]

roarquery.utils.**trim_doc_path**(*path*)

Remove leading project information from firestore document path.

Parameters
path (*str*) – The path to standardize.

Returns
The standardized path.

Return type
str

Examples

```
>>> trim_doc_path("projects/proj-id/databases/(default)/documents/prod/roar-prod")
prod/roar-prod
```

8.3 Contributor Guide

Thank you for your interest in improving this project. This project is open-source under the MIT license and welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- Source Code
- Documentation
- Issue Tracker
- Code of Conduct

8.3.1 How to report a bug

Report bugs on the Issue Tracker.

When filing an issue, make sure to answer these questions:

- Which operating system and Python version are you using?
- Which version of this project are you using?
- What did you do?
- What did you expect to see?
- What did you see instead?

The best way to get your bug fixed is to provide a test case, and/or steps to reproduce the issue.

8.3.2 How to request a feature

Request features on the Issue Tracker.

8.3.3 How to set up your development environment

You need Python 3.9+ and the following tools:

- Poetry
- Nox
- nox-poetry

Install the package with development requirements:

```
$ poetry install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run roarquery
```

8.3.4 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the `tests` directory, and are written using the pytest testing framework.

8.3.5 How to submit changes

Open a pull request to submit changes to this project.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains 100% code coverage.
- If your changes add functionality, update the documentation accordingly.

Feel free to submit early, though—we can always iterate on this.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.

8.4 Contributor Covenant Code of Conduct

8.4.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

8.4.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

8.4.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

8.4.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

8.4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at richiehalford@gmail.com. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

8.4.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

8.4.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

8.5 MIT License

PYTHON MODULE INDEX

r

roarquery.collections, 22
roarquery.runs, 19
roarquery.utils, 22

INDEX

Symbols

--class-id
    roarquery-runs command line option, 18
--district-id
    roarquery-runs command line option, 18
--group-id
    roarquery-runs command line option, 18
--legacy
    roarquery-runs command line option, 18
--pid-prefix
    roarquery-runs command line option, 18
--require-completed
    roarquery-runs command line option, 18
--return-trials
    roarquery-runs command line option, 18
--roar-uid
    roarquery-runs command line option, 18
--root-doc
    roarquery-runs command line option, 19
--school-id
    roarquery-runs command line option, 18
--started-after
    roarquery-runs command line option, 18
--started-before
    roarquery-runs command line option, 18
--study-id
    roarquery-runs command line option, 18
--task-id
    roarquery-runs command line option, 18
--variant-id
    roarquery-runs command line option, 18
--version
    roarquery command line option, 17

B

bytes2json() (*in module roarquery.utils*), 22

C

camel_case() (*in module roarquery.utils*), 23

D

drop_empty() (*in module roarquery.utils*), 23

F

filter_run_dates() (*in module roarquery.runs*), 19

G

get_collections() (*in module roarquery.collections*), 22
get_runs() (*in module roarquery.runs*), 20
get_runs_compat() (*in module roarquery.runs*), 21
get_trials_from_run() (*in module roarquery.runs*), 21
get_user_from_run() (*in module roarquery.runs*), 21

M

merge_data_with_metadata() (*in module roarquery.runs*), 21
module
    roarquery.collections, 22
    roarquery.runs, 19
    roarquery.utils, 22

O

OUTPUT_FILENAME
    roarquery-runs command line option, 19

P

page_results() (*in module roarquery.utils*), 23

R

roarquery command line option
    --version, 17
roarquery.collections
    module, 22
roarquery.runs
    module, 19
roarquery.utils
    module, 22
roarquery-runs command line option
    --class-id, 18
    --district-id, 18
    --group-id, 18
    --legacy, 18

--pid-prefix, 18
--require-completed, 18
--return-trials, 18
--roar-uid, 18
--root-doc, 19
--school-id, 18
--started-after, 18
--started-before, 18
--study-id, 18
--task-id, 18
--variant-id, 18
OUTPUT_FILENAME, 19

T

trim_doc_path() (*in module roarquery.utils*), 24