wft4galaxy Documentation

Release 0.1

Jul 31, 2018

Getting started

wft4galaxy is a Python module to automatically run Galaxy workflow tests.

- Run it as a command-line tool or use it programmatically as a Python module.
- Install it locally or in a Docker container – Docker image and helper scripts already provided.

Continue reading to find installation and usage instructions.

CHAPTER 1

Table of Contents (TOC)

1.1 Installation

To install **wft4galaxy** as native Python library, you have to:

1. clone the corresponding github repository:

```
git clone https://github.com/phnmnl/wft4galaxy
```

2. install the package from source code using the usual Python `setup.py`:

```
python setup.py install [--user]
```

Note: Use the option `--user` to install the module only for the current user.

Warning: If are using a linux base system (like *Ubuntu*), probably you need to install the `python-lxml` and `libyaml-dev` packages as a further requirement.

Note: If want to use wft4galaxy with Docker, you can skip the two steps above: see *Dockerized wft4galaxy* for more details.

As a final step, you need to get an **API KEY** from your Galaxy instance, which can be done from the *'User'* menu of the web Galaxy interface. This API KEY, together with the URL of your Galaxy instance (i.e., `http://192.168.64.2:30700`), must be provided to wft4galaxy in order for it to connect to and communicate with that server. This can be done either passing them as parameters to the command line script (see CLI options notebooks/1_run_suite_from_cli.ipynb) and to the the main API endpoints (see *Programmatic Usage*) or setting them as environment variables; i.e.:

```
export GALAXY_URL="<YOUR_GALAXY_SERVER_URL>"
export GALAXY_API_KEY="<YOUR_GALAXY_API_KEY>"
```

1.1.1 Docker-based Installation

wft4galaxy can also run within a Docker container, without installation.

To simplify the usage of the Docker images by command line, we provide a simple script mainly intended to allow users to interact with the dockerized version of the tool as if it was "native", i.e., like a locally installed `wft4galaxy`. This script is called `wft4galaxy-docker`.

Installation

To install `wft4galaxy-docker` so that it is available system-wide, you can use the following command which will download and install the script to `/usr/local/bin`:

```
curl -s https://raw.githubusercontent.com/phnmnl/wft4galaxy/master/utils/docker/
↪install.sh | bash
```

If your `PATH` includes `/usr/local/bin` you will have the `wft4galaxy-docker` script immediately available from your terminal. Alternatively, you can install the `wft4galaxy-docker` script in any other folder of your system by simply appending the string `/dev/stdin <TARGET_FOLDER>` to the line above, replacing `TARGET_FOLDER` with the folder you want to use for installation. For example, if you want to install the script to your current directory, cut and paste the following line to your terminal:

```
curl -s https://raw.githubusercontent.com/phnmnl/wft4galaxy/master/utils/docker/
↪install.sh | bash /dev/stdin .
```

Then, type `./wft4galaxy-docker` to launch from your current path.

1.2 Run a test suite

Given **wft4galaxy** installed as native Python library (see Installation) and a well formed workflow test suite definition file (we use "examples/change_case/workflow-test.yml" as example; see Define a test suite from a Workflow and "Test Definition file reference" for more details) you can run your tests by simply typing:

```
In [2]: wft4galaxy -f ../examples/change_case/workflow-test.yml

Workflow Test: 'change_case' ... ok

----------------------------------------------------------------------
Ran 1 test in 12.294s

OK
```

Notice that if the name of your *test-definition-file* is `workflow-test-suite.yml` and it is in your current working directory, you can ommit the option `-f <...>` and run the test above by simply typing `wft4galaxy` from your current path.

As you can see in the call [1] above, the default behaviour of the **wft4galaxy** tool is to run tests with a minimal output, showing the executed tests and which of them succeeded or failed. To display more information about the test execution you can enable logs, using one of the following options:

```
--enable-logger           Enable INFO level logs
--debug                   Show more details using DEBUG level logs
```

```
In [3]: wft4galaxy -f ../examples/change_case/workflow-test.yml --enable-logger
```

```
2017-03-29 16:54:57,414 [wft4galaxy] [ INFO]  Configuration: <wft4galaxy.core.WorkflowTestSuite objec
Workflow Test: 'change_case' ... 2017-03-29 16:54:58,434 [wft4galaxy] [ INFO]  Create a history '_Wor
2017-03-29 16:54:59,798 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Change Case (imported from API)
2017-03-29 16:55:04,081 [wft4galaxy] [ INFO]  waiting for datasets
2017-03-29 16:55:04,304 [wft4galaxy] [ INFO]  6e7233e069aad1a7: new
2017-03-29 16:55:06,861 [wft4galaxy] [ INFO]  6e7233e069aad1a7: queued
2017-03-29 16:55:07,608 [wft4galaxy] [ INFO]  6e7233e069aad1a7: ok
2017-03-29 16:55:08,113 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Change Case (imported from API)
2017-03-29 16:55:08,114 [wft4galaxy] [ INFO]  Checking test output: ...
2017-03-29 16:55:08,309 [wft4galaxy] [ INFO]  Checking test output: DONE
ok

----------------------------------------------------------------------
Ran 1 test in 11.289s

OK
```

Another useful option to deeply inspect the output of a test execution (especially when failed) is `--disable-cleanup`, which tells wft4galaxy to not clean all intermediate files produced during its execution. All of them (logs and workflow output datasets) are stored in the `./results` output folder, but you can set your preferred target folder with the option `-o <YOUR_FOLDER>` (short form of `--output <YOUR_FOLDER>`).

```
In [7]: wft4galaxy -f ../examples/change_case/workflow-test.yml --enable-logger -o change_case_result
```

```
2017-03-29 17:38:56,728 [wft4galaxy] [ INFO]  Configuration: <wft4galaxy.core.WorkflowTestSuite objec
Workflow Test: 'change_case' ... 2017-03-29 17:38:57,567 [wft4galaxy] [ INFO]  Create a history '_Wor
2017-03-29 17:38:58,910 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Change Case (imported from API)
2017-03-29 17:39:03,114 [wft4galaxy] [ INFO]  waiting for datasets
2017-03-29 17:39:03,501 [wft4galaxy] [ INFO]  c389501033f7c311: new
2017-03-29 17:39:04,420 [wft4galaxy] [ INFO]  c389501033f7c311: queued
2017-03-29 17:39:05,506 [wft4galaxy] [ INFO]  c389501033f7c311: queued
2017-03-29 17:39:06,229 [wft4galaxy] [ INFO]  c389501033f7c311: running
2017-03-29 17:39:06,864 [wft4galaxy] [ INFO]  c389501033f7c311: running
2017-03-29 17:39:07,806 [wft4galaxy] [ INFO]  c389501033f7c311: ok
2017-03-29 17:39:08,309 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Change Case (imported from API)
2017-03-29 17:39:08,309 [wft4galaxy] [ INFO]  Checking test output: ...
2017-03-29 17:39:08,442 [wft4galaxy] [ INFO]  Checking test output: DONE
ok

----------------------------------------------------------------------
Ran 1 test in 11.714s

OK
```

The resulting `change_case_results` folder will contain:

1. test-suite level logs (i.e., a global log file for the test suite):

```
In [18]: ls change_case_results/*.log
```

```
change_case_results/20170329@171841.log       change_case_results/20170329@173856.log
```

2. test-case level logs and output datasets (i.e., a log file specific for the test case and the datasets generated as output during the workflow execution):

```
In [13]: ls change_case_results/change_case
```

```
20170329@173857.log     OutputText
```

1.3 Dockerized wft4galaxy

wft4galaxy can also run within a Docker container, without installation.

To simplify the usage of the Docker images by command line, we provide a simple script mainly intended to allow users to interact with the dockerized version of the tool as if it was "native", i.e., like a locally installed wft4galaxy. This script is called **wft4galaxy-docker**.

wft4galaxy-docker supports different options; type `wft4galaxy-docker --help` to see all them:

```
usage: wft4galaxy-docker [-h] [--registry REGISTRY] [--repository REPO]
                         [--version VERSION] [--image IMAGE]
                         [--os {alpine,ubuntu}] [--skip-update]
                         [--server SERVER] [--api-key API_KEY] [--port PORT]
                         [--volume VOLUME] [--debug]
                         {jupyter,runtest,generate-test,ipython,generate-template,
↪bash}
                         ...

optional arguments:
  -h, --help            show this help message and exit
  --registry REGISTRY   Alternative Docker registry (default is "DockerHub")
  --repository REPO     Alternative Docker repository containing the "wft4galaxy"␣
↪Docker image (default is "crs4")
  --version VERSION     Alternative version of the "wft4galaxy" Docker␣
↪image(default is "latest")
  --image IMAGE         Alternative "wft4galaxy" Docker image name specified as␣
↪NAME:TAG
  --os {alpine,ubuntu}  Base OS of the Docker image (default is "alpine" and it␣
↪is ignored when the "--image" option is specified)
  --skip-update         Skip the update of the "wft4galaxy" Docker image and use␣
↪the local version if it is available
  --server SERVER       Galaxy server URL
  --api-key API_KEY     Galaxy server API KEY
  --port PORT           Docker port to expose
  --volume VOLUME       Docker volume to mount
  --debug               Enable debug mode

Container entrypoint:
  Available entrypoints for the 'wft4galaxy' Docker image.

  {jupyter,runtest,generate-test,ipython,generate-template,bash}
                        Choose one of the following options:
    jupyter             Execute the "Jupyter" server as entrypoint
    runtest             Execute the "wft4galaxy" tool as entrypoint (default)
    generate-test       Execute the "generate-test" wizard command as entrypoint
    ipython             Execute the "Ipython" shell as entrypoint
    generate-template   Execute the "generate-template" wizard command as␣
↪entrypoint
    bash                Execute the "Bash" shell as entrypoint
```

Which Galaxy?

You specify which Galaxy instance `wft4galaxy-docker` should use by setting these two environment variables which are automagically injected on Docker containers:

```
GALAXY_URL                    Galaxy URL
GALAXY_API_KEY                User API key
```

You can override this behaviour from the command line with these switches:

```
--server  SERVER       Galaxy server URL
--api-key API_KEY      Galaxy server API KEY
```

1.3.1 Basic Usage

As a first basic use case, you can use `wft4galaxy-docker` to launch your Galaxy workflow tests just as you would with the `wft4galaxy` script provided by native installation of **wft4galaxy** (see "*Installation*" section). This corresponds to launch `wft4galaxy-docker` with the argument `runtest`, which is its default execution mode.

Thus, you can run all the tests defined in a workflow test suite definition file, like `examples/workflow-testsuite-conf.yml` (see "notebooks/1_define_test_suite.ipynb" and "*Test Definition File Reference*" for more details), by simply typing:

```
wft4galaxy-docker runtest -f examples/workflow-testsuite-conf.yml
```

Being the default command, *runtest* can be omitted:

```
wft4galaxy-docker -f examples/workflow-testsuite-conf.yml
```

That is, the same syntax supported by the `wft4galaxy` script (see "notebooks/0_basic_usage.ipynb").

