WfCommons

Release 0.6-dev

WfCommons Team

Apr 29, 2021

QUICKSTART

1 Installation **3**
- 1.1 Requirements . . . 3
- 1.2 Installation using pip . . . 3
- 1.3 Retrieving the latest unstable version . . . 3

2 The WfCommons Project **5**
- 2.1 The WfCommons JSON Format . . . 6

3 Parsing Workflow Execution Logs **7**
- 3.1 Makeflow . . . 7
- 3.2 Pegasus WMS . . . 8

4 Analyzing Instances **9**
- 4.1 Workflow Execution Instances . . . 9
- 4.2 The Instance Analyzer . . . 10
- 4.3 Examples . . . 10

5 Generating Workflows **13**
- 5.1 Workflow Recipes . . . 13
- 5.2 The Workflow Generator . . . 14
- 5.3 Examples . . . 15

6 User API Reference **17**
- 6.1 wfcommons.common . . . 17
- 6.2 wfcommons.trace . . . 20
- 6.3 wfcommons.generator . . . 23

7 Developer API Reference **43**
- 7.1 wfcommons.utils . . . 43
- 7.2 wfcommons.generator . . . 44
- 7.3 wfcommons.trace . . . 68

Python Module Index **75**

Index **77**

WfCommons

WfCommons is a community framework for enabling scientific workflow research and development. This Python package provides a collection of tools for:

- Analyzing instances of actual workflow executions;
- Producing recipes structures for creating workflow recipes for workflow generation; and
- Generating synthetic realistic workflow instances.

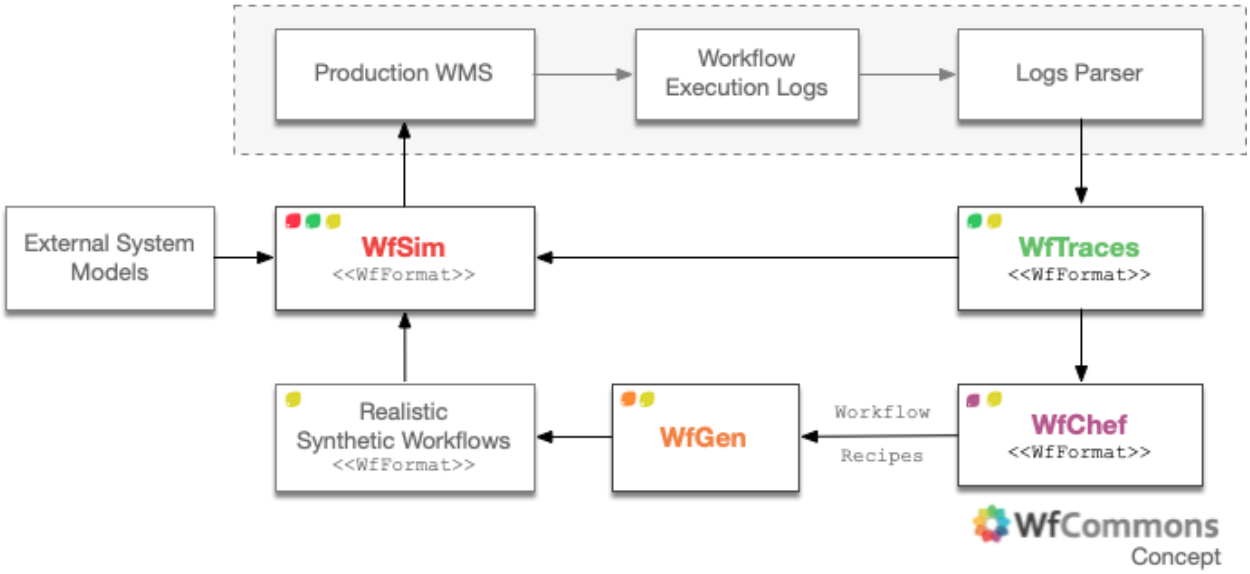

Fig. 1: The WfCommons conceptual architecture.

The source code for this WfCommons's Python package is available on GitHub. Our preferred channel to report a bug or request a feature is via WfCommons's Github Issues Track.

You can also reach the WfCommons team via our support email: support@wfcommons.org.

CHAPTER

ONE

INSTALLATION

WfCommons is available on PyPI. WfCommons requires Python3.6+ and has been tested on Linux and macOS.

1.1 Requirements

1.1.1 Graphviz

WfCommons uses pygraphviz and thus needs the graphviz package installed (version 2.16 or later). You can install graphviz easily on Linux with your favorite package manager, for example for Debian-based distributions:

```
$ sudo apt-get install graphviz libgraphviz-dev
```

and for RedHat-based distributions:

```
$ sudo yum install python-devel graphviz-devel
```

On macOS you can use the `brew` package manager:

```
$ brew install graphviz
```

1.2 Installation using pip

While `pip` can be used to install WfCommons, we suggest the following approach for reliable installation when many Python environments are available:

```
$ python3 -m pip install wfcommons
```

1.3 Retrieving the latest unstable version

If you want to use the latest WfCommons unstable version, that will contain brand new features (but also contain bugs as the stabilization work is still underway), you may consider retrieving the latest unstable version.

Cloning from WfCommons's GitHub repository:

```
$ git clone https://github.com/wfcommons/wfcommons
$ cd wfcommons
$ pip install .
```

CHAPTER

TWO

THE WFCOMMONS PROJECT

The WfCommons project is a community framework for enabling scientific workflow research and development by providing foundational tools for analyzing workflow execution instances, and generating synthetic, yet realistic, workflow instances that can be used to develop new techniques, algorithms and systems that can overcome the challenges of efficient and robust execution of ever larger workflows on increasingly complex distributed infrastructures. The figure below shows an overview of the workflow research life cycle process that integrates the three axis of the WfCommons project:

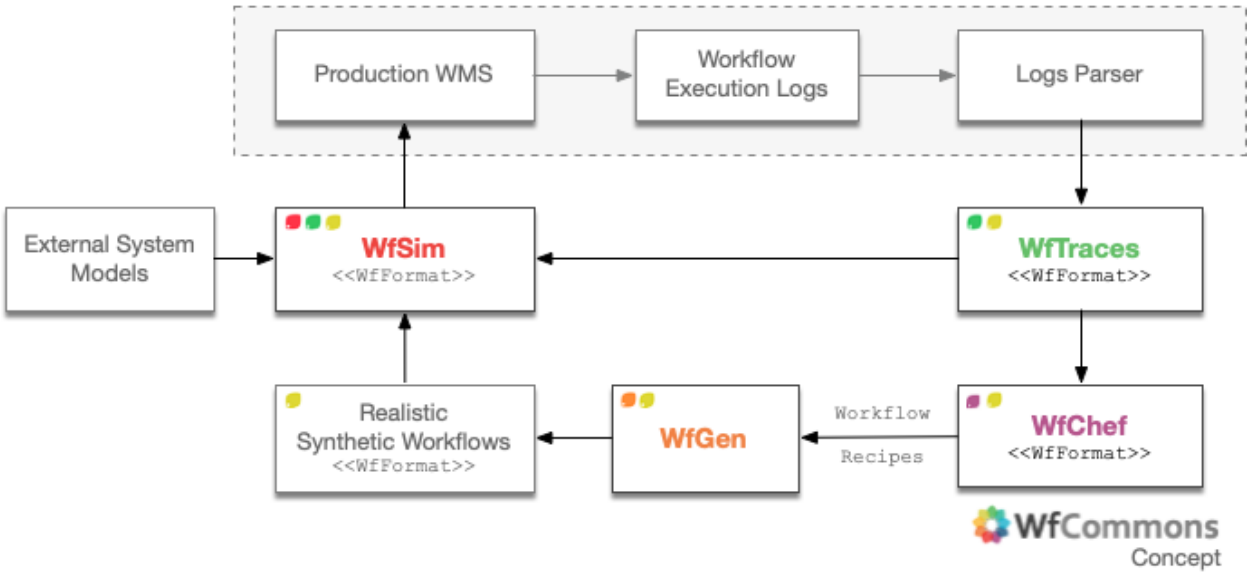

Fig. 1: The WfCommons conceptual architecture.

The *first axis* (**Workflow Instances**) of the WfCommons project targets the collection and curation of open access production workflow executions from various scientific applications shared in a common instance format (i.e., *The WfCommons JSON Format*). We keep a list of workflow execution instances in our project website.

The *second axis* (**Workflow Generator**) of the WfCommons project targets the generation of realistic synthetic workflow instances based on workflow execution profiles extracted from execution instances. We are constantly seeking for additional workflow execution instances for refining or developing new workflow recipes for the WfCommons's workflow generator.

