renga Documentation

Release 0.1.0.dev20171123

Swiss Data Science Center

Apr 27, 2018

Contents

A Python library for the Renga collaborative data science platform. It allows the user to create projects, manage datasets, and capture data provenance while performing analysis tasks.

NOTE: `renga-python` is the python library for Renga that provides an SDK and a command-line interface (CLI). It *does not* start the Renga platform itself - for that, refer to the Renga docs on running the platform.

This is the development branch of 'renga-python' and should be considered highly volatile. The documentation for certain components may be out of sync.

CHAPTER 1

Installation

The latest release is available on PyPI and can be installed using `pip`:

```
$ pip install renga
```

The development branch can be installed directly from the Git repository:

```
$ pip install -e git+https://github.com/SwissDataScienceCenter/renga-python.
↪git@development#egg=renga
```

For more information about the Renga API see its documentation.

CHAPTER 2

Use the Renga command line

Interaction with the platform can take place via the command-line interface (CLI).

Start by creating for folder where you want to keep your Renga project:

```
$ mkdir -p ~/temp/my-renga-project
$ cd ~/temp/my-renga-project
$ renga init
```

Create a dataset and add data to it:

```
$ renga dataset create my-dataset
$ renga dataset add my-dataset https://raw.githubusercontent.com/
↪SwissDataScienceCenter/renga-python/development/README.rst
```

Run an analysis:

```
$ renga run wc < data/my-dataset/README.rst > wc_readme
```

Trace the data provenance:

```
$ renga log wc_readme
```

These are the basics, but there is much more that Renga allows you to do with your data analysis workflows.

For more information about using *renga*, refer to the *Renga command line* instructions.

2.1 Renga Command Line

The base command for interacting with the Renga platform.

2.1.1 `renga` (base command)

To list the available commands, either run `renga` with no parameters or execute `renga help`:

```
$ renga help
Usage: renga [OPTIONS] COMMAND [ARGS]...

Check common Renga commands used in various situations.

Options:
  --version             Print version number.
  --config PATH         Location of client config files.
  --config-path         Print application config path.
  --path <path>         Location of a Renga repository.  [default: .]
  --renga-home <path>   Location of Renga directory.  [default: .renga]
  -h, --help            Show this message and exit.

Commands:
  # [...]
```

Configuration files

Depending on your system, you may find the configuration files used by Renga command line in a different folder. By default, the following rules are used:

MacOS: `~/Library/Application Support/Renga`

Unix: `~/.config/renga`

Windows: `C:\Users\<user>\AppData\Roaming\Renga`

If in doubt where to look for the configuration file, you can display its path by running `renga --config-path`.

You can specify a different location via the `RENGA_CONFIG` environment variable or the `--config` command line option. If both are specified, then the `--config` option value is used. For example:

```
$ renga --config ~/renga/config/ init
```

instructs Renga to store the configuration files in your `~/renga/config/` directory when running the `init` command.

2.1.2 `renga init`

Create an empty Renga project or reinitialize an existing one.

Starting a Renga project

If you have an existing directory which you want to turn into a Renga project, you can type:

```
$ cd ~/my_project
$ renga init
```

or:

```
$ renga init ~/my_project
```

This creates a new subdirectory named `.renga` that contains all the necessary files for managing the project configuration.

2.1.3 renga datasets

Work with datasets in the current repository.

Manipulating datasets

Creating an empty dataset inside a Renga project:

```
$ renga dataset create my-dataset
```

Adding data to the dataset:

```
$ renga dataset add my-dataset http://data-url
```

This will copy the contents of `data-url` to the dataset and add it to the dataset metadata.

2.1.4 renga run

Track provenance of data created by executing programs.

2.1.5 renga log

Show provenance of data created by executing programs.

2.1.6 renga workflow

Workflow operations.

2.2 Projects

Model objects representing projects.

class renga.models.projects.**Project**(*name=None*, *created=NOTHING*, *updated=NOTHING*, *version='1'*)

Represent a project.

Type:

```
"foaf:Project"
```

Context:

```
{
  "name": "foaf:name",
  "updated": "http://schema.org/dateUpdated",
  "version": "http://schema.org/schemaVersion",
  "created": "http://schema.org/dateCreated",
```

```
  "foaf": "http://xmlns.com/foaf/0.1/"
}
```

class renga.models.projects.**ProjectCollection**(*client=None*)

Represent projects on the server.

Example

Create a project and check its name.