The other basic two use cases for `wft4galaxy-docker` deal with the *wizard* feature that wft4galaxy provides (see *Wizard Tool*). For example, you can generate a test-suite template folder:

```
wft4galaxy-docker -o MyTestSuite generate-template
```

or generate a test from a Galaxy history:

```
wft4galaxy-docker -o MyTestSuite generate-test MyHistoryName
```

1.3.2 Development Usage

The `wft4galaxy-docker` script also provides several advanced usage commands mainly intended for development purposes. They provide you a development environment with wft4galaxy already installed, configured and ready to use running within a Docker container, where you can interact with wft4galaxy either via its main *wft4galaxy* command (see notebooks/0_basic_usage.ipynb) or programmatically, using its API (see *wft4galaxy — WorkflowTester for Galaxy API*).

Specifically, the dockerized development environments are:

Environment	wft4galaxy-docker CMD
BASH shell	`wft4galaxy-docker bash`
ipython interpreter	`wft4galaxy-docker ipython`
jupyter notebook server	`wft4galaxy-docker jupyter`

Note: The default port of the jupyter notebook server is **9876**: use `--web-port` to change it.

Customized container instances

wft4galaxy-docker allows to customize your running container set-up with a Docker-like syntax:

mount a volume	`-volume myhost-folder:/container-host-folder`
expose a port	`--port 8888:80`

Customized images

The wft4galaxy Docker images officially supported are based on Alpine and Ubuntu Linux. The two versions are equivalent since they have the same set of packages installed. But the `alpine` linux version is used by default due to its smaller size (~250 MB), about the half of the equivalent based on ubuntu (~548.3 MB). You can use the option `--os [alpine|ubuntu]` to choose which use.

You can also build your custom Docker image and tell `wft4galaxy-docker` how to pull it by using the options:

```
--registry REGISTRY   Alternative Docker registry (default is "DockerHub")
--repository REPO     Alternative Docker repository containing the "wft4galaxy"␣
↪Docker image (default is "crs4")
--version VERSION     Alternative version of the "wft4galaxy" Docker image(default is
↪"latest")
--image IMAGE         Alternative "wft4galaxy" Docker image name specified as NAME:TAG
```

Note: When you launch `wft4galaxy-docker`, by default it tries to pull the latest version of Docker image it requires. To avoid this behaviour we can launch it with the option `--skip-update` which forces the use of your local available version of the required Docker image.

1.3.3 Direct Docker Usage

For a direct Docker usage the following syntax holds:

```
docker run -it --rm [DOCKER_OPTIONS] crs4/wft4galaxy[-develop]:[image-version-tag] \
                   <ENTRYPOINT> [ENTRYPOINT_OPTIONS]
```

Note: When using `docker` directly you will need to explicitly mount the volumes that are required to read the configuration file of your suite and to write results.

Direct Docker Usage

```
In [1]: # Galaxy settings
        GALAXY_SERVER=${GALAXY_URL}
        GALAXY_API_KEY=${GALAXY_API_KEY}

        # set the working dir: path must be absolute as Docker requirement
        WORKSPACE="$(pwd)/.." #${WORKSPACE:-$(pwd)}

        # absolute path of your test definition and test data
        LOCAL_INPUT_FOLDER="${WORKSPACE}/examples/change_case"
        LOCAL_OUTPUT_FOLDER="${WORKSPACE}/results"
```

```
        # test definition file relative the $LOCAL_INPUT_FOLDER
        TEST_DEFINITION_FILENAME="workflow-test.yml"

        # test to be executed
        TESTS="change_case"

        # Docker settings
        DOCKER_IMAGE="crs4/wft4galaxy"
        DOCKER_INPUT_FOLDER="/data_input"
        DOCKER_OUTPUT_FOLDER="/data_output"
        DOCKER_CONFIG_FILE="${DOCKER_INPUT_FOLDER}/${TEST_DEFINITION_FILENAME}"

        # run test
        docker run -i --rm \
                      -v "${LOCAL_INPUT_FOLDER}":${DOCKER_INPUT_FOLDER} \
                      -v "${LOCAL_OUTPUT_FOLDER}":${DOCKER_OUTPUT_FOLDER} \
                      ${DOCKER_IMAGE} \
                      --server ${GALAXY_SERVER} --api-key ${GALAXY_API_KEY} \
                      -f ${DOCKER_CONFIG_FILE} \
                      -o ${DOCKER_OUTPUT_FOLDER} ${TESTS}
```

```
2016-11-12 13:25:18,992 INFO: Configuration: {'galaxy_url': None, 'enable_logger': True, 'galaxy_api_
Workflow Test: 'change_case' ... 2016-11-12 13:25:19,721 INFO: Create a history '_WorkflowTestHistory
2016-11-12 13:25:21,666 INFO: Workflow '_WorkflowTest_Change Case (imported from API)' (id: 6d2d4099e
2016-11-12 13:25:29,477 INFO: Workflow '_WorkflowTest_Change Case (imported from API)' (id: 6d2d4099e
2016-11-12 13:25:29,482 INFO: Checking test output: ...
2016-11-12 13:25:29,623 INFO: Checking test output: DONE
ok

----------------------------------------------------------------------
Ran 1 test in 11.003s

OK
```

You can find an example here.

1.4 Define a test-suite from a Workflow

This first example shows how to configure a **wft4galaxy** workflow test for Galaxy.

Main steps:

1. prepare the Galaxy workflow to test: check workflow consistency and export it from Galaxy as a **.ga** file;
2. choose its **input datasets** and generate its **expected output files**;
3. write a workflow test **configuration file**;
4. launch wft4galaxy to execute the test.

As an example, we consider the workflow **change_case** whose files are in the folder `examples/change_case`:

```
In [1]: SUITE_FOLDER="../examples/change_case"
        cd ${SUITE_FOLDER}
        ls
```

```
expected_output                workflow-test.yml
input                    workflow.ga
```

1.4.1 1. Workflow preparation

Consider the following simple workflow **''ChangeCase''** which simply changes (to upper) the case of an input text by using only one Galaxy text transformation tool: the *ChangeCase* tool.

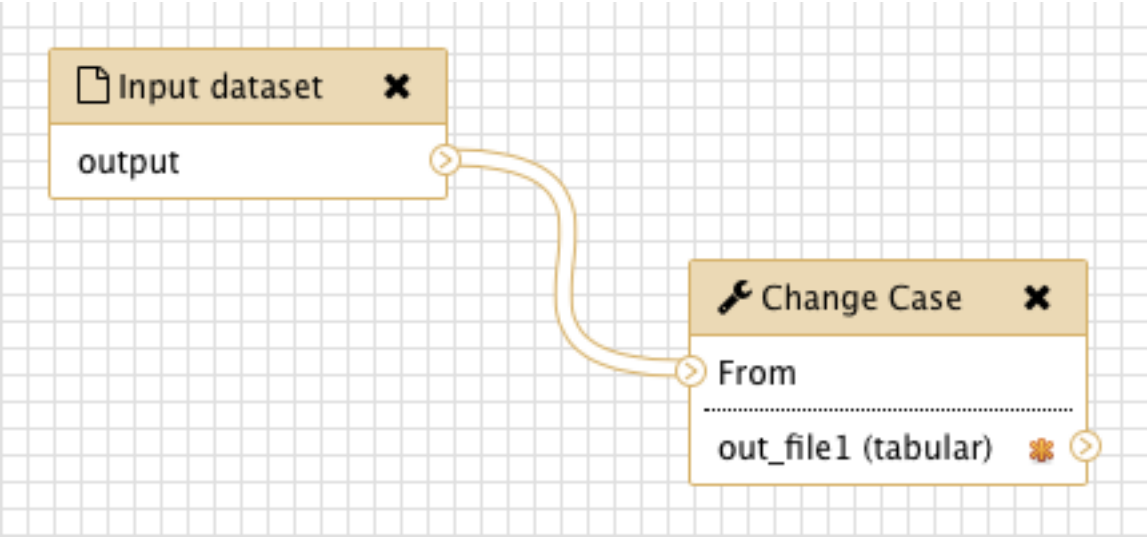

Figure 1. Workflow *"ChangeCase"*

As Fig. 1 shows, the workflow has one input and one output. To test it, both the input and the output must be uniquely identified. Typically, Galaxy identifies them using **Names** and **Labels**, respectively. For our sample workflow, the identifiers are:

- Name **"InputText"** for the input (Fig. 2);
- Label **"OutputText"** for the output (Fig. 3).

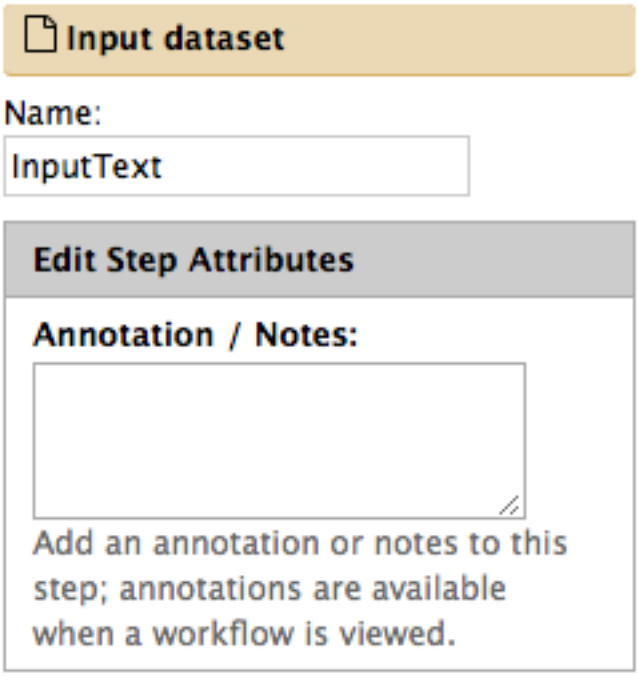

Figure 2. Workflow input: *"Input Dataset"*

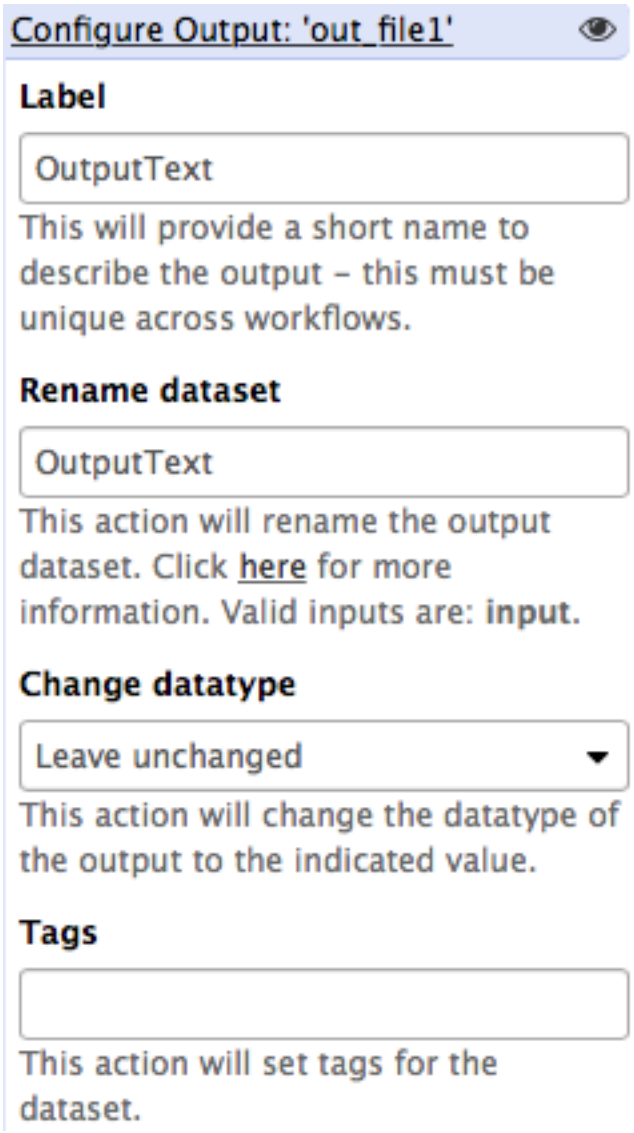

Figure 3. Workflow output: *"output1"*

Satisfied that both input and outputs are uniquely identified, we can download the Galaxy workflow definition, i.e., the **.ga** file. To obtain the .ga file you have to select your workflow from the *Workflows* menu of the Galaxy web interface, click *Share or Download* and finally click the button *Download*.

