floww Documentation

Release 0.5.2

Matthew Spellings

Sep 01, 2022

Contents

1 Introduction **3**
- 1.1 Installation . . . 3
- 1.2 Documentation . . . 3
- 1.3 Examples . . . 4

2 Indices and tables **9**

Python Module Index **11**

Index **13**

CHAPTER 1

Introduction

`flowws` is an in-development framework for building modular, reusable task workflows. The core library contains tools to abstract over storage locations and parse arguments in a uniform way for both python scripts and command-line-based execution. It is designed to help solve the following problems:

- Composing tasks from a series of modular actions
- Parameterizing tasks and exposing interfaces for both interactive and batch execution
- Improving reproducibility by encapsulating parameters within workflow definitions

`flowws` is being developed in conjunction with other projects, including:

- hoomd-flowws: perform simulations with hoomd-blue.
- flowws-analysis: run analysis and visualization workflows
- flowws-freud: molecular simulation-specific modules for `flowws-analysis`
- flowws-examples: example workflows using the above projects

1.1 Installation

Install flowws from PyPI:

```
pip install flowws
```

Alternatively, install from source:

```
pip install git+https://github.com/klarh/flowws.git#egg=flowws
```

1.2 Documentation

Browse more detailed documentation online or build the sphinx documentation from source:

```
git clone https://github.com/klarh/flowws
cd flowws/doc
pip install -r requirements.txt
make html
```

1.3 Examples

The flowws-examples project contains interactive notebook examples that demonstrate various workflows.

1.3.1 Workflows and Stages

class flowws.**Workflow**(*stages*, *storage=None*, *scope={}*)

Specify a complete sequence of operations to perform.

Workflow objects specify a sequence of stages (operations to perform) and a storage object to use (which could be a database, archive file, or simply a directory on the filesystem). In addition to direct creation within python, Workflows can be deserialized from command line and JSON-based descriptions.

Stages are executed sequentially in the order they are given and each stage can pass information to later stages in a freeform way by settings elements of a *scope*, which is a dictionary of named values.

Parameters

- **stages** – List of *Stage* objects specifying the operations to perform
- **storage** – *Storage* object specifying where results should be saved (default: create a DirectoryStorage using the current working directory)
- **scope** – Dictionary of key-value pairs specifying external input parameters

classmethod from_JSON(*json_object*, *module_names='flowws_modules'*)

Construct a Workflow from a JSON object.

classmethod from_command(*args=None*, *module_names='flowws_modules'*, *scope={}*)

Construct a Workflow from a command-line description.

Stages are found based on setuptools entry_point specified under *module_names*.

Parameters

- **args** – List of command-line arguments (list of strings)
- **module_names** – setuptools entry_point to use for module searches
- **scope** – Dictionary of initial key-value pairs to pass to child Stages

classmethod register_module(**args*, *module_names='flowws_modules'*, *name=None*)

Register a named module to be loaded inside *from_JSON* or other functions.

This method is intended to be used as a decorator for Stage classes in situations such as REPL loops or notebooks, where modules need to be deserialized without necessarily creating a standalone package and registering the endpoints through the setuptools machinery.

Examples:

```
@flowws.Workflow.register_module
class TestStage(flowws.Stage):
    pass
```

(continues on next page)

(continued from previous page)

```
@flowws.Workflow.register_module(name='OverruledName')
class Stage(flowws.Stage):
    pass
```

run()
Run each stage inside this workflow.

Returns the scope after running all stages.

class flowws.**Stage**(***kwargs*)
Base class for the building blocks of workflows.

Stage objects specify a discrete set of operations within a Workflow. Each Stage object has its own set of parameters and functionality that are then run in sequence when the workflow is run.

Stages can be instantiated within python by directly passing in arguments they take as keyword arguments, for example:

```
stages = [Initialize(seed=13), Run(parameter=1.5)]
```

Stages also can be instantiated from the command line using *flowws.run* (assuming they have been properly registered using setuptools *entry_points*):

```
python -m flowws.run Initialize --seed 13 Run --parameter 1.5
```

classmethod from_JSON(*json_object*)
Initialize this stage from a JSON representation

classmethod from_command(*args*)
Initialize this stage from a command-line description

run(*scope*, *storage*)
Run the contents of this stage

flowws.**register_module**(**args*, *module_names='flowws_modules'*, *name=None*)
Register a named module to be loaded inside *from_JSON* or other functions.

This method is intended to be used as a decorator for Stage classes in situations such as REPL loops or notebooks, where modules need to be deserialized without necessarily creating a standalone package and registering the endpoints through the setuptools machinery.