The *third axis* (**Workflow Simulator**) of the WfCommons project fosters the use of simulation for the development, evaluation, and verification of scheduling and resource provisioning algorithms (e.g., multi-objective function optimization, etc.), evaluation of current and emerging computing platforms (e.g., clouds, IoT, extreme scale, etc.), among others. We keep a list of open source workflow management systems simulators and simulation frameworks that provide support for the WfCommons JSON format in our project website.

This Python package provides a collection of tools for:

- Analyzing instances of actual workflow executions;
- Producing recipes structures for creating workflow recipes for workflow generation; and
- Generating synthetic realistic workflow instances.

2.1 The WfCommons JSON Format

The WfCommons project uses a common format for representing workflow execution instances and generated synthetic workflows instances, so that workflow simulators and simulation frameworks (that provide support for WfCommons format) can use such instances interchangeably. This common format uses a JSON specification available in the WfCommons JSON schema GitHub repository. The current version of the WfCommons Python package uses the schema version `1.1`. The schema GitHub repository provides detailed explanation of the WfCommons JSON format (including required fields), and also a validator script for verifying the compatibility of instances.

CHAPTER

THREE

PARSING WORKFLOW EXECUTION LOGS

The most common way for obtaining instances from actual workflow executions is to parse execution logs. As part of the WfCommons project, we are constantly developing parsers for commonly used workflow management systems.

Each parser class is derived from the abstract `LogsParser` class. Thus, each parser provides a `build_workflow()` method.

3.1 Makeflow

Makeflow is a workflow system for executing large complex workflows on clusters, clouds, and grids. The Makeflow language is similar to traditional Make, so if you can write a Makefile, then you can write a Makeflow. A workflow can be just a few commands chained together, or it can be a complex application consisting of thousands of tasks. It can have an arbitrary DAG structure and is not limited to specific patterns. Makeflow is used on a daily basis to execute complex scientific applications in fields such as data mining, high energy physics, image processing, and bioinformatics. It has run on campus clusters, the Open Science Grid, NSF XSEDE machines, NCSA Blue Waters, and Amazon Web Services. Makeflow logs provide time-stamped event instances from these executions. The following example shows the analysis of Makeflow execution logs, stored in a local folder (execution dir), for a workflow execution using the *`MakeflowLogsParser`* class:

```
from wfcommons.trace import MakeflowLogsParser

# creating the parser for the Makeflow workflow
parser = MakeflowLogsParser(execution_dir='/path/to/makeflow/execution/dir/blast/
↪chameleon-small-001/'
                            resource_monitor_logs_dir='/path/to/makeflow/resource/
↪monitor/logs/dir')

# generating the workflow instance object
workflow = parser.build_workflow('workflow-test')

# writing the workflow instance to a JSON file
workflow.write_json('workflow.json')
```

Note: The *`MakeflowLogsParser`* class requires that Makeflow workflows to run with the Resource Monitor tool (e.g., execute the workflow using the `--monitor=logs`).

3.2 Pegasus WMS

Pegasus is being used in production to execute workflows for dozens of high-profile applications in a wide range of scientific domains. Pegasus provides the necessary abstractions for scientists to create workflows and allows for transparent execution of these workflows on a range of compute platforms including clusters, clouds, and national cyberinfrastructures. Workflow execution with Pegasus includes data management, monitoring, and failure handling, and is managed by HTCondor DAGMan. Individual workflow tasks are managed by a workload management framework, HTCondor, which supervises task executions on local and remote resources. Pegasus logs provide time-stamped event instances from these executions. The following example shows the analysis of Pegasus execution logs, stored in a local folder (submit dir), for a workflow execution using the *PegasusLogsParser* class:

```
from wfcommons.trace import PegasusLogsParser

# creating the parser for the Pegasus workflow
parser = PegasusLogsParser(submit_dir='/path/to/pegasus/submit/dir/seismology/
↪chameleon-100p-001/')

# generating the workflow instance object
workflow = parser.build_workflow('workflow-test')

# writing the workflow instance to a JSON file
workflow.write_json('workflow.json')
```

Warning: By default, the *PegasusLogsParser* class assumes that the submit dir is from a Pegasus execution with **version 5.0** or later. To enable parsing of Pegasus execution logs from version 4.9 or earlier, the option `legacy=True` should be used.

CHAPTER

FOUR

ANALYZING INSTANCES

Workflow execution instances have been widely used to profile and characterize workflow executions, and to build distributions of workflow execution behaviors, which are used to evaluate methods and techniques in simulation or in real conditions.

The first axis of the WfCommons project targets the analysis of actual workflow execution instances (i.e., the workflow execution profile data and characterizations) in order to build **recipes** of workflow applications. These recipes contain the necessary information for generating synthetic, yet realistic, workflow instances that resemble the structure and distribution of the original workflow executions.

A list of workflow execution instances that are compatible with *The WfCommons JSON Format* is kept constantly updated in our project website.

4.1 Workflow Execution Instances

A workflow execution instance represents an actual execution of a scientific workflow on a distributed platform (e.g., clouds, grids, HPC, etc.). In the WfCommons project, an instance is represented in a JSON file following the schema described in *The WfCommons JSON Format* section. This Python package provides an *instance loader* tool for importing workflow execution instances for analysis. For instance, the code snippet below shows how an instance can be loaded using the `Trace` class:

```
from wfcommons import Trace
trace = Trace(input_trace='/path/to/trace/file.json')
```

The `Trace` class provides a number of methods for interacting with the workflow instance, including:

- `draw()`: produces an image or a pdf file representing the instance.
- `leaves()`: gets the leaves of the workflow (i.e., the tasks without any successors).
- `roots()`: gets the roots of the workflow (i.e., the tasks without any predecessors).
- `write_dot()`: writes a dot file of the instance.

4.2 The Instance Analyzer

The `TraceAnalyzer` class provides a number of tools for analyzing collection of workflow execution instances. The goal of the `TraceAnalyzer` is to perform analyzes of one or multiple workflow execution instances, and build summaries of the analyzes per workflow' task type prefix.

Note: Although any workflow execution instance represented as a `Trace` object (i.e., compatible with *The WfCommons JSON Format*) can be appended to the `TraceAnalyzer`, we strongly recommend that only instances of a single workflow application type be appended to an analyzer object. You may though create several analyzer objects per workflow application.

The `append_trace()` method allows you to include instances for analysis. The `build_summary()` method processes all appended instances. The method applies probability distributions fitting to a series of data to find the *best* (i.e., minimizes the mean square error) probability distribution that represents the analyzed data. The method returns a summary of the analysis of instances in the form of a Python dictionary object in which keys are task prefixes (provided when invoking the method) and values describe the best probability distribution fit for tasks' runtime, and input and output data file sizes. The code excerpt below shows an example of an analysis summary showing the best fit probability distribution for runtime of the `individuals` tasks (1000Genome workflow):

```
"individuals": {
    "runtime": {
        "min": 48.846,
        "max": 192.232,
        "distribution": {
            "name": "skewnorm",
            "params": [
                11115267.652937062,
                -2.9628504044929433e-05,
                56.03957070238482
            ]
        }
    },
    ...
}
```

Workflow analysis summaries can then be used to develop *Workflow Recipes*, in which themselves are used to *generate realistic synthetic workflow instances*.

Probability distribution fits can also be plotted by using the `generate_fit_plots()` or `generate_all_fit_plots()` methods – plots will be saved as `png` files.

4.3 Examples

The following example shows the analysis of a set of instances, stored in a local folder, of a Seismology workflow. In this example, we seek for finding the best probability distribution fitting for task *prefixes* of the Seismology workflow (`sG1IterDecon`, and `wrapper_siftSTFByMisfit`), and generate all fit plots (runtime, and input and output files) into the `fits` folder using `seismology` as a prefix for each generated plot:

```
from wfcommons import Trace, TraceAnalyzer
from os import listdir
from os.path import isfile, join
```

(continues on next page)

(continued from previous page)

```
# obtaining list of instance files in the folder
INSTANCES_PATH = "/Path/to/some/instance/folder/"
instance_files = [f for f in listdir(INSTANCES_PATH) if isfile(join(INSTANCES_PATH,␣
↪f))]

# creating the instance analyzer object
analyzer = TraceAnalyzer()

# appending instance files to the instance analyzer
for instance_file in instance_files:
    instance = Trace(input_trace=INSTANCES_PATH + instance_file)
    analyzer.append_trace(instance)

# list of workflow task name prefixes to be analyzed in each instance
workflow_tasks = ['sG1IterDecon', 'wrapper_siftSTFByMisfit']

# building the instance summary
instances_summary = analyzer.build_summary(workflow_tasks, include_raw_data=True)

# generating all fit plots (runtime, and input and output files)
analyzer.generate_all_fit_plots(outfile_prefix='fits/seismology')
```

CHAPTER

FIVE

GENERATING WORKFLOWS

The second axis of the WfCommons project targets the generation of realistic synthetic workflow instances with a variety of characteristics. The *WorkflowGenerator* class uses recipes of workflows (as described in *Analyzing Instances*) for creating many different synthetic workflows based on distributions of workflow task runtime, and input and output file sizes. The resulting workflows are represented in the WfCommons JSON format, which is already supported by simulation frameworks such as WRENCH.