1.4.2 2. Choose inputs and expected outputs

As input for this workflow, we can use any text file (e.g., `examples/change_case/input.txt`) and, as expected output, a file containing the same text but in upper case (e.g., `examples/change_case/expected_output.txt`).

1.4.3 3. Define the workflow test

The code below defines a simple test for our sample workflow. It's in the file `workflows-test.yml`:

```
workflows:
  # workflow test "change case"
  change_case:
    file: "workflow.ga"
    inputs:
      InputText: "input"
    expected:
      OutputText: "expected_output"
```

See the Section "Test Definition File" of the documentation for more details about the legal test definition syntax.

1.4.4 4. Execute the defined test

The only mandatory argument to run the test is the definition file, which can be specified using the option `-f | --file`:

```
In [2]: wft4galaxy -f "workflow-test.yml" --enable-logger \
                   --server http://192.168.64.8:30700 --api-key 4b86f51252b5f220012b3e259d0877f9

[]
2016-11-11 15:48:55,150 INFO: Configuration: {'galaxy_url': 'http://192.168.64.8:30700', 'enable_logg
Workflow Test: 'change_case' ... 2016-11-11 15:48:55,806 INFO: Create a history '_WorkflowTestHistory
2016-11-11 15:48:57,601 INFO: Workflow '_WorkflowTest_Change Case (imported from API)' (id: 52d6bdfa
2016-11-11 15:49:05,914 INFO: Workflow '_WorkflowTest_Change Case (imported from API)' (id: 52d6bdfa
2016-11-11 15:49:05,916 INFO: Checking test output: ...
2016-11-11 15:49:05,964 INFO: Checking test output: DONE
ok

----------------------------------------------------------------------
Ran 1 test in 11.152s

OK
```

1.5 Define a test-suite from a History

This example shows how to **wft4galaxy wizard tool** for automatically generate a workflow test suite from a Galaxy history (see also Wizard Tool).

Suppose you have the Galaxy history shown in **Figure 1** obtained by running the *Fluxomics Stationary* workflow (you can find it in `examples/fluxomics_stationary`). The first five datasets are inputs, while the remaining are obtained by running the workflow.

Figure 1. History: "_WorkflowTestHistory_21b96e0a-130b-11e7-8958-a45e60c4fc6b"

If you are somehow confident that the outputs above are exactly what you should obtain running the workflow running the workflow on the first five inputs, then you can use them as *expected output* for defining a workflow test case.

That described above is exactly the scenario where the **wft4galaxy-wizard** tool can help you from automatically generate a test case. Knowing that the history name is *_WorkflowTestHistory_21b96e0a-130b-11e7-8958-a45e60c4fc6b* you can obtain such a test case by simply typing:

```
In [1]: wft4galaxy-wizard -o MyTestSuite generate-test _WorkflowTestHistory_21b96e0a-130b-11e7-8958-a
```

```
2017-03-30 04:10:01,139 [wft4galaxy] [ INFO]  Loading Galaxy history info ...
2017-03-30 04:10:05,962 [wft4galaxy] [ INFO]  Selected history: _WorkflowTestHistory_21b96e0a-130b-11
2017-03-30 04:10:05,962 [wft4galaxy] [ INFO]  Loading history 0a248a1f62a0cc04 info
2017-03-30 04:10:06,285 [wft4galaxy] [ INFO]  Processing history info...
2017-03-30 04:10:07,719 [wft4galaxy] [ INFO]  Processing dataset 2fdbd5c5858e78fb ...
2017-03-30 04:10:07,836 [wft4galaxy] [ INFO]  Process dataset 2fdbd5c5858e78fb: done
2017-03-30 04:10:07,836 [wft4galaxy] [ INFO]  Processing dataset c385e49b9fe1853c ...
2017-03-30 04:10:08,025 [wft4galaxy] [ INFO]  Process dataset c385e49b9fe1853c: done
2017-03-30 04:10:08,025 [wft4galaxy] [ INFO]  Processing dataset f7bb1edd6b95db62 ...
2017-03-30 04:10:08,141 [wft4galaxy] [ INFO]  Process dataset f7bb1edd6b95db62: done
```

```
2017-03-30 04:10:08,141 [wft4galaxy] [ INFO]  Processing dataset 52e496b945151ee8 ...
2017-03-30 04:10:08,257 [wft4galaxy] [ INFO]  Process dataset 52e496b945151ee8: done
2017-03-30 04:10:08,257 [wft4galaxy] [ INFO]  Processing dataset b887d74393f85b6d ...
2017-03-30 04:10:08,444 [wft4galaxy] [ INFO]  Process dataset b887d74393f85b6d: done
2017-03-30 04:10:08,444 [wft4galaxy] [ INFO]  Processing dataset f0f309c56aff0025 ...
2017-03-30 04:10:08,561 [wft4galaxy] [ INFO]  Process dataset f0f309c56aff0025: done
2017-03-30 04:10:08,561 [wft4galaxy] [ INFO]  Processing dataset 36ddb788a0f14eb3 ...
2017-03-30 04:10:08,679 [wft4galaxy] [ INFO]  Process dataset 36ddb788a0f14eb3: done
2017-03-30 04:10:08,679 [wft4galaxy] [ INFO]  Processing dataset 72ad249754f05d26 ...
2017-03-30 04:10:08,806 [wft4galaxy] [ INFO]  Process dataset 72ad249754f05d26: done
2017-03-30 04:10:08,806 [wft4galaxy] [ INFO]  Processing dataset eafb646da3b7aac5 ...
2017-03-30 04:10:08,806 [wft4galaxy] [ INFO]  Process dataset eafb646da3b7aac5: done
2017-03-30 04:10:08,807 [wft4galaxy] [ INFO]  Processing dataset 42a2c611109e5ed3 ...
2017-03-30 04:10:08,807 [wft4galaxy] [ INFO]  Process dataset 42a2c611109e5ed3: done
2017-03-30 04:10:08,807 [wft4galaxy] [ INFO]  Processing extra info...
2017-03-30 04:10:08,914 [wft4galaxy] [ INFO]  Processing extra info: done
2017-03-30 04:10:08,914 [wft4galaxy] [ INFO]  History info processing: done
2017-03-30 04:10:08,914 [wft4galaxy] [ INFO]  Extracting Workflow from history...
2017-03-30 04:10:08,940 [wft4galaxy] [ INFO]  Saving workflow to file...
2017-03-30 04:10:08,943 [wft4galaxy] [ INFO]  Saving workflow to file: done
2017-03-30 04:10:08,943 [wft4galaxy] [ INFO]  Extracting Workflow from history: done
2017-03-30 04:10:08,944 [wft4galaxy] [ INFO]  Downloading input datasets...
2017-03-30 04:10:09,729 [wft4galaxy] [ INFO]  Downloading input datasets: done
2017-03-30 04:10:09,729 [wft4galaxy] [ INFO]  Downloading output datasets...
2017-03-30 04:10:09,850 [wft4galaxy] [ INFO]  Downloading output datasets: done
2017-03-30 04:10:09,852 [wft4galaxy] [ INFO]  Saving workflow test definition ...
2017-03-30 04:10:09,888 [wft4galaxy] [ INFO]  Saving workflow test definition: done
```

The wizard tool collects all history info which are useful to generate a test case. In particular, it extracts the workflow executed to generate the history and produces its definition file, i.e., the `.ga` definition file. Then, it downloads all history datasets to generate the test case and creates a new folder called `MyTestSuite` with the following structure:

```
\ MyTestSuite
| -- workflow-test-suite.yml
| -- workflow.ga
+ -- inputs
|      | constraints.csv
|      | exchanged_file.csv
|      | sbml_model.csv
|      | tracing_model.csv
|      | wd.zip
+ -- expected
|      | best_fit_fluxes.csv
|      | best_fit_label.csv
|      | constrained_sbml_model.xml
```

```
In [2]: ls MyTestSuite
```

```
expected                workflow-test-suite.yml
inputs                  workflow.ga
```

As you can see in the `workflow-test-suite.yml` file, datasets uploaded by users are mapped to test case `inputs`, while datasets generated by the workflow execution are mapped to test case `expected-outputs`.

```
In [3]: cat MyTestSuite/workflow-test-suite.yml
```

```
################################################################################
#  Global settings
################################################################################
galaxy_url:     "http://192.168.8.105:30700"            # default is GALAXY_URL
galaxy_api_key: "a4725a45123fd949d24878143801cd82"      # default is GALAXY_API_KEY
```

```
enable_debug:   "False"                                    # enable debug level
output_folder:  "results"                                  # path folder for outputs

################################################################################################
#  Workflow tests
################################################################################################
workflows:

  ##############################################################################################
  # workflow test case: "workflow_test_case_1"
  ##############################################################################################
  workflow_test_case_1:

    # worflow definition file (i.e., `.ga`file)
    file: "workflow.ga"

    # input section
    ############################################################################################
    inputs:

      # extended form
      zipped_data:
        file: "inputs/wd.zip"
        type: "zip"

      # extended form
      tracing_model:
        file: "inputs/tracing_model.csv"
        type: "csv"

      # extended form
      sbml_model:
        file: "inputs/sbml_model.sbml"
        type: "xml"

      # extended form
      inputExchange:
        file: "inputs/exchanged_file.csv"
        type: "csv"

      # extended form
      constraints:
        file: "inputs/constraints.csv"
        type: "csv"

    # expected outputs
    ############################################################################################
    expected:

      # extended form
      constrained_sbml_model:
        file: "expected/constrained_sbml_model.xml"
        comparator: "filecmp.cmp"

      # extended form
      best_fit_fluxes:
        file: "expected/best_fit_fluxes.csv"
        comparator: "filecmp.cmp"
```

```
    # extended form
    best_fit_label:
      file: "expected/best_fit_label.csv"
      comparator: "filecmp.cmp"
```

1.6 Programmatic Usage

In this section we will see how to use **wft4galaxy** programmatically, which can be useful to integrate it with other third party tools.