>>> project = client.projects.create(name='test-project') # >>> project.name # 'test-project'

Create a representation of objects on the server.

class Meta

Information about individual projects.

model

alias of *Project*

create(*name=None*, ***kwargs*)

Create a new project.

Parameters **name** – The name of the project.

Returns An instance of the newly create project.

Return type *Project*

2.3 Datasets

Manage datasets and their metadata.

2.3.1 Dataset object

class renga.models.datasets.**Dataset**(*name*, *created=NOTHING*, *identifier=NOTHING*, *authors=NOTHING*, *files=NOTHING*)

Repesent a dataset.

Type:

```
"dctypes:Dataset"
```

Context:

```
{
  "name": "dcterms:name",
  "affiliation": "scoro:affiliate",
  "added": "http://schema.org/dateCreated",
  "created": "http://schema.org/dateCreated",
  "prov": "http://www.w3.org/ns/prov#",
  "email": "dcterms:email",
  "dctypes": "http://purl.org/dc/dcmitypes/",
  "scoro": "http://purl.org/spar/scoro/",
  "identifier": {
    "@id": "dctypes:Dataset",
    "@type": "@id"
  },
```

```
  "url": "http://schema.org/url",
  "files": {
    "@container": "@index"
  },
  "authors": {
    "@container": "@list"
  },
  "dcterms": "http://purl.org/dc/terms/",
  "foaf": "http://xmlns.com/foaf/0.1/"
}
```

from_jsonld(*data*)
Instantiate a JSON-LD class from data.

2.3.2 Dataset file

Manage files in the dataset.

class renga.models.datasets.**DatasetFile**(*path*, *url=None*, *authors=NOTHING*, *dataset=None*, *added=NOTHING*)
Represent a file in a dataset.

Type:

```
"http://schema.org/DigitalDocument"
```

Context:

```
{
  "name": "dcterms:name",
  "affiliation": "scoro:affiliate",
  "added": "http://schema.org/dateCreated",
  "scoro": "http://purl.org/spar/scoro/",
  "email": "dcterms:email",
  "authors": {
    "@container": "@list"
  },
  "dcterms": "http://purl.org/dc/terms/",
  "url": "http://schema.org/url",
  "foaf": "http://xmlns.com/foaf/0.1/"
}
```

from_jsonld(*data*)
Instantiate a JSON-LD class from data.

2.3.3 Author

class renga.models.datasets.**Author**(*name*, *email*, *affiliation=None*)
Represent the author of a resource.

Type:

```
"dcterms:creator"
```

Context:

```
{
  "name": "dcterms:name",
  "affiliation": "scoro:affiliate",
  "email": "dcterms:email",
  "dcterms": "http://purl.org/dc/terms/",
  "scoro": "http://purl.org/spar/scoro/",
  "foaf": "http://xmlns.com/foaf/0.1/"
}
```

check_email(*attribute*, *value*)
Check that the email is valid.

from_commit(*commit*)
Create an instance from a Git commit.

from_git(*git*)
Create an instance from a Git repo.

from_jsonld(*data*)
Instantiate a JSON-LD class from data.

2.4 Tools and Workflows

Manage creation of tools and workflows using the Common Workflow Language (CWL).

2.4.1 Common Workflow language

Renga uses CWL to represent runnable steps (tools) along with their inputs and outputs. Similarly, tools can be chained together to form CWL-defined workflows.

Command-line tool

Represent a `CommandLineTool` from the Common Workflow Language.

class renga.models.cwl.command_line_tool.**CommandLineTool**(*requirements=NOTHING*, *hints=NOTHING*, *label=None*, *doc=None*, *cwlVersion='v1.0'*, *baseCommand=''*, *arguments=NOTHING*, *stdin=None*, *stdout=None*, *stderr=None*, *inputs=NOTHING*, *outputs=NOTHING*, *successCodes=NOTHING*, *temporaryFailCodes=NOTHING*, *permanentFailCodes=NOTHING*)

Represent a command line tool.

get_output_id(*path*)
Return an id of the matching path from default values.

to_argv(*job=None*)
Generate arguments for system call.

class renga.models.cwl.command_line_tool.**CommandLineToolFactory**(*command_line*, *directory='.'*, *stdin=None*, *stderr=None*, *stdout=None*)
Command Line Tool Factory.

file_candidate(*candidate*)
Return a path instance if it exists in current directory.

generate_tool()
Return an instance of command line tool.

guess_inputs(**arguments*)
Yield command input parameters and command line bindings.

guess_outputs(*paths*)
Yield detected output and changed command input parameter.

guess_type(*value*)
Return new value and CWL parameter type.

split_command_and_args()
Return tuple with command and args from command line arguments.

validate_command_line(*attribute*, *value*)
Check the command line structure.

validate_path(*attribute*, *value*)
Path must exists.

watch(*repo=None*, *no_output=False*)
Watch a Renga repository for changes to detect outputs.