5.1 Workflow Recipes

The WfCommons package provides a number of *workflow recipes* for generating realistic synthetic workflow instances. Each recipe may provide their own methods for instantiating a *WorkflowRecipe* object depending on the properties that define the structure of the actual workflow. For instance, the code snippet below shows how to instantiate a recipe of the Epigenomics and 1000Genome workflows:

```
from wfcommons.generator import EpigenomicsRecipe, GenomeRecipe

# creating an Epigenomics workflow recipe
epigenomics_recipe = EpigenomicsRecipe.from_sequences(num_sequence_files=2, num_
↪lines=100, bin_size=10)

# creating a 1000Genome workflow recipe
genome_recipe = GenomeRecipe.from_num_chromosomes(num_chromosomes=3, num_
↪sequences=10000, num_populations=1)
```

All workflow recipes also provide a common method (`from_num_tasks`) for instantiating a *WorkflowRecipe* object as follows:

```
from wfcommons.generator import EpigenomicsRecipe, GenomeRecipe

# creating an Epigenomics workflow recipe
epigenomics_recipe = EpigenomicsRecipe.from_num_tasks(num_tasks=9)

# creating a 1000Genome workflow recipe
genome_recipe = GenomeRecipe.from_num_tasks(num_tasks=5)
```

Note that `num_tasks` defines the upper bound for the total number of tasks in the workflow, and that each workflow recipe may define different lower bound values so that the workflow structure is guaranteed. Please, refer to the *documentation of each workflow recipe* for the lower bound values.

The current list of available workflow recipes include:

- *BLASTRecipe*: `from wfcommons.generator import BLASTRecipe`
- *BWARecipe*: `from wfcommons.generator import BWARecipe`

- *CyclesRecipe*: `from wfcommons.generator import CyclesRecipe`
- *EpigenomicsRecipe*: `from wfcommons.generator import EpigenomicsRecipe`
- *GenomeRecipe*: `from wfcommons.generator import GenomeRecipe`
- *MontageRecipe*: `from wfcommons.generator import MontageRecipe`
- *SeismologyRecipe*: `from wfcommons.generator import SeismologyRecipe`
- *SoyKBRecipe*: `from wfcommons.generator import SoyKBRecipe`
- *SRASearchRecipe*: `from wfcommons.generator import SRASearchRecipe`

5.1.1 Increasing/Reducing Runtime and File Sizes

Workflow recipes also allow the generation of synthetic workflows with increased/reduced runtimes and/or files sizes determined by a factor provided by the user:

- `runtime_factor`: The factor of which tasks runtime will be increased/decreased.
- `input_file_size_factor`: The factor of which tasks input files size will be increased/decreased.
- `output_file_size_factor`: The factor of which tasks output files size will be increased/decreased.

The following example shows how to create a Seismology workflow recipe in which task runtime is increased by 10%, input files by 50%, and output files reduced by 20%:

```
from wfcommons.generator import SeismologyRecipe

# creating a Seismology workflow recipe with increased/decreased runtime and file␣
↪sizes
recipe = SeismologyRecipe.from_num_tasks(num_tasks=100, runtime_factor=1.1, input_
↪file_size_factor=1.5, output_file_size_factor=0.8)
```

5.2 The Workflow Generator

Synthetic workflow instances are generated using the *WorkflowGenerator* class. This class takes as input a *WorkflowRecipe* object (see above), and provides two methods for generating synthetic workflow instances:

- *build_workflow()*: generates a single synthetic workflow instance based on the workflow recipe used to instantiate the generator.
- *build_workflows()*: generates a number of synthetic workflow instances based on the workflow recipe used to instantiate the generator.

The build methods use the workflow recipe for generating realistic synthetic workflow instances, in which the workflow structure follows workflow composition rules defined in the workflow recipe, and tasks runtime, and input and output data sizes are generated according to distributions obtained from actual workflow execution instances (see *Analyzing Instances*).

Each generated instance is a represented as a *Workflow* object (which in itself is an extension of the NetworkX DiGraph class). The *Workflow* class provides two methods for writing the generated workflow instance into files:

- *write_dot()*: write a DOT file of a workflow instance.
- *write_json()*: write a JSON file of a workflow instance.

5.3 Examples

The following example generates a *Seismology* synthetic workflow instance based on the number of pair of signals to estimate earthquake STFs (`num_pairs`), builds a synthetic workflow instance, and writes the synthetic instance to a JSON file.

```
from wfcommons import WorkflowGenerator
from wfcommons.generator import SeismologyRecipe

# creating a Seismology workflow recipe based on the number
# of pair of signals to estimate earthquake STFs
recipe = SeismologyRecipe.from_num_pairs(num_pairs=10)

# creating an instance of the workflow generator with the
# Seismology workflow recipe
generator = WorkflowGenerator(recipe)

# generating a synthetic workflow instance of the Seismology workflow
workflow = generator.build_workflow()

# writing the synthetic workflow instance into a JSON file
workflow.write_json('seismology-workflow.json')
```

The example below generates a number of *Cycles* (agroecosystem) synthetic workflow instances based on the upper bound number of tasks allowed per workflow.

```
from wfcommons import WorkflowGenerator
from wfcommons.generator import CyclesRecipe

# creating a Cycles workflow recipe based on the number of tasks per workflow
recipe = CyclesRecipe.from_num_tasks(num_tasks=1000)

# creating an instance of the workflow generator with the
# Cycles workflow recipe
generator = WorkflowGenerator(recipe)

# generating 10 synthetic workflow instances of the Cycles workflow
workflows_list = generator.build_workflows(num_workflows=10)

# writing each synthetic workflow instance into a JSON file
count = 1
for workflow in workflows_list:
    workflow.write_json('cycles-workflow-{:02}.json'.format(count))
    count += 1
```

CHAPTER

SIX

USER API REFERENCE

The user API reference targets users who want to use WfCommons Python package for analyzing traces or generating realistic synthetic workflow traces, using existing workflow recipes already implemented in this Python package. Users are NOT expected to develop new *workflow recipes*.

6.1 wfcommons.common

6.1.1 wfcommons.common.file

class wfcommons.common.file.**File**(*name: str*, *size: int*, *link:* wfcommons.common.file.FileLink, *logger: Optional[logging.Logger] = None*)

Bases: object

Representation of a file.

Parameters

- **name** (*str*) – The name of the file.
- **size** (*int*) – File size in KB.
- **link** (FileLink) – Type of file link.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

as_dict() → Dict

A JSON representation of the file.

Returns A JSON object representation of the file.

Return type Dict

class wfcommons.common.file.**FileLink**(*value*)

Bases: *wfcommons.utils.NoValue*

Type of file link.

INPUT = 'input'

OUTPUT = 'output'

6.1.2 wfcommons.common.task

class wfcommons.common.task.**Task**(*name: str*, *task_type:* wfcommons.common.task.TaskType, *runtime: float*, *cores: int*, *machine: Optional[*wfcommons.common.machine.Machine*] = None*, *args: List[str] = []*, *avg_cpu: Optional[float] = None*, *bytes_read: Optional[int] = None*, *bytes_written: Optional[int] = None*, *memory: Optional[int] = None*, *energy: Optional[int] = None*, *avg_power: Optional[float] = None*, *priority: Optional[int] = None*, *files: List[*wfcommons.common.file.File*] = []*, *logger: Optional[logging.Logger] = None*)

Bases: object

Representation of a task.

Parameters

- **name** (*str*) – The name of the task.
- **task_type** (TaskType) – The type of the task.
- **runtime** (*float*) – Task runtime in seconds.
- **cores** (*int*) – Number of cores required by the task.
- **machine** (Machine) – Machine on which is the task has been executed.
- **args** (*List[str]*) – List of task arguments.
- **avg_cpu** (*float*) – Average CPU utilization in %.
- **bytes_read** (*int*) – Total bytes read in KB.
- **bytes_written** (*int*) – Total bytes written in KB.
- **memory** (*int*) – Memory (resident set) size of the process in KB.
- **energy** (*int*) – Total energy consumption in kWh.
- **avg_power** (*float*) – Average power consumption in W.
- **priority** (*int*) – Task priority.
- **files** (*List[File]*) – List of input/output files used by the task.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

as_dict() → Dict

A JSON representation of the task.

Returns A JSON object representation of the task.

Return type Dict

class wfcommons.common.task.**TaskType**(*value*)

Bases: *wfcommons.utils.NoValue*

Task type.