1.6.1 Run a test suite programmatically

See the folder *example*:

```
In [1]: EXAMPLES_FOLDER = "../examples"
```

. . . where you can find the following files:

```
In [2]: import os, pprint
        [f for f in os.listdir(EXAMPLES_FOLDER) if not f.startswith('.')]
```

```
Out[2]: ['change_case',
         'fluxomics_stationary',
         'multivariate',
         'sacurine',
         'workflow-test-suite-full.yml',
         'workflow-test-suite-min.yml',
         'workflow-test-suite.yml',
         'workflows.json']
```

To run a test suite you need test suite **definition file** (see here for more details) like `workflow-test-suite-min.yml` that you can find in `EXAMPLES_FOLDER`:

```
In [3]: suite_conf_filename = os.path.join(EXAMPLES_FOLDER, "workflow-test-suite-min.yml")
```

The content of the definition file is:

```
In [4]: import yaml, json
        with open(suite_conf_filename, "r") as fp:
            data = yaml.load(fp)
            print(json.dumps(data, indent=4))
```

```
{
    "enable_logger": false,
    "workflows": {
        "change_case": {
            "expected": {
                "OutputText": "change_case/expected_output"
            },
            "inputs": {
                "InputText": "change_case/input"
            },
            "file": "change_case/workflow.ga"
        },
        "multivariate": {
            "expected": {
                "variableMetadata_out": "multivariate/variableMetadata_out",
                "sampleMetadata_out": "multivariate/sampleMetadata_out"
```

```
            },
            "inputs": {
                "DataMatrix": "multivariate/dataMatrix.tsv",
                "SampleMetadata": "multivariate/sampleMetadata.tsv",
                "VariableMetadata": "multivariate/variableMetadata.tsv"
            },
            "params": {
                "3": {
                    "predI": "1",
                    "respC": "gender",
                    "orthoI": "NA",
                    "testL": "FALSE"
                }
            },
            "file": "multivariate/workflow.ga"
        }
    }
}
```

To run a test suite programmatically, you need an instance of `wft4galaxy.core.WorkflowTestSuite` which mantains the configuration of the whole test suite. You can directly instantiate that starting from the test definition file above (cell [4]) by means of the class function `load` (steps [5-6]):

```
In [5]: from wft4galaxy.core import WorkflowTestSuite
```

```
In [6]: suite = WorkflowTestSuite.load(suite_conf_filename)
```

The property `workflows_test` of the suite configuration object contains a dictionary which maps the name of the workflow test to its configuration (step [7]). Notice that the configuration of a workflow test is wrapped by a `wft4galaxy.core.WorkflowTestCase` instance (step[8]).

```
In [7]: for wft_name, wft_config in suite.workflow_tests.items():
            print("{0} ==> {1}\n".format(wft_name, wft_config))
```

```
change_case ==> WorkflowTestConfig: name=change_case, file=change_case/workflow.ga, inputs=[InputText

multivariate ==> WorkflowTestConfig: name=multivariate, file=multivariate/workflow.ga, inputs=[DataMa
```

Now, having the suite definition loaded, we can run the test suite, by calling the `run` method of the suite instance (step [9]) and collect their results:

```
In [8]: test_results = suite.run(enable_logger=True)
```

```
Workflow Test: 'change_case' ... 2017-03-30 11:48:34,390 [wft4galaxy] [ INFO]  Create a history '_Wor
2017-03-30 11:48:35,583 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Change Case (imported from API)
2017-03-30 11:48:37,457 [wft4galaxy] [ INFO]  waiting for datasets
2017-03-30 11:48:40,115 [wft4galaxy] [ INFO]  5148364840389881: new
2017-03-30 11:48:41,313 [wft4galaxy] [ INFO]  5148364840389881: queued
2017-03-30 11:48:42,371 [wft4galaxy] [ INFO]  5148364840389881: queued
2017-03-30 11:48:43,015 [wft4galaxy] [ INFO]  5148364840389881: running
2017-03-30 11:48:43,734 [wft4galaxy] [ INFO]  5148364840389881: running
2017-03-30 11:48:44,614 [wft4galaxy] [ INFO]  5148364840389881: running
2017-03-30 11:48:45,510 [wft4galaxy] [ INFO]  5148364840389881: ok
2017-03-30 11:48:46,011 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Change Case (imported from API)
2017-03-30 11:48:46,013 [wft4galaxy] [ INFO]  Checking test output: ...
2017-03-30 11:48:46,146 [wft4galaxy] [ INFO]  Checking test output: DONE
ok
Workflow Test: 'multivariate' ... 2017-03-30 11:48:47,557 [wft4galaxy] [ INFO]  Create a history '_Wo
2017-03-30 11:48:58,185 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Multivariate (imported from API
2017-03-30 11:49:03,892 [wft4galaxy] [ INFO]  waiting for datasets
2017-03-30 11:49:04,361 [wft4galaxy] [ INFO]  2d0caaef345630d8: queued
```

```
2017-03-30 11:49:04,738 [wft4galaxy] [ INFO]  72e7b234f232ef23: queued
2017-03-30 11:49:04,879 [wft4galaxy] [ INFO]  dbc510811ab4034e: queued
2017-03-30 11:49:05,351 [wft4galaxy] [ INFO]  b7e089dc153354a5: queued
2017-03-30 11:49:05,980 [wft4galaxy] [ INFO]  2d0caaef345630d8: queued
2017-03-30 11:49:06,270 [wft4galaxy] [ INFO]  72e7b234f232ef23: queued
2017-03-30 11:49:06,598 [wft4galaxy] [ INFO]  dbc510811ab4034e: queued
2017-03-30 11:49:06,741 [wft4galaxy] [ INFO]  b7e089dc153354a5: queued
2017-03-30 11:49:07,383 [wft4galaxy] [ INFO]  2d0caaef345630d8: queued
2017-03-30 11:49:07,610 [wft4galaxy] [ INFO]  72e7b234f232ef23: queued
2017-03-30 11:49:07,756 [wft4galaxy] [ INFO]  dbc510811ab4034e: queued
2017-03-30 11:49:07,905 [wft4galaxy] [ INFO]  b7e089dc153354a5: queued
2017-03-30 11:49:08,558 [wft4galaxy] [ INFO]  2d0caaef345630d8: queued
2017-03-30 11:49:08,709 [wft4galaxy] [ INFO]  72e7b234f232ef23: queued
2017-03-30 11:49:08,871 [wft4galaxy] [ INFO]  dbc510811ab4034e: queued
2017-03-30 11:49:08,990 [wft4galaxy] [ INFO]  b7e089dc153354a5: queued
2017-03-30 11:49:09,729 [wft4galaxy] [ INFO]  2d0caaef345630d8: queued
2017-03-30 11:49:10,058 [wft4galaxy] [ INFO]  72e7b234f232ef23: queued
2017-03-30 11:49:10,316 [wft4galaxy] [ INFO]  dbc510811ab4034e: queued
2017-03-30 11:49:10,561 [wft4galaxy] [ INFO]  b7e089dc153354a5: queued
2017-03-30 11:49:11,336 [wft4galaxy] [ INFO]  2d0caaef345630d8: ok
2017-03-30 11:49:11,467 [wft4galaxy] [ INFO]  72e7b234f232ef23: ok
2017-03-30 11:49:11,608 [wft4galaxy] [ INFO]  dbc510811ab4034e: ok
2017-03-30 11:49:11,762 [wft4galaxy] [ INFO]  b7e089dc153354a5: ok
2017-03-30 11:49:12,268 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Multivariate (imported from API
2017-03-30 11:49:12,271 [wft4galaxy] [ INFO]  Checking test output: ...
2017-03-30 11:49:12,470 [wft4galaxy] [ INFO]  Checking test output: DONE
ok

----------------------------------------------------------------------
Ran 2 tests in 39.369s

OK
```

`test_results` is a list of instances of `WorkflowTestResult`, a class which contains several information about an executed workflow, like its ID (dynamically generated when the test starts), the workflow definition and the results of the comparator function for each step (step[9]):

```
In [9]: for r in test_results:
            print("Test %s:\n\t - workflow: [%s] \n\t - results: %r" % (r.test_id, r.workflow.name,
```

```
Test 0a12a39e-152e-11e7-875d-a45e60c4fc6b:
         - workflow: [_WorkflowTest_Change Case (imported from API)]
         - results: {u'OutputText': True}
Test 11e13235-152e-11e7-9e12-a45e60c4fc6b:
         - workflow: [_WorkflowTest_Multivariate (imported from API)]
         - results: {u'variableMetadata_out': True, u'sampleMetadata_out': True}
```

Given a `WorkflowTestResult` instance:

```
In [10]: a_result = test_results[0]
```

the list of available methods for inspecting the results of the workflow test are:

```
In [11]: help(a_result)
```

```
Help on WorkflowTestResult in module wft4galaxy.core object:

class WorkflowTestResult(__builtin__.object)
 |  Class for representing the result of a workflow test.
 |
 |  Methods defined here:
 |
```

```
 |  __init__(self, test_id, workflow, inputs, outputs, output_history, expected_outputs, missing_tool
 |
 |  __repr__(self)
 |
 |  __str__(self)
 |
 |  check_output(self, output)
 |      Assert whether the actual `output` is equal to the expected accordingly
 |      to its associated `comparator` function.
 |
 |      :type output: str or dict
 |      :param output: output name
 |
 |      :rtype: bool
 |      :return: ``True`` if the test is passed; ``False`` otherwise.
 |
 |  check_outputs(self)
 |      Return a map of pairs <OUTPUT_NAME>:<RESULT>, where <RESULT> is ``True``
 |      if the actual `OUTPUT_NAME` is equal to the expected accordingly
 |      to its associated `comparator` function.
 |
 |      :rtype: dict
 |      :return: map of output results
 |
 |  failed(self)
 |      Assert whether the test is failed.
 |
 |      :rtype: bool
 |      :return: ``True`` if the test is failed; ``False`` otherwise.
 |
 |  passed(self)
 |      Assert whether the test is passed.
 |
 |      :rtype: bool
 |      :return: ``True`` if the test is passed; ``False`` otherwise.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

For example, you can extract the list of tested outputs:

```
In [12]: print "Outputs: ", a_result.results.keys()
```

```
Outputs:  [u'OutputText']
```

... or explicitly check if the test is globally passed or failed (all actual outputs are equal to the expected):

```
In [13]: a_result.passed(), a_result.failed()
```

```
Out[13]: (True, False)
```

... or check whether a specific actual output is equal or not to the expected one:

```
In [14]: a_result.check_output("OutputText")
```

```
Out[14]: True
```

1.6.2 Run a single test

See the folder *example*:

```
In [1]: EXAMPLES_FOLDER = "../examples"
```

. . . where you can find the following files:

```
In [2]: import os, pprint
        [f for f in os.listdir(EXAMPLES_FOLDER) if not f.startswith('.')]
```