Examples:

```
@flowws.Workflow.register_module
class TestStage(flowws.Stage):
    pass

@flowws.Workflow.register_module(name='OverruledName')
class Stage(flowws.Stage):
    pass
```

flowws.**try_to_import**(*pkg*, *name*, *current_pkg=None*)
Import an attribute from a module, or return an error-producing fake.

This method is provided as a convenience for libraries that want to easily expose modules with a variety of prerequisite libraries without forcing the user to install prerequisites for the modules they do not use. The fake is produced if an import fails while importing the given package.

Parameters

- **name** – Name of the attribute to return from the module
- **pkg** – Package name (can be relative)
- **current_pkg** – Name of the current package to use (i.e. if *pkg* is relative)

Returns Either the attribute from the successfully-imported module, or a fake module object that will produce an error if evaluated

1.3.2 Command Line Utilities

flowws.run

Directly run a user-defined workflow from the command line

The *flowws.run* utility is used to execute a workflow from a brief, text-only description. It works by finding modules installed using a particular setuptools *entry_point*, which each parse their own command-line parameters in their own way. Automatically-generated documentation can be accessed in the standard way for *flowws.run* via:

```
python -m flowws.run -h
```

Automatically-generated documentation for any module (for this example, simply named Module) as:

```
python -m flowws.run Module -h
```

A complete workflow specification using modules Module1 and Module2 may look something like this:

```
python -m flowws.run Module1 --param-1 x --param-2 y Module2
```

JSON workflows defined by *flowws.freeze* can also be executed using *flowws.run*:

```
python -m flowws.run workflow.json
```

A *flowws_run* script is also installed for this command for convenience.

flowws.freeze

Save a user-defined workflow from the command line for later execution

The *flowws.freeze* utility is used to store a workflow description in JSON form. It finds and parameterizes modules identically to *flowws.run*, but saves the result to a file to run later rather than immediately executing the workflow. Before the workflow definition, it takes a single argument specifying the location to store the resulting JSON file:

```
python -m flowws.freeze workflow.json Module1 Module2
```

A *flowws_freeze* script is also installed for this command for convenience.

1.3.3 Storage API

class flowws.Storage.**Storage**

Base class for file storage.

Storage objects expose methods for reading and writing of files which could actually be backed by a database or archive file, for example.

open(*filename*, *mode='r'*, *modifiers=[]*, *on_filesystem=False*, *noop=False*)
Open a file stored within this object.

Parameters

- **filename** – Name of the (internal) file
- **mode** – One of 'r' (read), 'w' (write/overwrite), 'a' (append) and, optionally, 'b' (open in binary mode)
- **modifiers** – List of filename modifiers which will be appended to the filename, respecting the file suffix
- **on_filesystem** – If True, the file must exist as a real file on the filesystem; otherwise, a python stream object may be returned
- **noop** – If True, return a dummy file object instead that does nothing

open_file(*full_name*, *mode*)
Open a file stored within this object as a real file on the filesystem.

The default implementation simply copies a stream object onto the filesystem.

open_stream(*full_name*, *mode*)
Open a file stored within this object as a stream.

class flowws.**DirectoryStorage**(*root='.'*, *group=None*)
Stores files directly on the filesystem.

open_file(*full_name*, *mode*)
Open a file stored within this object as a real file on the filesystem.

The default implementation simply copies a stream object onto the filesystem.

open_stream(*full_name*, *mode*)
Open a file stored within this object as a stream.

class flowws.**GetarStorage**(*target*, *group=None*)
Class to store files as records of getar-format files.

These can be zip, tar, or sqlite-formatted archives. Note that zip and tar files will currently accumulate copies of files as they are appended to or overwritten.

open_stream(*full_name*, *mode*)
Open a file stored within this object as a stream.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

flowws.freeze, 6
flowws.run, 6

Index

D

DirectoryStorage (*class in flowws*), 7

F

flowws.freeze (*module*), 6
flowws.run (*module*), 6
from_command() (*flowws.Stage class method*), 5
from_command() (*flowws.Workflow class method*), 4
from_JSON() (*flowws.Stage class method*), 5
from_JSON() (*flowws.Workflow class method*), 4

G

GetarStorage (*class in flowws*), 7

O

open() (*flowws.Storage.Storage method*), 6
open_file() (*flowws.DirectoryStorage method*), 7
open_file() (*flowws.Storage.Storage method*), 7
open_stream() (*flowws.DirectoryStorage method*), 7
open_stream() (*flowws.GetarStorage method*), 7
open_stream() (*flowws.Storage.Storage method*), 7

R

register_module() (*flowws.Workflow class method*), 4
register_module() (*in module flowws*), 5
run() (*flowws.Stage method*), 5
run() (*flowws.Workflow method*), 5

S

Stage (*class in flowws*), 5
Storage (*class in flowws.Storage*), 6

T

try_to_import() (*in module flowws*), 5

W

Workflow (*class in flowws*), 4