AUXILIARY = 'auxiliary'

COMPUTE = 'compute'

TRANSFER = 'transfer'

6.1.3 wfcommons.common.machine

class wfcommons.common.machine.**Machine**(*name: str, cpu: Dict[str, Union[int, str]], system: Optional[wfcommons.common.machine.MachineSystem] = None, architecture: Optional[str] = None, memory: Optional[int] = None, release: Optional[str] = None, hashcode: Optional[str] = None, logger: Optional[logging.Logger] = None*)

Bases: object

Representation of one compute machine.

Parameters

- **name** (*str*) – Machine node name.
- **cpu** (*Dict[str, Union[int, str]]*) – A dictionary containing information about the CPU specification. Must at least contains two fields: *count* (number of CPU cores) and speed (CPU speed of each core in MHz).

```
cpu = {
    'count': 48,
    'speed': 1200
}
```

- **system** (MachineSystem) – Machine system (linux, macos, windows).
- **architecture** (*str*) – Machine architecture (e.g., x86_64, ppc).
- **memory** (*int*) – Total machine's RAM memory in KB.
- **release** (*str*) – Machine release.
- **hashcode** (*str*) – MD5 Hashcode for the Machine.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

as_dict() → Dict

A JSON representation of the machine.

Returns A JSON object representation of the machine.

Return type Dict

class wfcommons.common.machine.**MachineSystem**(*value*)

Bases: *wfcommons.utils.NoValue*

Machine system type.

LINUX = 'linux'

MACOS = 'macos'

WINDOWS = 'windows'

6.1.4 wfcommons.common.workflow

class wfcommons.common.workflow.**Workflow**(*name: str*, *description: Optional[str] = None*, *wms_name: Optional[str] = None*, *wms_version: Optional[str] = None*, *wms_url: Optional[str] = None*, *executed_at: Optional[str] = None*, *makespan: Optional[int] = 0.0*)

Bases: networkx.classes.digraph.DiGraph

Representation of a workflow. The workflow representation is an extension of the NetworkX DiGraph class.

Parameters

- **name** (*str*) – Workflow name.
- **description** (*str*) – Workflow trace description.
- **wms_name** (*str*) – WMS name.
- **wms_version** (*str*) – WMS version.
- **wms_url** (*str*) – URL for the WMS website.
- **executed_at** (*str*) – Workflow start timestamp in the ISO 8601 format.
- **makespan** (*int*) – Workflow makespan in seconds.

write_dot(*dot_filename: Optional[str] = None*) → None

Write a dot file of the workflow trace.

Parameters **dot_filename** (*str*) – DOT output file name.

write_json(*json_filename: Optional[str] = None*) → None

Write a JSON file of the workflow trace.

Parameters **json_filename** (*str*) – JSON output file name.

6.2 wfcommons.trace

6.2.1 wfcommons.trace.trace

class wfcommons.trace.trace.**Trace**(*input_trace: str*, *schema_file: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

Bases: object

Representation of one execution of one workflow on a set of machines

```
Trace(input_trace = 'trace.json')
```

Parameters

- **input_trace** (*str*) – The JSON trace.
- **schema_file** (*str*) – The path to the JSON schema that defines the trace. If no schema file is provided, it will look for a local copy of the WfCommons schema, and if not available it will fetch the latest schema from the WfCommons schema GitHub repository.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

draw(*output: Optional[str] = None*, *extension: str = 'pdf'*) → None

Produce an image or a pdf file representing the trace.

Parameters

- **output** (*str*) – Name of the output file.
- **extension** – Type of the file extension (`pdf`, `png`, or `svg`).

leaves() → List[str]
Get the leaves of the workflow (i.e., the tasks without any successors).

Returns List of leaves

Return type List[str]

roots() → List[str]
Get the roots of the workflow (i.e., the tasks without any predecessors).

Returns List of roots

Return type List[str]

write_dot(*output: Optional[str] = None*) → None
Write a dot file of the trace.

Parameters **output** (*str*) – The output `dot` file name (optional).

6.2.2 wfcommons.trace.trace_analyzer

class wfcommons.trace.trace_analyzer.**TraceAnalyzer**(*logger: Optional[logging.Logger] = None*)

Bases: `object`

Set of tools for analyzing collections of traces.

Parameters **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

append_trace(*trace:* wfcommons.trace.trace.Trace) → None
Append a workflow trace object to the trace analyzer.

```
trace = Trace(input_trace = 'trace.json', schema = 'schema.json')
trace_analyzer = TraceAnalyzer()
trace_analyzer.append_trace(trace)
```

Parameters **trace** (`Trace`) – A workflow trace object.

build_summary(*tasks_list: List[str]*, *include_raw_data: Optional[bool] = True*) → Dict[str, Dict[str, Any]]
Analyzes appended traces and produce a summary of the analysis per task prefix.

```
workflow_tasks = ['sG1IterDecon', 'wrapper_siftSTFByMisfit']
traces_summary = trace_analyzer.build_summary(workflow_tasks, include_raw_
↪data=False)
```

Parameters

- **tasks_list** (*List[str]*) – List of workflow tasks prefix (e.g., mProject, sol2sanger, add_replace)
- **include_raw_data** (*bool*) – Whether to include the raw data in the trace summary.

Returns A summary of the analysis of traces in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Dict[str, Any]]

generate_all_fit_plots(*outfile_prefix: Optional[str] = None*) → None
Produce fit plots as images for each entry of the summary analysis. For entries in which there are no distribution (i.e., constant value), no plot will be generated.

Parameters **outfile_prefix** (*str*) – Prefix to be attached to each generated plot file name (optional).

generate_fit_plots(*trace_element: wfcommons.trace.trace_analyzer.TraceElement, outfile_prefix: Optional[str] = None*) → None
Produce fit plots as images for each entry of a trace element generated by the summary analysis. For entries in which there are no distribution (i.e., constant value), no plot will be generated.

Parameters

- **trace_element** (TraceElement) – Workflow element for which the fit plots will be generated.
- **outfile_prefix** (*str*) – Prefix to be attached to each generated plot file name (optional).

class wfcommons.trace.trace_analyzer.**TraceElement**(*value*)
Bases: *wfcommons.utils.NoValue*

An enumeration.

INPUT = ('input', 'Input File Size (bytes)')

OUTPUT = ('output', 'Input File Size (bytes)')

RUNTIME = ('runtime', 'Runtime (s)')

6.2.3 wfcommons.trace.logs.makeflow

class wfcommons.trace.logs.makeflow.**MakeflowLogsParser**(*execution_dir: str, resource_monitor_logs_dir: str, description: Optional[str] = None, logger: Optional[logging.Logger] = None*)
Bases: wfcommons.trace.logs.abstract_logs_parser.LogsParser

Parse Makeflow submit directory to generate workflow trace.

Parameters

- **execution_dir** (*str*) – Makeflow workflow execution directory (contains .mf and .makeflowlog files).
- **resource_monitor_logs_dir** (*str*) – Resource Monitor log files directory.
- **description** (*str*) – Workflow trace description.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*
Create workflow trace based on the workflow execution logs.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A workflow trace object.

Return type *Workflow*

6.2.4 wfcommons.trace.logs.pegasus

class wfcommons.trace.logs.pegasus.**PegasusLogsParser**(*submit_dir: str*, *description: Optional[str] = None*, *ignore_auxiliary: Optional[bool] = True*, *legacy: Optional[bool] = False*, *logger: Optional[logging.Logger] = None*)

Bases: wfcommons.trace.logs.abstract_logs_parser.LogsParser

Parse Pegasus submit directory to generate workflow trace.

Parameters

- **submit_dir** (*str*) – Pegasus submit directory.
- **legacy** (*bool*) – Whether the submit directory is from a Pegasus 4.x version.
- **description** (*str*) – Workflow trace description.
- **ignore_auxiliary** (*bool*) – Ignore auxiliary jobs.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Create workflow trace based on the workflow execution logs.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A workflow trace object.

Return type *Workflow*

6.3 wfcommons.generator

6.3.1 wfcommons.generator.generator

class wfcommons.generator.generator.**WorkflowGenerator**(*workflow_recipe: wfcommons.generator.workflow.abstract_recipe.WorkflowRe*
logger: Optional[logging.Logger] = None)

Bases: object

A generator of synthetic workflow traces based on workflow recipes obtained from the analysis of real workflow execution traces.

Parameters

- **workflow_recipe** (WorkflowRecipe) – The workflow recipe to be used for this generator.

- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace based on the workflow recipe used to instantiate the generator.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A synthetic workflow trace object.

Return type *Workflow*

build_workflows(*num_workflows: int*) → List[*wfcommons.common.workflow.Workflow*]

Generate a number of synthetic workflow traces based on the workflow recipe used to instantiate the generator.