Parameter

Represent parameters from the Common Workflow Language.

class renga.models.cwl.parameter.**CommandInputParameter**(*id*, *streamable=None*, *type='string'*, *description=None*, *default=None*, *inputBinding=None*)
An input parameter for a CommandLineTool.

to_argv()
Format command input parameter as shell argument.

class renga.models.cwl.parameter.**CommandLineBinding**(*position=None*, *prefix=None*, *separate=True*, *itemSeparator=None*, *valueFrom=None*, *shellQuote=True*)
Define the binding behavior when building the command line.

to_argv(*default=None*)
Format command line binding as shell argument.

class renga.models.cwl.parameter.**CommandOutputBinding**(*glob=None*)
Define the binding behavior for outputs.

class renga.models.cwl.parameter.**CommandOutputParameter**(*id*, *streamable=None*, *type='string'*, *description=None*, *format=None*, *outputBinding=None*)

Define an output parameter for a CommandLineTool.

class renga.models.cwl.parameter.**InputParameter**(*id*, *streamable=None*, *type='string'*, *description=None*, *default=None*, *inputBinding=None*)

An input parameter.

class renga.models.cwl.parameter.**OutputParameter**(*id*, *streamable=None*, *type='string'*, *description=None*, *format=None*, *outputBinding=None*)

An output parameter.

class renga.models.cwl.parameter.**Parameter**(*streamable=None*)

Define an input or output parameter to a process.

class renga.models.cwl.parameter.**WorkflowOutputParameter**(*id*, *streamable=None*, *type='string'*, *description=None*, *format=None*, *outputBinding=None*, *outputSource=None*)

Define an output parameter for a Workflow.

renga.models.cwl.parameter.**convert_default**(*value*)

Convert a default value.

Process

Represent a `Process` from the Common Workflow Language.

class renga.models.cwl.process.**Process**

Represent a process.

Types

Represent the Common Workflow Language types.

class renga.models.cwl.types.**File**(*path*)

Represent a file.

Workflow

Represent workflows from the Common Workflow Language.

class renga.models.cwl.workflow.**Workflow**(*inputs=NOTHING*, *requirements=NOTHING*, *hints=NOTHING*, *label=None*, *doc=None*, *cwlVersion='v1.0'*, *outputs=NOTHING*, *steps=NOTHING*)

Define a workflow representation.

add_step(***kwargs*)

Add a workflow step.

get_output_id(*path*)
Return an id of the matching path from default values.

class renga.models.cwl.workflow.**WorkflowStep**(*run*, *id=NOTHING*, *in_=None*, *out=None*)
Define an executable element of a workflow.

2.5 Client

2.5.1 Creating a client

There are several ways to instantiate a client used for communication with the Renga platform.

1. The easiest way is by calling the function *from_env()* when running in an environment created by the Renga platform itself.
2. The client can be created from a local configuration file by calling *from_config()*.
3. Lastly, it can also be configured manually by instantiating a *RengaClient* class.

renga.client.**from_env**()
Return a client configured from environment variables.

RENGA_ENDPOINT
The URL to the Renga platform.

RENGA_ACCESS_TOKEN
An access token obtained from Renga authentication service.

Example:

```
>>> import renga
>>> client = renga.from_env()
```

renga.cli._client.**from_config**()
Create a new client for endpoint in the config.

Use `renga` command-line interface to manage multiple configurations.

2.5.2 Client reference

class renga.client.**RengaClient**
A client for communicating with a Renga platform.

Example:

```
>>> import renga
>>> client = renga.RengaClient('http://localhost')
```

Create a Renga API client.

2.6 Low-level API

This API is built on top of REST API endpoints exposed by Renga services.

Warning: Renga services are currently in **beta preview** status and they are subject to change in forseenable future.

HTTP clients for Renga platform.

class renga.api.**APIClient**(*endpoint=None*, ***kwargs*)
A low-level client for communicating with a Renga Platform API.