```
Out[2]: ['change_case',
         'fluxomics_stationary',
         'multivariate',
         'sacurine',
         'workflow-test-suite-full.yml',
         'workflow-test-suite-min.yml',
         'workflow-test-suite.yml',
         'workflows.json']
```

Consider the **definition file** `workflow-test-suite-min.yml` (steps[3-4]), which contains the two workflow tests named `change_case` and `multivariate` respectively:

```
In [3]: suite_conf_filename = os.path.join(EXAMPLES_FOLDER, "workflow-test-suite-min.yml")
```

```
In [4]: import yaml, json
        with open(suite_conf_filename, "r") as fp:
            data = yaml.load(fp)
            print(json.dumps(data, indent=4))
```

```
{
    "enable_logger": false,
    "workflows": {
        "change_case": {
            "expected": {
                "OutputText": "change_case/expected_output"
            },
            "inputs": {
                "InputText": "change_case/input"
            },
            "file": "change_case/workflow.ga"
        },
        "multivariate": {
            "expected": {
                "variableMetadata_out": "multivariate/variableMetadata_out",
                "sampleMetadata_out": "multivariate/sampleMetadata_out"
            },
            "inputs": {
                "DataMatrix": "multivariate/dataMatrix.tsv",
                "SampleMetadata": "multivariate/sampleMetadata.tsv",
                "VariableMetadata": "multivariate/variableMetadata.tsv"
            },
            "params": {
                "3": {
                    "predI": "1",
                    "respC": "gender",
                    "orthoI": "NA",
                    "testL": "FALSE"
                }
            },
            "file": "multivariate/workflow.ga"
        }
```

```
    }
}
```

Similar to that in respect of test suite (example 2), we need to load the test configuration within a Python object. The class specialized for prepresenting a test configuration in `wft4galaxy` is `wft4galaxy.core.WorkflowTestCase` (step [5]):

```
In [5]: from wft4galaxy.core import WorkflowTestCase
```

We can create the class instance using the its static `load` method:

```
In [6]: test_case = WorkflowTestCase.load(suite_conf_filename, workflow_test_name="change_case")
```

Having the `test_case` object, we can simply run the test that it represents by calling its `run` method:

```
In [7]: test_result = test_case.run(enable_logger=True)
```

```
2017-03-30 13:50:18,052 [wft4galaxy] [ INFO]  Create a history '_WorkflowTestHistory_0b688d4c-153f-11
2017-03-30 13:50:19,480 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Change Case (imported from API)
2017-03-30 13:50:24,497 [wft4galaxy] [ INFO]  waiting for datasets
2017-03-30 13:50:24,777 [wft4galaxy] [ INFO]  f711c56f400864d1: new
2017-03-30 13:50:25,697 [wft4galaxy] [ INFO]  f711c56f400864d1: new
2017-03-30 13:50:28,410 [wft4galaxy] [ INFO]  f711c56f400864d1: queued
2017-03-30 13:50:29,249 [wft4galaxy] [ INFO]  f711c56f400864d1: ok
2017-03-30 13:50:29,754 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Change Case (imported from API)
2017-03-30 13:50:29,758 [wft4galaxy] [ INFO]  Checking test output: ...
2017-03-30 13:50:29,920 [wft4galaxy] [ INFO]  Checking test output: DONE
```

`test_result` is an instance of `WorkflowTestResult`, a class which contains several information about an executed workflow, like its ID (dynamically generated when the test starts), the workflow definition and the results of the comparator function for each step:

```
In [8]: print("Test %s:\n\t - workflow: [%s] \n\t - results: %r" % \
              (test_result.test_id, test_result.workflow.name, test_result.results))
```

```
Test 0b688d4c-153f-11e7-87e4-a45e60c4fc6b:
         - workflow: [_WorkflowTest_Change Case (imported from API)]
         - results: {u'OutputText': True}
```

For example, you can extract the list of tested outputs:

```
In [9]: print "Outputs: ", test_result.results.keys()
```

```
Outputs:  [u'OutputText']
```

... or explicitly check if the test is globally passed or failed (all actual outputs are equal to the expected):

```
In [10]: test_result.passed(), test_result.failed()
```

```
Out[10]: (True, False)
```

... or check whether a specific actual output is equal or not to the expected one:

```
In [11]: test_result.check_output("OutputText")
```

```
Out[11]: True
```

1.6.3 Define a single test

See the folder *example*:

```
In [1]: EXAMPLES_FOLDER = "../examples"
```

... where you can find the following files:

```
In [2]: import os, pprint
        [f for f in os.listdir(EXAMPLES_FOLDER) if not f.startswith('.')]
```

```
Out[2]: ['change_case',
         'fluxomics_stationary',
         'multivariate',
         'sacurine',
         'workflow-test-suite-full.yml',
         'workflow-test-suite-min.yml',
         'workflow-test-suite.yml',
         'workflows.json']
```

Consider the **definition file** `workflow-test-suite-min.yml` (steps[3-4]), which contains the two workflow tests named `change_case` and `multivariate` respectively:

```
In [3]: suite_conf_filename = os.path.join(EXAMPLES_FOLDER, "workflow-test-suite-min.yml")
```

```
In [4]: import json, yaml
        with open(suite_conf_filename, "r") as fp:
            data = yaml.load(fp)
            print(json.dumps(data, indent=4))
```

```
{
    "enable_logger": false,
    "workflows": {
        "change_case": {
            "expected": {
                "OutputText": "change_case/expected_output"
            },
            "inputs": {
                "InputText": "change_case/input"
            },
            "file": "change_case/workflow.ga"
        },
        "multivariate": {
            "expected": {
                "variableMetadata_out": "multivariate/variableMetadata_out",
                "sampleMetadata_out": "multivariate/sampleMetadata_out"
            },
            "inputs": {
                "DataMatrix": "multivariate/dataMatrix.tsv",
                "SampleMetadata": "multivariate/sampleMetadata.tsv",
                "VariableMetadata": "multivariate/variableMetadata.tsv"
            },
            "params": {
                "3": {
                    "predI": "1",
                    "respC": "gender",
                    "orthoI": "NA",
                    "testL": "FALSE"
                }
            },
            "file": "multivariate/workflow.ga"
        }
    }
}
```

. . . and focus on the first example `change_case`:

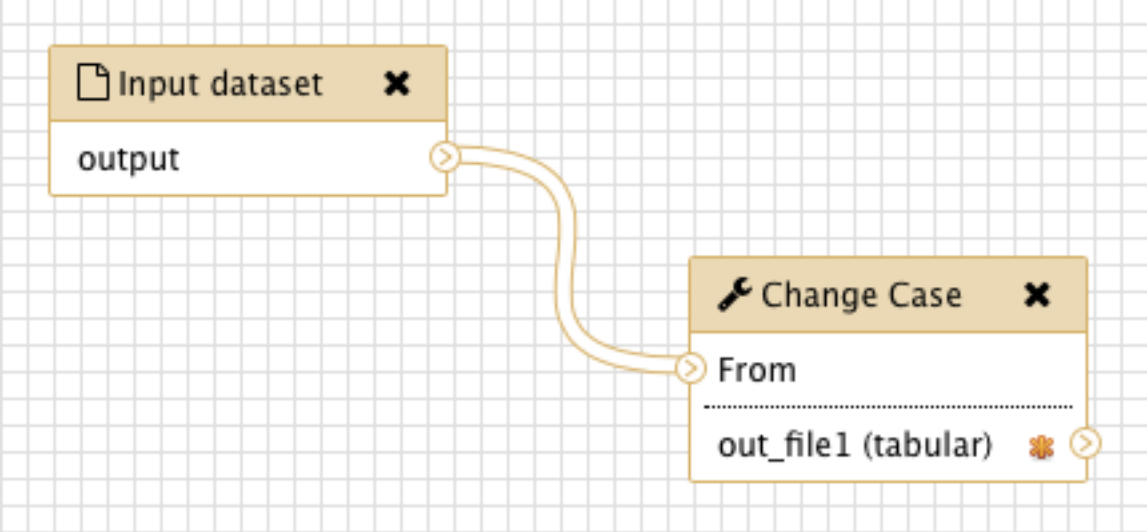

Figure 1. Workflow *"ChangeCase"*

As Fig. 1 shows, the workflow has one input and one output. To test it, both the input and the output must be uniquely identified. Typically, Galaxy identifies them using **Names** and **Labels**, respectively. For our sample workflow, the identifiers are:

- Name **"InputText"** for the input (Fig. 2);
- Label **"OutputText"** for the output (Fig. 3).

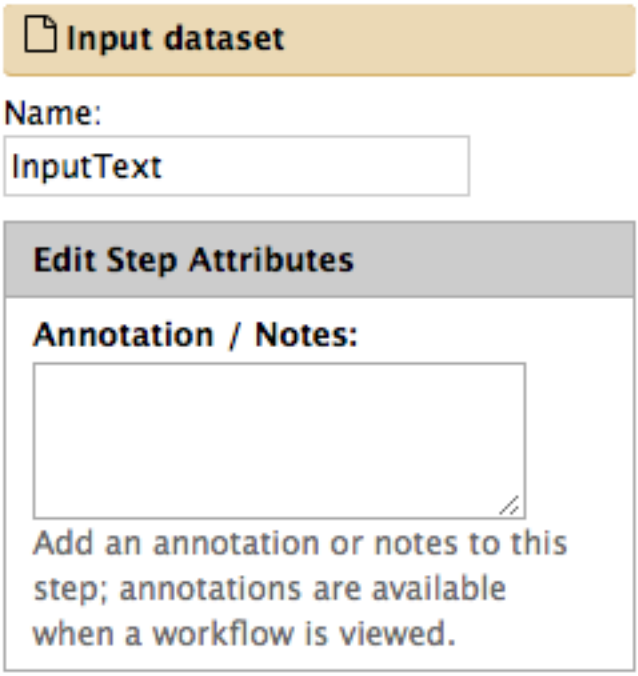

Figure 2. Workflow input: *"Input Dataset"*

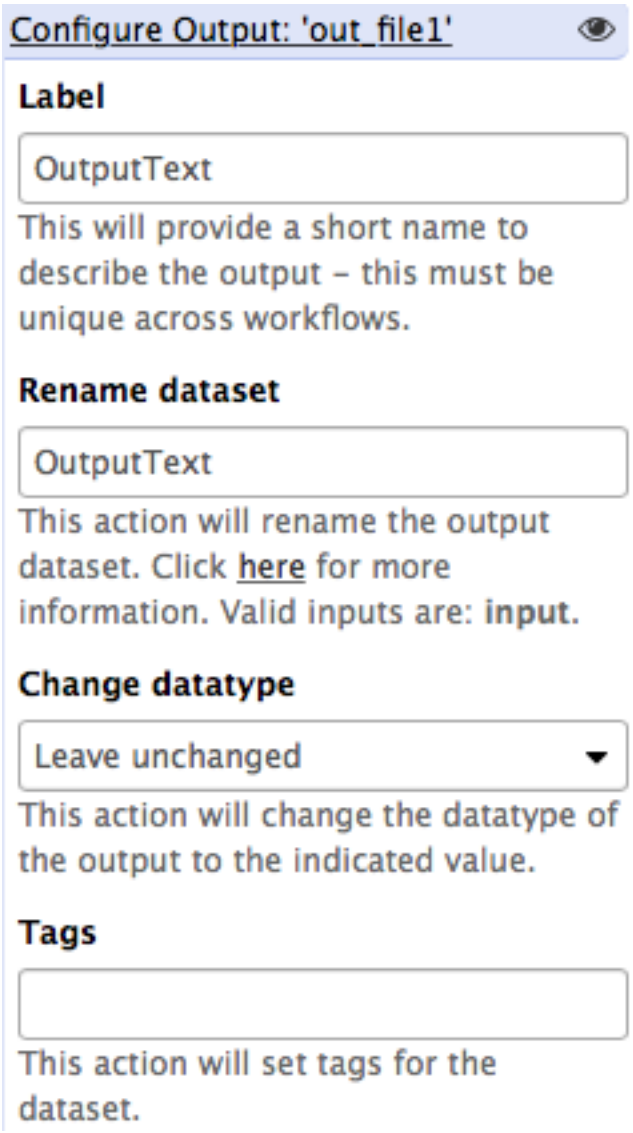

Figure 3. Workflow output: *"output1"*

You can programmatically inpect the Galaxy workflow definition by means of the `Workflow` class, which allows you to see which are the workflow inputs and outputs (steps[5-8]):