Parameters **num_workflows** (*int*) – The number of workflows to be generated.

Returns A list of synthetic workflow trace objects.

Return type List[*Workflow*]

6.3.2 wfcommons.generator.workflow.blast_recipe

class wfcommons.generator.workflow.blast_recipe.**BLASTRecipe**(*num_subsample: Optional[int] = 2, data_footprint: Optional[int] = 0, num_tasks: Optional[int] = 5, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

A BLAST workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_subsample** (*int*) – The number of subsample the reference file will be split.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a BLAST workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_subsample(*num_subsample: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.blast_recipe.BLASTRecipe*

Instantiate a BLAST workflow recipe that will generate synthetic workflows using the defined number of subsample.

Parameters

- **num_subsample** (*int*) – The number of subsample the reference file will be split.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A BLAST workflow recipe object that will generate synthetic workflows using the defined number of subsample.

Return type *BLASTRecipe*

classmethod from_num_tasks(*num_tasks: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.blast_recipe.BLASTRecipe*

Instantiate a BLAST workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 5).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A BLAST workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *BLASTRecipe*

num_subsample: int

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

6.3.3 wfcommons.generator.workflow.bwa_recipe

class wfcommons.generator.workflow.bwa_recipe.**BWARecipe**(*num_subsample: Optional[int] = 2, data_footprint: Optional[int] = 0, num_tasks: Optional[int] = 5, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

A BLAST workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_subsample** (*int*) – The number of subsample the reference file will be split.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a BWA workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_subsample(*num_subsample: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.bwa_recipe.BWARecipe*

Instantiate a BWA workflow recipe that will generate synthetic workflows using the defined number of subsample.

Parameters

- **num_subsample** (*int*) – The number of subsample the reference file will be split.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.

- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A BWA workflow recipe object that will generate synthetic workflows using the defined number of subsample.

Return type *BWARecipe*

classmethod from_num_tasks(*num_tasks: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.bwa_recipe.BWARecipe*

Instantiate a BWA workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 6).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A BWA workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *BWARecipe*

num_subsample: int

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

6.3.4 wfcommons.generator.workflow.cycles_recipe

class wfcommons.generator.workflow.cycles_recipe.**CyclesRecipe**(*num_points: Optional[int] = 1, num_crops: Optional[int] = 1, num_params: Optional[int] = 4, data_footprint: Optional[int] = 0, num_tasks: Optional[int] = 7, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

A Cycles workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_points** (*int*) – The number of points of the spatial grid cell.
- **num_crops** (*int*) – The number of crops being evaluated.
- **num_params** (*int*) – The number of parameter values from the simulation matrix.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a Cycles workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_tasks(*num_tasks: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.cycles_recipe.CyclesRecipe*

Instantiate a Cycles workflow recipe that will generate synthetic workflows up to the total number of tasks

provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 7).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Cycles workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *CyclesRecipe*

classmethod from_points_and_crops(*num_points: int*, *num_crops: int*, *num_params: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.cycles_recipe.CyclesRecipe*

Instantiate a Cycles workflow recipe that will generate synthetic workflows using the defined number of points, crops, and params.

Parameters

- **num_points** (*int*) – The number of points of the spatial grid cell.
- **num_crops** (*int*) – The number of crops being evaluated.
- **num_params** (*int*) – The number of parameter values from the simulation matrix.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Cycles workflow recipe object that will generate synthetic workflows using the defined number of points, crops, and params.

Return type *CyclesRecipe*

num_crops: int

num_params: int

num_points: int

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

6.3.5 wfcommons.generator.workflow.epigenomics_recipe

class wfcommons.generator.workflow.epigenomics_recipe.**EpigenomicsRecipe**(*num_sequence_files: Optional[int] = 1*, *num_lines: Optional[int] = 10*, *bin_size: Optional[int] = 10*, *data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 9*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

An Epigenomics workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_sequence_files** (*int*) – Number of FASTQ files processed by the workflow.
- **num_lines** (*int*) – Number of lines in each FASTQ file.
- **bin_size** (*int*) – Number of DNA and protein sequence information to be processed by each computational task.

- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

bin_size: int

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of an Epigenomics workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_tasks(*num_tasks: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.epigenomics_recipe.EpigenomicsRecipe*

Instantiate an Epigenomics workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 9).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns An Epigenomics workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *EpigenomicsRecipe*

classmethod from_sequences(*num_sequence_files: int, num_lines: int, bin_size: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.epigenomics_recipe.EpigenomicsRecipe*

Instantiate an Epigenomics workflow recipe that will generate synthetic workflows using the defined number of sequence files, lines, and bin size.

Parameters

- **num_sequence_files** (*int*) – Number of FASTQ files processed by the workflow.
- **num_lines** (*int*) – Number of lines in each FASTQ file.

- **bin_size** (*int*) – Number of DNA and protein sequence information to be processed by each computational task.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns An Epigenomics workflow recipe object that will generate synthetic workflows using the defined number of sequence files, lines, and bin size.

Return type *EpigenomicsRecipe*

num_lines: int

num_sequence_files: int

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

6.3.6 wfcommons.generator.workflow.genome_recipe

class wfcommons.generator.workflow.genome_recipe.**GenomeRecipe**(*num_chromosomes: Optional[int] = 1*, *num_sequences: Optional[int] = 1*, *num_populations: Optional[int] = 1*, *data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 5*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

A 1000Genome workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_chromosomes** (*int*) – The number of chromosomes evaluated in the workflow execution.
- **num_sequences** (*int*) – The number of sequences per chromosome file.
- **num_populations** (*int*) – The number of populations being evaluated.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a 1000Genome workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_chromosomes(*num_chromosomes: int, num_sequences: int, num_populations: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.genome_recipe.GenomeRecipe*

Instantiate a 1000Genome workflow recipe that will generate synthetic workflows using the defined number of chromosomes, sequences, and populations.

Parameters

- **num_chromosomes** (*int*) – The number of chromosomes evaluated in the workflow execution.
- **num_sequences** (*int*) – The number of sequences per chromosome file.
- **num_populations** (*int*) – The number of populations being evaluated.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A 1000Genome workflow recipe object that will generate synthetic workflows using the defined number of chromosomes, sequences, and populations.

Return type *GenomeRecipe*

classmethod from_num_tasks(*num_tasks: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.genome_recipe.GenomeRecipe*

Instantiate a 1000Genome workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 5).

- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A 1000Genome workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *GenomeRecipe*

num_chromosomes: int

num_populations: int

num_sequences: int

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

6.3.7 wfcommons.generator.workflow.montage_recipe

class wfcommons.generator.workflow.montage_recipe.**MontageDataset**(*value*)

Bases: *wfcommons.utils.NoValue*

An enumeration of Montage datasets.

DSS = 'dss'

TWOMASS = '2mass'

class wfcommons.generator.workflow.montage_recipe.**MontageRecipe**(*dataset: Optional[wfcommons.generator.workflow.r = <MontageDataset.DSS>, num_bands: Optional[int] = 1, degree: Optional[float] = 0.5, data_footprint: Optional[int] = 0, num_tasks: Optional[int] = 133, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*, wfcommons.generator.workflow.montage_recipe._MontagetaskRatios

A Montage workflow recipe class for creating synthetic workflow traces. In this workflow recipe, traces will follow different recipes for different *MontageDataset*.

Parameters

- **dataset** (*MontageDataset*) – The dataset to use for the mosaic (e.g., 2mass, dss).
- **num_bands** (*int*) – The number of bands (e.g., red, blue, and green) used by the workflow.
- **degree** (*float*) – The size (in degrees) to be used for the width/height of the final mosaic.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a Montage workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

dataset: ***MontageDataset***

degree: **Optional[float]**

classmethod from_degree(*dataset:* wfcommons.generator.workflow.montage_recipe.MontageDataset, *num_bands: int*, *degree: float*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.montage_recipe.MontageRecipe*

Instantiate a Montage workflow recipe that will generate synthetic workflows using the defined dataset, number of bands, and degree.

Parameters

- **dataset** (MontageDataset) – The dataset to use for the mosaic (e.g., 2mass, dss).
- **num_bands** (*int*) – The number of bands (e.g., red, blue, and green) used by the workflow (at least 1).
- **degree** (*float*) – The size (in degrees) to be used for the width/height of the final mosaic (at least 0.5).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Montage workflow recipe object that will generate synthetic workflows using the defined dataset, number of bands, and degree.