Example:

```
>>> import renga
>>> client = renga.APIClient('http://localhost')
```

Create a remote API client.

delete(**args*, ***kwargs*)
Perform the DELETE request and check its status code.

endpoint
Return endpoint value.

get(**args*, ***kwargs*)
Perform the GET request and check its status code.

post(**args*, ***kwargs*)
Perform the POST request and check its status code.

put(**args*, ***kwargs*)
Perform the PUT request and check its status code.

class renga.api.**LocalClient**(*renga_home='.renga'*, *datadir='data'*, *path=NOTHING*)
A low-level client for communicating with a local Renga repository.

Example:

```
>>> import renga
>>> client = renga.LocalClient('.')
```

2.6.1 Projects

Client for handling projects.

class renga.api.projects.**ProjectsApiMixin**
Client for handling projects.

create_project(*project*)
Create a new project and register it on the knowledge graph.

get_project(*project_id*)
Get existing project.

list_projects()
Return an iterator for all projects.

2.6.2 Storage

Client for storage service.

class renga.api.storage.**BucketsApiMixin**
Client for handling storage buckets.

create_bucket(***kwargs*)
Create a new storage bucket.

storage_bucket_metadata_replace(*resource_id*, *data*)
Replace resource metadata.

storage_info()
Return information about available bucket backends.

class renga.api.storage.**FilesApiMixin**
Client for handling file objects in a bucket.

create_file(***kwargs*)
Create a new file object.

storage_authorize(*resource_id=None*, *request_type=None*)
Request authorization token for performing file handle request.

storage_copy_file(*resource_id=None*, *file_name=None*, ***kwargs*)
Request a file copy.

storage_file_metadata_replace(*resource_id*, *data*)
Replace resource metadata.

storage_io_read(**args*, ***kwargs*)
Write data to the file.

Note: Use only with `access_token` issued by storage service.

storage_io_write(*data*)
Write data to the file.

Note: Use only with `access_token` issued by storage service.

2.6.3 Deployer

Client for deployer service.

class renga.api.deployer.**ContextsApiMixin**
Manage deployer contexts.

create_context(*spec*)
Create a new deployer context.

create_execution(*context_id*, ***kwargs*)
Create an execution of a context on a given engine.

execution_logs(*context_id*, *execution_id*)
Retrieve logs of an execution.

execution_ports(*context_id*, *execution_id*)
Retrieve port mappings for an execution.

get_context(*context_id*)
List all known contexts.

get_execution(*context_id*, *execution_id*)
Retrieve an execution.

list_contexts()
List all known contexts.

list_executions(*context_id*)
List all executions of a given context.

stop_execution(*context_id*, *execution_id*)
Stop a running execution.

2.7 Contributing

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

2.7.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/SwissDataScienceCenter/renga-python/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" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Renga could always use more documentation, whether as part of the official Renga 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/SwissDataScienceCenter/renga-python/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 :)

2.7.2 Get Started!

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

1. Fork the *SwissDataScienceCenter/renga-python* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/renga.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 renga
$ cd renga/
$ pip install -e .[all]
```

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 tests:

```
$ ./run-tests.sh
```

   The tests will provide you with test coverage and also check PEP8 (code style), PEP257 (documentation), flake8 as well as build the Sphinx documentation and run doctests.

   Before you submit a pull request, please reformat the code using yapf.

```
$ yapf -irp .
```

   You may want to set up yapf styling as a pre-commit hook to do this automatically:

```
$ curl https://raw.githubusercontent.com/google/yapf/master/plugins/pre-commit.sh␣
↪-o .git/hooks/pre-commit
$ chmod u+x .git/hooks/pre-commit
```

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

```
$ git add .
$ git commit -s
    -m "component: title without verbs"
    -m "* NEW Adds your new feature."
    -m "* FIX Fixes an existing issue."
    -m "* BETTER Improves and existing feature."
    -m "* Changes something that should not be visible in release notes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

2.7.3 Pull Request Guidelines

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

1. The pull request should include tests and must not decrease test coverage.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring.
3. The pull request should work for Python 2.7, 3.5, and 3.6. Check https://travis-ci.org/SwissDataScienceCenter/renga-python/pull_requests and make sure that the tests pass for all supported Python versions.

2.8 Changes

Version 0.1.0 (released TBD)

- Initial public release.

2.9 License

2.10 Authors

Python SDK and CLI for the Renga platform.

- Swiss Data Science Center <contact@datascience.ch>

Python Module Index

r

Index

I

L

M

O

P

R

S

T

V

W