```
In [5]: from wft4galaxy.wrapper import Workflow

In [6]: wf = Workflow.load(os.path.join(EXAMPLES_FOLDER, "change_case/workflow.ga"))

In [7]: wf.show_inputs()

-  InputText

In [8]:  wf.show_outputs()

'1': OutputText
```

Steps 9-13 define a `WorkflowTestCase` instance programmatically:

```
In [9]: from wft4galaxy.core import WorkflowTestCase

In [10]: # prepare the workflow definition file name
         base_path = os.path.join(EXAMPLES_FOLDER, "change_case")
         workflow_filename = "workflow.ga"
```

```
In [11]: # prepare test inputs
         inputs = {"InputText": {"file": "input"}}

In [12]: # prepare test outputs
         expected_outputs = {"OutputText": {"file": "expected_output"}}

In [13]: # create test case instance
         test_case = WorkflowTestCase(base_path=base_path,
                                      workflow_filename=workflow_filename,
                                      inputs=inputs, expected_outputs=expected_outputs)
```

You can now run your test case and inspect the results of its execution (steps [14-16]), as described in "Run a single test":

```
In [14]: test_result = test_case.run(enable_logger=True)

2017-03-30 14:53:38,244 [wft4galaxy] [ INFO]  Create a history '_WorkflowTestHistory_e46ecb26-1547-11
2017-03-30 14:53:40,120 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Change Case (imported from API)
2017-03-30 14:53:45,298 [wft4galaxy] [ INFO]  waiting for datasets
2017-03-30 14:53:45,690 [wft4galaxy] [ INFO]  44015a2f24f1f430: queued
2017-03-30 14:53:46,639 [wft4galaxy] [ INFO]  44015a2f24f1f430: queued
2017-03-30 14:53:47,330 [wft4galaxy] [ INFO]  44015a2f24f1f430: running
2017-03-30 14:53:48,075 [wft4galaxy] [ INFO]  44015a2f24f1f430: running
2017-03-30 14:53:49,025 [wft4galaxy] [ INFO]  44015a2f24f1f430: running
2017-03-30 14:53:49,660 [wft4galaxy] [ INFO]  44015a2f24f1f430: ok
2017-03-30 14:53:50,165 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Change Case (imported from API)
2017-03-30 14:53:50,168 [wft4galaxy] [ INFO]  Checking test output: ...
2017-03-30 14:53:50,341 [wft4galaxy] [ INFO]  Checking test output: DONE

In [15]: print("Test %s:\n\t - workflow: [%s] \n\t - results: %r" % \
                (test_result.test_id, test_result.workflow.name, test_result.results))

Test e46ecb26-1547-11e7-8812-a45e60c4fc6b:
         - workflow: [_WorkflowTest_Change Case (imported from API)]
         - results: {u'OutputText': True}

In [16]: test_result.check_output("OutputText")

Out[16]: True
```

1.6.4 Define a test suite

See the folder *example*:

```
In [1]: EXAMPLES_FOLDER = "../examples"
```

... where you can find the following files:

```
In [2]: import os, pprint
        [f for f in os.listdir(EXAMPLES_FOLDER) if not f.startswith('.')]

Out[2]: ['change_case',
         'fluxomics_stationary',
         'multivariate',
         'sacurine',
         'workflow-test-suite-full.yml',
         'workflow-test-suite-min.yml',
         'workflow-test-suite.yml',
         'workflows.json']
```

Consider the **definition file** `workflow-test-suite-min.yml` (steps[3-4]), which contains the two workflow tests named `change_case` and `multivariate` respectively:

```
In [3]: suite_conf_filename = os.path.join(EXAMPLES_FOLDER, "workflow-test-suite-min.yml")
```

```
In [4]: import yaml, json
        with open(suite_conf_filename, "r") as fp:
            data = yaml.load(fp)
            print(json.dumps(data, indent=4))
```

```
{
    "enable_logger": false,
    "workflows": {
        "change_case": {
            "expected": {
                "OutputText": "change_case/expected_output"
            },
            "inputs": {
                "InputText": "change_case/input"
            },
            "file": "change_case/workflow.ga"
        },
        "multivariate": {
            "expected": {
                "variableMetadata_out": "multivariate/variableMetadata_out",
                "sampleMetadata_out": "multivariate/sampleMetadata_out"
            },
            "inputs": {
                "DataMatrix": "multivariate/dataMatrix.tsv",
                "SampleMetadata": "multivariate/sampleMetadata.tsv",
                "VariableMetadata": "multivariate/variableMetadata.tsv"
            },
            "params": {
                "3": {
                    "predI": "1",
                    "respC": "gender",
                    "orthoI": "NA",
                    "testL": "FALSE"
                }
            },
            "file": "multivariate/workflow.ga"
        }
    }
}
```

Suppose that you have two `WorkflowTestCase` instances (we load them in steps[5-6] from a definition file, but you can define them programmatically as described in "Define a single test case"):

```
In [5]: from wft4galaxy.core import WorkflowTestCase
```

```
In [6]: wft1 = WorkflowTestCase.load(suite_conf_filename, "change_case")
        wft2 = WorkflowTestCase.load(suite_conf_filename, "multivariate")
```

To programmatically create test suite you need to create an instance of the `WorkflowTestSuite` class:

```
In [7]: from wft4galaxy.core import WorkflowTestSuite
        suite = WorkflowTestSuite()
```

and register the `WorkflowTestSuite` instances to the suite instance:

```
In [8]: suite.add_workflow_test(wft1)
        suite.add_workflow_test(wft2)
```

You can now run your test suite and inspect the results of its execution (steps [9-10]), as described in "Run a test suite":

```
In [9]: test_results = suite.run(enable_logger=True)
```

```
Workflow Test: 'change_case' ... 2017-03-30 15:08:02,477 [wft4galaxy] [ INFO]  Create a history '_Wor
2017-03-30 15:08:03,756 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Change Case (imported from API)
2017-03-30 15:08:06,367 [wft4galaxy] [ INFO]  waiting for datasets
2017-03-30 15:08:06,559 [wft4galaxy] [ INFO]  53fb5c5113d9beb7: new
2017-03-30 15:08:07,502 [wft4galaxy] [ INFO]  53fb5c5113d9beb7: new
2017-03-30 15:08:08,239 [wft4galaxy] [ INFO]  53fb5c5113d9beb7: new
2017-03-30 15:08:08,900 [wft4galaxy] [ INFO]  53fb5c5113d9beb7: new
2017-03-30 15:08:09,830 [wft4galaxy] [ INFO]  53fb5c5113d9beb7: new
2017-03-30 15:08:10,476 [wft4galaxy] [ INFO]  53fb5c5113d9beb7: new
2017-03-30 15:08:11,386 [wft4galaxy] [ INFO]  53fb5c5113d9beb7: queued
2017-03-30 15:08:12,103 [wft4galaxy] [ INFO]  53fb5c5113d9beb7: running
2017-03-30 15:08:12,807 [wft4galaxy] [ INFO]  53fb5c5113d9beb7: running
2017-03-30 15:08:13,638 [wft4galaxy] [ INFO]  53fb5c5113d9beb7: running
2017-03-30 15:08:14,357 [wft4galaxy] [ INFO]  53fb5c5113d9beb7: running
2017-03-30 15:08:15,365 [wft4galaxy] [ INFO]  53fb5c5113d9beb7: ok
2017-03-30 15:08:15,872 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Change Case (imported from API)
2017-03-30 15:08:15,875 [wft4galaxy] [ INFO]  Checking test output: ...
2017-03-30 15:08:16,054 [wft4galaxy] [ INFO]  Checking test output: DONE
ok
Workflow Test: 'multivariate' ... 2017-03-30 15:08:17,738 [wft4galaxy] [ INFO]  Create a history '_Wo
2017-03-30 15:08:30,657 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Multivariate (imported from API
2017-03-30 15:08:39,075 [wft4galaxy] [ INFO]  waiting for datasets
2017-03-30 15:08:39,483 [wft4galaxy] [ INFO]  b425b2361a02d64e: queued
2017-03-30 15:08:39,970 [wft4galaxy] [ INFO]  eff20c5ba6bf6f87: queued
2017-03-30 15:08:40,455 [wft4galaxy] [ INFO]  77efbec71a7e1a47: queued
2017-03-30 15:08:40,781 [wft4galaxy] [ INFO]  ea3d97b57366b47f: queued
2017-03-30 15:08:41,453 [wft4galaxy] [ INFO]  b425b2361a02d64e: queued
2017-03-30 15:08:41,952 [wft4galaxy] [ INFO]  eff20c5ba6bf6f87: queued
2017-03-30 15:08:42,266 [wft4galaxy] [ INFO]  77efbec71a7e1a47: queued
2017-03-30 15:08:42,443 [wft4galaxy] [ INFO]  ea3d97b57366b47f: queued
2017-03-30 15:08:43,290 [wft4galaxy] [ INFO]  b425b2361a02d64e: queued
2017-03-30 15:08:43,526 [wft4galaxy] [ INFO]  eff20c5ba6bf6f87: queued
2017-03-30 15:08:43,788 [wft4galaxy] [ INFO]  77efbec71a7e1a47: queued
2017-03-30 15:08:44,030 [wft4galaxy] [ INFO]  ea3d97b57366b47f: queued
2017-03-30 15:08:44,678 [wft4galaxy] [ INFO]  b425b2361a02d64e: queued
2017-03-30 15:08:45,228 [wft4galaxy] [ INFO]  eff20c5ba6bf6f87: queued
2017-03-30 15:08:45,440 [wft4galaxy] [ INFO]  77efbec71a7e1a47: queued
2017-03-30 15:08:45,629 [wft4galaxy] [ INFO]  ea3d97b57366b47f: queued
2017-03-30 15:08:46,418 [wft4galaxy] [ INFO]  b425b2361a02d64e: queued
2017-03-30 15:08:46,848 [wft4galaxy] [ INFO]  eff20c5ba6bf6f87: queued
2017-03-30 15:08:47,146 [wft4galaxy] [ INFO]  77efbec71a7e1a47: ok
2017-03-30 15:08:47,662 [wft4galaxy] [ INFO]  ea3d97b57366b47f: ok
2017-03-30 15:08:48,312 [wft4galaxy] [ INFO]  b425b2361a02d64e: ok
2017-03-30 15:08:48,485 [wft4galaxy] [ INFO]  eff20c5ba6bf6f87: ok
2017-03-30 15:08:48,986 [wft4galaxy] [ INFO]  Workflow '_WorkflowTest_Multivariate (imported from API
2017-03-30 15:08:48,989 [wft4galaxy] [ INFO]  Checking test output: ...
2017-03-30 15:08:49,187 [wft4galaxy] [ INFO]  Checking test output: DONE
ok