Return type *MontageRecipe*

classmethod from_num_tasks(*num_tasks: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.montage_recipe.MontageRecipe*

Instantiate a Montage workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 133).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Montage workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *MontageRecipe*

num_bands: **Optional[int]**

tasks_files: **Dict[str, List[File]]**

workflows: **List[Workflow]**

6.3.8 wfcommons.generator.workflow.seismology_recipe

class wfcommons.generator.workflow.seismology_recipe.**SeismologyRecipe**(*num_pairs: Optional[int] = 2*, *data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

A Seismology workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_pairs** (*int*) – The number of pair of signals to estimate earthquake STFs.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a Seismology workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_pairs(*num_pairs: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.seismology_recipe.SeismologyRecipe*

Instantiate a Seismology workflow recipe that will generate synthetic workflows using the defined number of pairs.

Parameters

- **num_pairs** (*int*) – The number of pair of signals to estimate earthquake STFs (at least 2).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Seismology workflow recipe object that will generate synthetic workflows using the defined number of pairs.

Return type *SeismologyRecipe*

classmethod from_num_tasks(*num_tasks: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.seismology_recipe.SeismologyRecipe*

Instantiate a Seismology workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 3).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Seismology workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *SeismologyRecipe*

num_pairs: int

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

6.3.9 wfcommons.generator.workflow.soykb_recipe

class wfcommons.generator.workflow.soykb_recipe.**SoyKBRecipe**(*num_fastq_files: Optional[int] = 2, num_chromosomes: Optional[int] = 1, data_footprint: Optional[int] = 0, num_tasks: Optional[int] = 14, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

A SoyKB workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_fastq_files** (*int*) – The number of FASTQ files to be analyzed.
- **num_chromosomes** (*int*) – The number of chromosomes.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a SoyKB workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_tasks(*num_tasks: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.soykb_recipe.SoyKBRecipe*

Instantiate a SoyKB workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 14).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A SoyKB workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *SoyKBRecipe*

classmethod from_sequences(*num_fastq_files: int*, *num_chromosomes: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.soykb_recipe.SoyKBRecipe*

Instantiate a SoyKB workflow recipe that will generate synthetic workflows using the defined number of FASTQ files and chromosomes.

Parameters

- **num_fastq_files** (*int*) – The number of FASTQ files to be analyzed (at least 2).
- **num_chromosomes** (*int*) – The number of chromosomes (range [1,22].
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A SoyKB workflow recipe object that will generate synthetic workflows using the defined number of FASTQ files and chromosomes.

Return type *SoyKBRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

6.3.10 wfcommons.generator.workflow.srasearch_recipe

class wfcommons.generator.workflow.srasearch_recipe.**SRASearchRecipe**(*num_accession: Optional[int] = 2, data_footprint: Optional[int] = 0, num_tasks: Optional[int] = 3, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

An SRA Search workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_accession** (*int*) – The number of NCBI accession numbers.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of an SRA Search workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_accession(*num_accession: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.srasearch_recipe.SRASearchRecipe*

Instantiate an SRA Search workflow recipe that will generate synthetic workflows using the defined number of pairs.

Parameters

- **num_accession** (*int*) – The number of NCBI accession numbers.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns An SRA Search workflow recipe object that will generate synthetic workflows using the defined number of pairs.

Return type *SRASearchRecipe*

classmethod from_num_tasks(*num_tasks: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.srasearch_recipe.SRASearchRecipe*

Instantiate an SRA Search workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 6).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns An SRA Search workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *SRASearchRecipe*

num_accession: int

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

CHAPTER

SEVEN

DEVELOPER API REFERENCE

The developer API reference targets developers and researchers who want to contribute to the WfCommons project by, for example, developing novel techniques for trace analysis, developing new *workflow recipes*, etc. The developer API reference documentation includes detailed information for interacting with all classes and methods that compose this Python package.

7.1 wfcommons.utils

class wfcommons.utils.**NoValue**(*value*)

Bases: enum.Enum

An enumeration.

wfcommons.utils.**best_fit_distribution**(*data: List[float]*, *logger: Optional[logging.Logger] = None*) → Tuple

Fit a list of values to a distribution.

Parameters

- **data** (*List[float]*) – List of values to be fitted to a distribution.
- **logger** (*Logger*) – The logger uses to output debug information.

Returns The name of the distribution and its parameters.

Return type Tuple

wfcommons.utils.**generate_rvs**(*distribution: Dict*, *min_value: float*, *max_value: float*) → float

Generate a random variable from a distribution.

Parameters

- **distribution** (*Dict*) – Distribution dictionary (name and parameters).
- **min_value** (*float*) – Minimum value accepted as a random variable.
- **max_value** (*float*) – Maximum value accepted as a random variable.

Returns Random variable generated from a distribution.

Return type float

wfcommons.utils.**ncr**(*n: int*, *r: int*) → int

Calculate the number of combinations.

Parameters

- **n** (*int*) – The number of items.

- **r** (*int*) – The number of items being chosen at a time.

Returns The number of combinations.

Return type int

wfcommons.utils.**read_json**(*trace_filename: str*) → Dict[str, Any]

Read the JSON from the file path.

Parameters **trace_filename** (*str*) – The absolute path of the trace file.

Returns The json object loaded with json data from the file

Return type Dict[str, Any]

7.2 wfcommons.generator

7.2.1 wfcommons.generator.generator

class wfcommons.generator.generator.**WorkflowGenerator**(*workflow_recipe:* wfcommons.generator.workflow.abstract_recipe.WorkflowRe *logger: Optional[logging.Logger] = None*)

Bases: object

A generator of synthetic workflow traces based on workflow recipes obtained from the analysis of real workflow execution traces.

Parameters

- **workflow_recipe** (WorkflowRecipe) – The workflow recipe to be used for this generator.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace based on the workflow recipe used to instantiate the generator.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A synthetic workflow trace object.

Return type *Workflow*

build_workflows(*num_workflows: int*) → List[*wfcommons.common.workflow.Workflow*]

Generate a number of synthetic workflow traces based on the workflow recipe used to instantiate the generator.

Parameters **num_workflows** (*int*) – The number of workflows to be generated.

Returns A list of synthetic workflow trace objects.

Return type List[*Workflow*]

7.2.2 wfcommons.generator.workflow.abstract_recipe

class wfcommons.generator.workflow.abstract_recipe.**WorkflowRecipe**(*name: str*, *data_footprint: Optional[int]*, *num_tasks: Optional[int]*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*, *logger: Optional[logging.Logger] = None*)

Bases: abc.ABC

An abstract class of workflow recipes for creating synthetic workflow traces.

Parameters

- **name** (*str*) – The workflow recipe name.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

_abc_impl = <_abc_data object>

_generate_file(*extension: str*, *recipe: Dict[str, Any]*, *link:* wfcommons.common.file.FileLink) → *wfcommons.common.file.File*

Generate a file according to a file recipe.

Parameters

- **extension** (*str*) –
- **recipe** (*Dict[str, Any]*) – Recipe for generating the file.
- **link** (FileLink) – Type of file link.

Returns The generated file.

Return type *File*

_generate_files(*task_id: str*, *recipe: Dict[str, Any]*, *link:* wfcommons.common.file.FileLink, *files_recipe: Optional[Dict[*wfcommons.common.file.FileLink*, Dict[str, int]]] = None*) → None

Generate files for a specific task ID.

Parameters

- **task_id** (*str*) – task ID.
- **recipe** (*Dict[str, Any]*) – Recipe for generating the task.
- **link** (*FileLink*) – Type of file link.
- **files_recipe** (*Dict[FileLink, Dict[str, int]]*) – Recipe for generating task files.

_generate_task(*task_name: str*, *task_id: str*, *input_files: Optional[List[*wfcommons.common.file.File*]] = None*, *files_recipe: Optional[Dict[*wfcommons.common.file.FileLink*, Dict[str, int]]] = None*) → *wfcommons.common.task.Task*

Generate a synthetic task.

Parameters

- **task_name** (*str*) – task name.
- **task_id** (*str*) – task ID.
- **input_files** (*List[File]*) – List of input files to be included.
- **files_recipe** (*Dict[FileLink, Dict[str, int]]*) – Recipe for generating task files.

Returns A task object.

Return type task

_generate_task_name(*prefix: str*) → str

Generate a task name from a prefix appended with an ID.

Parameters **prefix** (*str*) – task prefix.

Returns task name from prefix appended with an ID.

Return type str

_get_files_by_task_and_link(*task_id: str*, *link:* wfcommons.common.file.FileLink) → List[*wfcommons.common.file.File*]

Get the list of files for a task ID and link type.

Parameters

- **task_id** (*str*) – task ID.
- **link** (*FileLink*) – Type of file link.

Returns List of files for a task ID and link type.

Return type List[*File*]

abstract _workflow_recipe() → Dict[str, Any]

Recipe for generating synthetic traces for a workflow. Recipes can be generated by using the *TraceAnalyzer*.

Returns A recipe in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Any]

abstract build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A synthetic workflow trace object.

Return type *Workflow*

abstract classmethod from_num_tasks(*num_tasks: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

Instantiate a workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *WorkflowRecipe*

7.2.3 wfcommons.generator.workflow.blast_recipe

class wfcommons.generator.workflow.blast_recipe.**BLASTRecipe**(*num_subsample: Optional[int] = 2*, *data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 5*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

A BLAST workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_subsample** (*int*) – The number of subsample the reference file will be split.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict

Recipe for generating synthetic traces of the BLAST workflow. Recipes can be generated by using the *TraceAnalyzer*.

Returns A recipe in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Any]

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a BLAST workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_subsample(*num_subsample: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.blast_recipe.BLASTRecipe*

Instantiate a BLAST workflow recipe that will generate synthetic workflows using the defined number of subsample.

Parameters

- **num_subsample** (*int*) – The number of subsample the reference file will be split.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A BLAST workflow recipe object that will generate synthetic workflows using the defined number of subsample.

Return type *BLASTRecipe*

classmethod from_num_tasks(*num_tasks: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.blast_recipe.BLASTRecipe*

Instantiate a BLAST workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 5).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A BLAST workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *BLASTRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.4 wfcommons.generator.workflow.bwa_recipe

class wfcommons.generator.workflow.bwa_recipe.**BWARecipe**(*num_subsample: Optional[int] = 2*, *data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 5*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

A BLAST workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_subsample** (*int*) – The number of subsample the reference file will be split.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.

- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict

Recipe for generating synthetic traces of the BWA workflow. Recipes can be generated by using the *TraceAnalyzer*.

Returns A recipe in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Any]

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a BWA workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_subsample(*num_subsample: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.bwa_recipe.BWARecipe*

Instantiate a BWA workflow recipe that will generate synthetic workflows using the defined number of subsample.

Parameters

- **num_subsample** (*int*) – The number of subsample the reference file will be split.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A BWA workflow recipe object that will generate synthetic workflows using the defined number of subsample.

Return type *BWARecipe*

classmethod from_num_tasks(*num_tasks: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.bwa_recipe.BWARecipe*

Instantiate a BWA workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 6).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.

- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A BWA workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *BWARecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.5 wfcommons.generator.workflow.cycles_recipe

class wfcommons.generator.workflow.cycles_recipe.**CyclesRecipe**(*num_points: Optional[int] = 1, num_crops: Optional[int] = 1, num_params: Optional[int] = 4, data_footprint: Optional[int] = 0, num_tasks: Optional[int] = 7, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

A Cycles workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_points** (*int*) – The number of points of the spatial grid cell.
- **num_crops** (*int*) – The number of crops being evaluated.
- **num_params** (*int*) – The number of parameter values from the simulation matrix.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict
Recipe for generating synthetic traces of the Cycles workflow. Recipes can be generated by using the *TraceAnalyzer*.

Returns A recipe in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Any]

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*
Generate a synthetic workflow trace of a Cycles workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_tasks(*num_tasks: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.cycles_recipe.CyclesRecipe*
Instantiate a Cycles workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 7).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Cycles workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *CyclesRecipe*

classmethod from_points_and_crops(*num_points: int, num_crops: int, num_params: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.cycles_recipe.CyclesRecipe*
Instantiate a Cycles workflow recipe that will generate synthetic workflows using the defined number of points, crops, and params.

Parameters

- **num_points** (*int*) – The number of points of the spatial grid cell.
- **num_crops** (*int*) – The number of crops being evaluated.
- **num_params** (*int*) – The number of parameter values from the simulation matrix.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.

- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Cycles workflow recipe object that will generate synthetic workflows using the defined number of points, crops, and params.

Return type *CyclesRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.6 wfcommons.generator.workflow.epigenomics_recipe

class wfcommons.generator.workflow.epigenomics_recipe.**EpigenomicsRecipe**(*num_sequence_files: Optional[int] = 1*, *num_lines: Optional[int] = 10*, *bin_size: Optional[int] = 10*, *data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 9*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

An Epigenomics workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_sequence_files** (*int*) – Number of FASTQ files processed by the workflow.
- **num_lines** (*int*) – Number of lines in each FASTQ file.
- **bin_size** (*int*) – Number of DNA and protein sequence information to be processed by each computational task.

- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict

Recipe for generating synthetic traces of the Epigenomics workflow. Recipes can be generated by using the *TraceAnalyzer*.

Returns A recipe in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Any]

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of an Epigenomics workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_tasks(*num_tasks: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.epigenomics_recipe.EpigenomicsRecipe*

Instantiate an Epigenomics workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 9).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns An Epigenomics workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *EpigenomicsRecipe*

classmethod from_sequences(*num_sequence_files: int, num_lines: int, bin_size: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.epigenomics_recipe.EpigenomicsRecipe*

Instantiate an Epigenomics workflow recipe that will generate synthetic workflows using the defined number of sequence files, lines, and bin size.

Parameters

- **num_sequence_files** (*int*) – Number of FASTQ files processed by the workflow.
- **num_lines** (*int*) – Number of lines in each FASTQ file.
- **bin_size** (*int*) – Number of DNA and protein sequence information to be processed by each computational task.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns An Epigenomics workflow recipe object that will generate synthetic workflows using the defined number of sequence files, lines, and bin size.

Return type *EpigenomicsRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.7 wfcommons.generator.workflow.genome_recipe

class wfcommons.generator.workflow.genome_recipe.**GenomeRecipe**(*num_chromosomes: Optional[int] = 1*, *num_sequences: Optional[int] = 1*, *num_populations: Optional[int] = 1*, *data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 5*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

A 1000Genome workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_chromosomes** (*int*) – The number of chromosomes evaluated in the workflow execution.
- **num_sequences** (*int*) – The number of sequences per chromosome file.

- **num_populations** (*int*) – The number of populations being evaluated.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

_abc_impl = <_abc_data object>

_get_populations_files_recipe(*index: int*) → Dict[*wfcommons.common.file.FileLink*, Dict[str, int]]

Get the recipe for generating a population file.

Parameters **index** (*int*) – Index of the population in the list.

Returns Recipe for generating a population file.

Return type Dict[*FileLink*, Dict[str, int]]

_workflow_recipe() → Dict

Recipe for generating synthetic traces of the 1000Genome workflow. Recipes can be generated by using the *TraceAnalyzer*.

Returns A recipe in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Any]

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a 1000Genome workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_chromosomes(*num_chromosomes: int*, *num_sequences: int*, *num_populations: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.genome_recipe.GenomeRecipe*

Instantiate a 1000Genome workflow recipe that will generate synthetic workflows using the defined number of chromosomes, sequences, and populations.

Parameters

- **num_chromosomes** (*int*) – The number of chromosomes evaluated in the workflow execution.
- **num_sequences** (*int*) – The number of sequences per chromosome file.
- **num_populations** (*int*) – The number of populations being evaluated.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.

- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A 1000Genome workflow recipe object that will generate synthetic workflows using the defined number of chromosomes, sequences, and populations.

Return type *GenomeRecipe*

classmethod from_num_tasks(*num_tasks: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.genome_recipe.GenomeRecipe*

Instantiate a 1000Genome workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 5).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A 1000Genome workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *GenomeRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.8 wfcommons.generator.workflow.montage_recipe

class wfcommons.generator.workflow.montage_recipe.**MontageDataset**(*value*)

Bases: *wfcommons.utils.NoValue*

An enumeration of Montage datasets.

DSS = 'dss'

TWOMASS = '2mass'

class wfcommons.generator.workflow.montage_recipe.**MontageRecipe**(*dataset: Optional[wfcommons.generator.workflow.r = <MontageDataset.DSS>, num_bands: Optional[int] = 1, degree: Optional[float] = 0.5, data_footprint: Optional[int] = 0, num_tasks: Optional[int] = 133, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*, wfcommons.generator.workflow.montage_recipe._MontagetaskRatios

A Montage workflow recipe class for creating synthetic workflow traces. In this workflow recipe, traces will follow different recipes for different *MontageDataset*.

Parameters

- **dataset** (*MontageDataset*) – The dataset to use for the mosaic (e.g., 2mass, dss).
- **num_bands** (*int*) – The number of bands (e.g., red, blue, and green) used by the workflow.
- **degree** (*float*) – The size (in degrees) to be used for the width/height of the final mosaic.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict

Recipe for generating synthetic traces of the Montage workflow. Recipes can be generated by using the *TraceAnalyzer*.

Returns A recipe in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Any]

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a Montage workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_degree(*dataset:* wfcommons.generator.workflow.montage_recipe.MontageDataset, *num_bands: int*, *degree: float*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.montage_recipe.MontageRecipe*

Instantiate a Montage workflow recipe that will generate synthetic workflows using the defined dataset, number of bands, and degree.