----------------------------------------------------------------------
Ran 2 tests in 48.039s

OK
```

```
In [10]: for r in test_results:
             print("Test %s:\n\t - workflow: [%s] \n\t - results: %r" % (r.test_id, r.workflow.name,
```

```
Test e79f068a-1549-11e7-823b-a45e60c4fc6b:
         - workflow: [_WorkflowTest_Change Case (imported from API)]
         - results: {u'OutputText': True}
```

```
Test f08acdb0-1549-11e7-bf2e-a45e60c4fc6b:
        - workflow: [_WorkflowTest_Multivariate (imported from API)]
        - results: {u'variableMetadata_out': True, u'sampleMetadata_out': True}
```

1.7 Wizard Tool

wft4galaxy supports users on the creation of their workflow test cases by means of a *"wizard"* functionality which can be called by typing `wft4galaxy-wizard` after installation (see *Installation*) or using Docker (see *Dockerized wft4galaxy*).

It provides users with two main features:

1. **a test suite template generator,** which simply creates a template of workflow test suite definition file (see *Test Definition File Reference*) within a directory reasonable structured for containing a test suite (see *TestSuite Directory Structure* below);
2. **a test suite generator,** which automatically generates a test suite from an existing Galaxy history.

Type `wft4galaxy-wizard --help` to see its main options and arguments:

Listing 1: wft4galaxy-wizard help

```
usage: wft4galaxy-wizard [-h] [--debug] [--server GALAXY_URL]
                         [--api-key GALAXY_API_KEY] [-o OUTPUT_FOLDER]
                         [-f FILE]
                         {generate-test,generate-template} ...

optional arguments:
  -h, --help            show this help message and exit
  --debug               Enable debug mode
  --server GALAXY_URL   Galaxy server URL (default $GALAXY_URL)
  --api-key GALAXY_API_KEY
                        Galaxy server API KEY (default $GALAXY_API_KEY)
  -o OUTPUT_FOLDER, --output OUTPUT_FOLDER
                        absolute path of the output folder (default is "test-
                        config")
  -f FILE, --file FILE  YAML configuration file of workflow tests (default is
                        "workflow-test-suite.yml")

command:

  {generate-test,generate-template}
                        Wizard tool command: [generate-test | generate-
                        template]
    generate-test       Generate a test definition file from a history
    generate-template   Generate a test definition template
```

As with other wft4galaxy features, the options `--server` and `--api-key` override the two environment variable which define the Galaxy instance to use, respectively *GALAXY_URL* and *GALAXY_API_KEY*.

1.7.1 Generate Template

A test suite template directory can be generate by typing:

```
wft4galaxy-wizard generate-template [-f <TEST_SUITE_FILE>] [-o <OUTPUT_FOLDER>]
```

The former option *-f <TEST_SUITE_FILE>* allows you to customize the filename containing the test suite definition (the default filename is *workflow-test-suite.yml*), while the later overrides the default name of the output folder, which is *test-config*. A part from the actual files (workflow definition *.ga* file and datasets), its structure looks like shown in *TestSuite Directory Structure*.

1.7.2 Generate TestSuite from a history

The main purpose is to allow users to automatically generate a workflow test case from a history. Given a history name, the wizard tool collects all history info which are useful to generate a test case.

In particular, it extracts the workflow executed to generate the history and produces its definition file, i.e., the *.ga* definition file. Then, it downloads all history datasets to generate the test case:

- datasets uploaded by users are mapped to test case *inputs* ;
- datasets generated by the workflow execution are mapped to test case *expected-outputs*.

Finally, a workflow test definition file is generated.

The typical directory structure of the resulting test case is:

Listing 2: TestSuite Directory Structure

```
\ <TEST_CASE_OUTPUT_FOLDER>
| -- workflow-test-suite.yml
| -- workflow.ga
+ -- inputs
|      | -- <input-dataset>.<dataset-type>
|     ...
+ -- expected
|      | -- <output-dataset>.<dataset-type>
|     ...
```

To generate a test case from your Galaxy history, type:

```
wft4galaxy-wizard -o <TEST_CASE_OUTPUT_FOLDER> generate-test <HISTORY_NAME>
```

Note: The mandatory parameter *HISTORY_NAME* selects the history to be used for generating the test case. If more than one history matches that name, the wizard asks for choosing one of them, displaying their creation time.

1.8 Test Definition File Reference

wft4galaxy supports defining tests in both YAML and JSON formats.

As an example, consider the following YAML test definition file:

```
## Galaxy server settings
#############################################################################
↪#####
# galaxy_url: "http://192.168.64.8:30700" # default is GALAXY_URL
# galaxy_api_key: "4b86f51252b5f220012b3e259d0877f9" # default is GALAXY_API_KEY
```

(continues on next page)

(continued from previous page)

```
################################################################################################
↪#####
enable_logger: False
output_folder: "results"

# workflow tests
workflows:
  # workflow test "change case"
  change_case:
    file: "change_case/workflow.ga"
    inputs:
      "Input Dataset": "change_case/input"
        #file: "change_case/input"
    expected:
      output1:
        file: "change_case/expected_output_1"
        comparator: "filecmp.cmp"
      output2: "change_case/expected_output_2"

  # workflow test "sacurine"
  sacurine:
    file: "sacurine/workflow.ga"
    params:
      3:
        "orthoI": "NA"
        "predI": "1"
        "respC": "gender"
        "testL": "FALSE"
    inputs:
      "DataMatrix": "sacurine/input/dataMatrix.tsv"
      "SampleMetadata": "sacurine/input/sampleMetadata.tsv"
      "VariableMetadata": "sacurine/input/variableMetadata.tsv"
    expected:
      Univariate_variableMetadata: "sacurine/expected/Univariate_variableMetadata.tsv"
      Multivariate_sampleMetadata: "sacurine/expected/Multivariate_sampleMetadata.tsv"
      Multivariate_variableMetadata: "sacurine/expected/Multivariate_variableMetadata.
↪tsv"
      Biosigner_variableMetadata: "sacurine/expected/Biosigner_variableMetadata.tsv"
```

The definition has two main parts:

1. **global configuration**, where global settings are defined;
2. **workflow configuration**, which contains workflow-specific settings.

1.8.1 Global settings

- `galaxy_url` and `galaxy_api`: Galaxy instance and credentials to run the workflows.
- `output_folder`, if test outputs are to be saved.
- `logging_level`: one of `INFO` and `DEBUG` (default is `INFO`).
- `base_path`: path with respect to which the relative file paths are specified – for datasets, workflow files, etc. (see note).

1.8.2 Workflow settings

- `base_path`: overrides global `base_path`, if specified (see note).
- `file`: workflow file definition (.ga file).
- `inputs`: input files for the workflow. More details below.
- `expected`: output files expected from the workflow in the case of correct execution. More details below.

Base path

The `base_path` is used to locate every relative path specified in a workflow test configuration (i.e., .ga files, input datasets, expected_output datasets).

It can be set at two levels:

1. **global base_path**: specifies the base path for all workflow tests;
2. **test base_path**: specifies the base path for all files defined within the workflow test

---

Note: If you provide a path starting with `/` it is considered absolute and the `base_path` setting will not affect it.

The only exception to this logic is for the output directory, which is created relative to the `base_path` if the location is specified in the test definition (see the `output_folder` key) or, if the `-o` command line option is used, in the path specified by the user.

Specifying workflow inputs

Inputs are specified as a dictionary "Galaxy dataset name" -> file. Here's an example:

```
inputs:
  "DataMatrix": "sacurine/input/dataMatrix.tsv"
```

In the example, the Galaxy workflow has an input dataset with the label "DataMatrix." We're specifying that the file `sacurine/input/dataMatrix.tsv` should be used for that input.

The relative path is interpreted relative to the *Base path*.

By default Galaxy automatically detects the type of the input dataset by its file extension. If you need to specify the type of the input dataset you can use the following extended syntax:

```
inputs:
  DataMatrix:
      file: "sacurine/input/dataMatrix.tsv"
      type: "tsv"
```

Specifying workflow outputs

Outputs are also specified as a dictionary that has the Galaxy output name as the key. The associated value can be the path to a file, so that `wft4galaxy` will require that the output dataset generated from the workflow exactly matches the contents of the file. Here's an example:

```
expected:
  Univariate_variableMetadata: "sacurine/expected/Univariate_variableMetadata.tsv"
```

On the other hand, if required, the comparison operation can be customized files. In this case, the `expected` looks more like this:

```
expected:
  Univariate_variableMetadata:
    file: "sacurine/expected/Univariate_variableMetadata.bin"
    comparator: "filecmp.cmp"
```

The name of the output dataset is still the key the key. In this case, however, the value is another dictionary that requires the `file` key, to specify the data file, and the `comparator` key, to specify the comparator function.

Comparator function

The comparator function is expected to be a function that accepts two paths as arguments and is in a Python module that can be imported by `wft4galaxy` (so mind your `PYTHONPATH`!!). So, for example, `filecmp.cmp` roughly translates to:

```
import filecmp
return filecmp.cmp(expected_file_path, generated_file_path)
```

1.9 Integration with CI tools

`wft4galaxy` can be easily integrated with *Continous Integration* (CI) tools like Jenkins.

A typical approach is to create a repository (e.g., Git) to host your workflow and its tests, which can be defined as `wft4galaxy` tests. More specifically, ,the repository needs to include:

1. the *test definition file*, containing the definition all workflow tests (see config_file for more information);
2. every resource referenced in the definition file above, such as input and expected output datasets.

Finally, for proper continuous integration you should configure your CI tool:

1. to be notified every time a changes are committed to your workflow repository;
2. to start wft4galaxy tests automatically.

How do this in practice depends on the specific CI tool you choose. However, in principle you'll need to configure your tool to launch `wft4galaxy` to run the tests defined for your workflows.

Note: Remember that to run wft4galaxy from your CI tool you have three alternatives:

1. **wft4galaxy** script, if wft4galaxy is installed and accessible from your CI tool's execution environment or if you install wft4galaxy as a step in your CI testing script;
2. **wft4galaxy-docker** script, which can be downloaded on the fly and used to run wft4galaxy tests within a Docker container (in this case, your CI tool must support Docker);
3. **direct docker usage** (see notebooks/6_direct_docker_usage.ipynb for an example).

1.9.1 Jenkins Integration

To configure a Jenkins project to use *wft4galaxy* to test workflows hosted on Github, follow this procedure.

1. Create a new *free style software project*;

2. Set your Git repository in the project box;
3. Set Git as your *Source Code Management* and then the URL of your Git repository;
4. Check the box *Build when a change is pushed to Github*;
5. Add a new *Execute Shell* build step;
6. From the *Execute Shell* box call the `wft4galaxy` tool to run your workflow tests (see note above);
7. Save the configuration of your Jenkins project;
8. Set the proper *Webhook* in your GitHub repository to notify the Jenkins instance when a change happens (i.e., in *Webhooks & Services* > *Jenkins plugin* put the URL of your Jenkins instance).

1.10 `wft4galaxy` — WorkflowTester for Galaxy API

1.10.1 Core Classes

WorkflowTestCase

class wft4galaxy.core.**WorkflowTestCase**(*name=None*, *base_path='.'*, *workflow_filename='workflow.ga'*, *inputs=None*, *params=None*, *expected_outputs=None*, *output_folder=None*, *disable_cleanup=False*, *disable_assertions=False*, *enable_logger=False*, *enable_debug=False*)

A representation of the configuration of a workflow test.

Parameters

- **base_path** (*str*) – base path for workflow and dataset files (the current working path is assumed as default)
- **workflow_filename** (*str*) – the path (relative to `base_path`) of the file containing the workflow definition (i.e., the `.ga` file which can be downloaded from a Galaxy server)
- **name** (*str*) – a name for the workflow test
- **inputs** (*dict*) – a dictionary which defines the mapping between a workflow input and a test dataset.

  Example {"DataMatrix": "dataMatrix.tsv"} where *DataMatrix* is the name of a workflow input and *dataMatrix.tsv* is the file containing the dataset to be used as input for the workflow test.

- **params** (*dict*) – a dictionary which defines the mapping between steps and the set of parameters which has to be used to configure the corresponding tools of each step.

  Example

```
params = {
            3: {
                 "orthoI": "NA"
                 "predI": "1"
                 "respC": "gender"
                 "testL": "FALSE"
                }
         }
```

- **expected_outputs** (*dict*) – a dictionary to configure the expected output, i.e., the output which has to be compared to the actual one produced by a workflow execution. Each output of a workflow step is eligible to be compared with an expected output. It is also possible to specify the python function which has to be used to perform the actual comparison. Such a function takes two parameters, i.e., `actual_output_filename` and `expected_output_filename`, and returns `True` whether the comparison between the two files succeeds and `False` otherwise.

  Example Skeleton of a user-defined comparator:

  ```
  def compare_outputs(actual_output_filename, expected_
  ↪output_filename):
      ....
      return True | False
  ```

  Example The example below shows an `expected_outputs` dictionary that configures the expected output datasets for the two actual workflow outputs `output1` and `output2`. A user defined 'comparator' is also given to compare the expected to the actual `output2`.

  ```
  {
      'output1': 'change_case/expected_output_1',
      'output2': {
          'comparator': 'filecmp.cmp',
          'file': 'change_case_2/expected_output_2'
      }
  }
  ```

- **output_folder** (*str*) – path (relative to `base_path`) of the folder where workflow outputs are written. By default, it is the folder `results/<name>` within the `base_path` (where `name` is the name of the workflow test).
- **disable_cleanup** (*bool*) – `True` to avoid the clean up of the workflow and history created on the Galaxy server; `False` (default) otherwise.
- **disable_assertions** (*bool*) – `True` to disable assertions during the execution of the workflow test; `False` (default) otherwise.

base_path
The base path of the workflow file definition and the input and output datasets.

set_base_path (*base_path*)
Set the base path of the workflow file definition and the input and output datasets.

Parameters **base_path** (*str*) – a path within the local file system

filename
The filename (relative to `base_paht`) of the workflow definition.

set_filename (*filename*)
Set the filename (relative to `base_path`) containing the workflow definition.

Parameters **filename** (*str*) – the path (relative to the `base_path`) to the `.ga` file

inputs
Return the dictionary which defines the mapping between workflow inputs and test datasets.

set_inputs (*inputs*)
Update the mapping between workflow inputs and test datasets.

Parameters **inputs** – dict

Returns a dictionary of mappings (see *WorkflowTestCase*)

add_input(*name*, *path*, *type_=None*)
Add a new input mapping.

Parameters

- **name** (*str*) – the Galaxy label of an input
- **path** (*str*) – the path (relative to the base_path) of the file containing an input dataset
- **type** (*str*) – the type of the input dataset

remove_input(*name*)
Remove an input mapping.

Parameters **name** (*str*) – the Galaxy label of an input

get_input(*name*)
Return the input mapping for the input labeled as name.

Parameters **name** (*str*) – the Galaxy label of the input

Return type dict

Returns input configuration as dict (e.g., {'name': 'Input Dataset', 'file': "input.txt"})

params
Return the dictionary containing the configured parameters (see *WorkflowTestCase*)

Return type dict

Returns a dictionary of configured parameters

set_params(*params*)
Add a new set of parameters.

Parameters **params** (*dict*) – dictionary of parameters indexed by step id (see *WorkflowTestCase*)

add_param(*step_id*, *name*, *value*)
Add a new parameter to the step identified by step_id.

Parameters

- **step_id** (*int*) – step index
- **name** (*str*) – parameter name
- **value** (*str*) – parameter value

remove_param(*step_id*, *name*)
Remove the parameter labeled name from the step identified by step_id.

Parameters

- **step_id** (*int*) – step index
- **name** (*str*) – name of the parameter to be removed

get_params(*step_id*)
Return the dictionary of parameters related to the step identified by 'step_id'.

Parameters **step_id** (*int*) – the step index

Return type dict

Returns the dictionary of parameters related to the step identified by 'step_id'

get_param(*step_id*, *name*)
Return the value of a specific step parameter.

Parameters

- **step_id** (*int*) – the index of the step which the parameter is related to
- **name** (*str*) – the name of the parameter to be returned

Returns the value of the requested parameter

expected_outputs
A dictionary to configure the expected output, i.e., the output which has to be compared to the actual one produced by a workflow execution (see *WorkflowTestCase*).

set_expected_outputs(*expected_outputs*)
Add a new set of expected outputs (see *WorkflowTestCase*).

Parameters **expected_outputs** (*dict*) – a dictionary structured as specified in *WorkflowTestCase*

add_expected_output(*name*, *filename*, *comparator='filecmp.cmp'*)
Add a new expected output to the workflow test configuration.

Parameters

- **name** (*str*) – the Galaxy name of the output which the expected dataset has to be mapped.
- **filename** (*str*) – the path (relative to the base_path) of the file containing the expected output dataset
- **comparator** (*str*) – a fully qualified name of a *comparator'function (see :class:'WorkflowTestCase*)

remove_expected_output(*name*)
Remove an expected output from the workflow test configuration.

Parameters **name** (*str*) – the Galaxy name of the output which the expected output has to be mapped

get_expected_output(*name*)
Return the configuration of an expected output.

Parameters **name** (*str*) – the Galaxy name of the output which the expected output has to be mapped.

Return type dict

Returns a dictionary containing the configuration of the expected output as specified in *WorkflowTestCase*

to_dict()
Return a dictionary representation of the current class instance.

Return type dict

Returns

save(*filename=None*, *file_format='YAML'*)
Serialize this workflow test configuration to a file (YAML or JSON).

Parameters

- **filename** (*str*) – absolute path of the file
- **file_format** (*str*) – YAML or JSON

static load(*filename='workflow-test-suite.yml'*, *workflow_test_name='workflow_test_case'*, *output_folder=None*)

Load the configuration of a workflow test suite or a single workflow test from a YAML or JSON configuration file.

Parameters

- **filename** (*str*) – the path of the file containing the suite definition
- **workflow_test_name** (*str*) – the optional name of a workflow test (default is "workflow-test-case")
- **output_folder** (*str*) – the path of the output folder

Return type *WorkflowTestCase*

Returns the *WorkflowTestCase* instance which matches the name provided by the argument *workflow_test_name_*

static dump(*filename*, *worflow_tests_config*, *file_format='YAML'*)

Write the configuration of a workflow test suite to a YAML or JSON file.

Parameters

- **filename** (*str*) – the absolute path of the YAML or JSON configuration file
- **worflow_tests_config** (*dict or list*) – a dictionary which maps a workflow test name to the corresponding configuration (*WorkflowTestCase*) or a list of *WorkflowTestCase* instances
- **file_format** (*str*) – YAML or JSON

WorkflowTestSuite

class wft4galaxy.core.**WorkflowTestSuite**(*galaxy_url=None*, *galaxy_api_key=None*, *output_folder='results'*, *enable_logger=True*, *enable_debug=False*, *disable_cleanup=False*, *disable_assertions=False*, *max_retries=None*, *retry_delay=None*, *polling_interval=None*)

Represent a test suite.

workflow_tests

Return the configuration of workflow tests associated to this test suite.

Return type dict

Returns a dictionary which maps a *workflow test name* to the *WorkflowTestCase* instance representing its configuration

add_workflow_test(*workflow_test_configuration*)

Add a new workflow test to this suite.

Parameters **workflow_test_configuration** (*:class:"WorkflowTestCase"*) – the *WorkflowTestCase* instance representing the workflow test configuration

remove_workflow_test(*workflow_test*)

Remove a workflow test from this suite.

Parameters **workflow_test** (*str or :class:"WorkflowTestCase"*) – the name of the workflow test to be removed or the *WorkflowTestCase* instance representing the workflow test configuration

dump (*filename*)
Write a suite configuration to a file.

Parameters **filename** (*str*) – the absolute path of the file

Base comparator function

wft4galaxy.comparators.**base_comparator**(*actual_output_filename*, *expected_output_filename*)

1.10.2 Bioblend Wrappers

Workflow

class wft4galaxy.wrapper.**Workflow**(*definition*, *inputs*, *params*, *outputs*)
Display workflow information which are relevant to configure a workflow test.

show_inputs(*stream=<open file '<stdout>', mode 'w'>*)
Print workflow inputs to file.

show_params(*stream=<open file '<stdout>', mode 'w'>*)
Print parameters needed by workflow tools to file.

show_outputs(*stream=<open file '<stdout>', mode 'w'>*)
Print workflow outputs (indexed by workflow step) to file.

static load(*filename*, *galaxy_url=None*, *galaxy_api_key=None*, *tools_folder='.tools'*)
Return the *Workflow* instance related to the workflow defined in `filename`

Parameters

- **filename** (*str*) – the path of the `.ga` workflow definition
- **galaxy_url** (*str*) – url of your Galaxy server instance.
- **galaxy_api_key** (*str*) – an API key from your Galaxy server instance.
- **tools_folder** (*str*) – optional temp folder where tool definitions are downloaded (`.tools` by default)

Return type *Workflow*

Returns the *Workflow* instance related to the workflow defined in `filename`

History

class wft4galaxy.wrapper.**History**(*history_id*, *galaxy_url=None*, *galaxy_api_key=None*)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index

T

W