Parameters

- **dataset** (*MontageDataset*) – The dataset to use for the mosaic (e.g., 2mass, dss).
- **num_bands** (*int*) – The number of bands (e.g., red, blue, and green) used by the workflow (at least 1).
- **degree** (*float*) – The size (in degrees) to be used for the width/height of the final mosaic (at least 0.5).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Montage workflow recipe object that will generate synthetic workflows using the defined dataset, number of bands, and degree.

Return type *MontageRecipe*

classmethod from_num_tasks(*num_tasks: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.montage_recipe.MontageRecipe*

Instantiate a Montage workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 133).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.

- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Montage workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *MontageRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

class wfcommons.generator.workflow.montage_recipe.**_MontagetaskRatios**

Bases: object

An auxiliary class for generating Montage tasks.

_get_max_num_tasks(*task_name: str*, *degree: float*, *dataset:* wfcommons.generator.workflow.montage_recipe.MontageDataset) → int

Get the maximum number of tasks that can be generated for a defined task.

Parameters

- **task_name** (*str*) – The task name prefix.
- **degree** (*float*) – The size (in degrees) to be used for the width/height of the final mosaic.
- **dataset** (MontageDataset) – The dataset to use for the mosaic (e.g., 2mass, dss).

Returns The maximum number of tasks that can be generated for a defined task.

Return type int

_get_max_rate_increase(*task_name: str*, *dataset:* wfcommons.generator.workflow.montage_recipe.MontageDataset) → int

Get the maximum rate of increase for a task prefix by increasing the workflow degree.

Parameters

- **task_name** (*str*) – The task name prefix.
- **dataset** (MontageDataset) – The dataset to use for the mosaic (e.g., 2mass, dss).

Returns The maximum rate of increase for a task prefix by increasing the workflow degree.

Return type int

_get_num_tasks(*task_name: str*, *degree: float*, *dataset:* wfcommons.generator.workflow.montage_recipe.MontageDataset) → int

Get a random number of tasks to be generated for a task prefix and workflow degree.

Parameters

- **task_name** (*str*) – The task name prefix.
- **degree** (*float*) – The size (in degrees) to be used for the width/height of the final mosaic.
- **dataset** (MontageDataset) – The dataset to use for the mosaic (e.g., 2mass, dss).

Returns A random number of tasks to be generated for a task prefix and workflow degree.

Return type int

```
tasks_ratios = {<MontageDataset.TWOMASS>:  {'mProject':  (68, 44, 21), 'mDiffFit':  (4
```

7.2.9 wfcommons.generator.workflow.seismology_recipe

class wfcommons.generator.workflow.seismology_recipe.**SeismologyRecipe**(*num_pairs: Optional[int] = 2*, *data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

A Seismology workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_pairs** (*int*) – The number of pair of signals to estimate earthquake STFs.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict

Recipe for generating synthetic traces of the Seismology workflow. Recipes can be generated by using the

TraceAnalyzer.

Returns A recipe in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Any]

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a Seismology workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_pairs(*num_pairs: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.seismology_recipe.SeismologyRecipe*

Instantiate a Seismology workflow recipe that will generate synthetic workflows using the defined number of pairs.

Parameters

- **num_pairs** (*int*) – The number of pair of signals to estimate earthquake STFs (at least 2).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Seismology workflow recipe object that will generate synthetic workflows using the defined number of pairs.

Return type *SeismologyRecipe*

classmethod from_num_tasks(*num_tasks: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.seismology_recipe.SeismologyRecipe*

Instantiate a Seismology workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 3).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A Seismology workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *SeismologyRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.10 wfcommons.generator.workflow.soykb_recipe

class wfcommons.generator.workflow.soykb_recipe.**SoyKBRecipe**(*num_fastq_files: Optional[int] = 2, num_chromosomes: Optional[int] = 1, data_footprint: Optional[int] = 0, num_tasks: Optional[int] = 14, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

A SoyKB workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_fastq_files** (*int*) – The number of FASTQ files to be analyzed.
- **num_chromosomes** (*int*) – The number of chromosomes.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict

Recipe for generating synthetic traces of the SoyKB workflow. Recipes can be generated by using the *TraceAnalyzer*.

Returns A recipe in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Any]

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of a SoyKB workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_tasks(*num_tasks: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.soykb_recipe.SoyKBRecipe*

Instantiate a SoyKB workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 14).
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A SoyKB workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *SoyKBRecipe*

classmethod from_sequences(*num_fastq_files: int*, *num_chromosomes: int*, *runtime_factor: Optional[float] = 1.0*, *input_file_size_factor: Optional[float] = 1.0*, *output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.soykb_recipe.SoyKBRecipe*

Instantiate a SoyKB workflow recipe that will generate synthetic workflows using the defined number of FASTQ files and chromosomes.

Parameters

- **num_fastq_files** (*int*) – The number of FASTQ files to be analyzed (at least 2).
- **num_chromosomes** (*int*) – The number of chromosomes (range [1,22].
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns A SoyKB workflow recipe object that will generate synthetic workflows using the defined number of FASTQ files and chromosomes.

Return type *SoyKBRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.2.11 wfcommons.generator.workflow.srasearch_recipe

class wfcommons.generator.workflow.srasearch_recipe.**SRASearchRecipe**(*num_accession: Optional[int] = 2, data_footprint: Optional[int] = 0, num_tasks: Optional[int] = 3, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*)

Bases: *wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe*

An SRA Search workflow recipe class for creating synthetic workflow traces.

Parameters

- **num_accession** (*int*) – The number of NCBI accession numbers.
- **data_footprint** (*int*) – The upper bound for the workflow total data footprint (in bytes).
- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

_abc_impl = <_abc_data object>

_add_merge_task(*workflow*, *input_files*, *parents*) → *wfcommons.common.task.Task*

Create a merge task.

Parameters

- **workflow** – Workflow object instance.
- **input_files** – List of input files for the task.

- **parents** – List of parent tasks.

Rtype workflow Workflow

Rtype input_files List[File]

Rtype parents List[Task]

Returns A merge task object.

_workflow_recipe() → Dict

Recipe for generating synthetic traces of the SRA Search workflow. Recipes can be generated by using the *TraceAnalyzer*.

Returns A recipe in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Any]

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Generate a synthetic workflow trace of an SRA Search workflow.

Parameters **workflow_name** (*int*) – The workflow name

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_accession(*num_accession: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.srasearch_recipe.SRASearchRecipe*

Instantiate an SRA Search workflow recipe that will generate synthetic workflows using the defined number of pairs.

Parameters

- **num_accession** (*int*) – The number of NCBI accession numbers.
- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns An SRA Search workflow recipe object that will generate synthetic workflows using the defined number of pairs.

Return type *SRASearchRecipe*

classmethod from_num_tasks(*num_tasks: int, runtime_factor: Optional[float] = 1.0, input_file_size_factor: Optional[float] = 1.0, output_file_size_factor: Optional[float] = 1.0*) → *wfcommons.generator.workflow.srasearch_recipe.SRASearchRecipe*

Instantiate an SRA Search workflow recipe that will generate synthetic workflows up to the total number of tasks provided.

Parameters

- **num_tasks** (*int*) – The upper bound for the total number of tasks in the workflow (at least 6).

- **runtime_factor** (*float*) – The factor of which tasks runtime will be increased/decreased.
- **input_file_size_factor** (*float*) – The factor of which tasks input files size will be increased/decreased.
- **output_file_size_factor** (*float*) – The factor of which tasks output files size will be increased/decreased.

Returns An SRA Search workflow recipe object that will generate synthetic workflows up to the total number of tasks provided.

Return type *SRASearchRecipe*

tasks_files: Dict[str, List[File]]

workflows: List[Workflow]

7.3 wfcommons.trace

7.3.1 wfcommons.trace.schema

class wfcommons.trace.schema.**SchemaValidator**(*schema_file: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

Bases: object

Validate JSON files against WfCommons schema. If schema file path is not provided, it will look for a local copy of the WfCommons schema, and if not available it will fetch the latest schema from the WfCommons schema GitHub repository.

Parameters

- **schema_file** (*str*) – JSON schema file path.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

_load_schema(*schema_file: Optional[str] = None*)

Load the schema file. If schema file path is not provided, it will look for a local copy of the WfCommons schema, and if not available it will fetch the latest schema from the GitHub repository.

Parameters **schema_file** (*str*) – JSON schema file path.

Returns The JSON schema.

Return type json

_semantic_validation(*data: Dict[str, Any]*)

Validate the semantics of the JSON workflow execution trace.

Parameters **data** (*Dict[str, Any]*) – Workflow trace in JSON format.

_syntax_validation(*data: Dict[str, Any]*)

Validate the JSON workflow execution trace against the schema.

Parameters **data** (*Dict[str, Any]*) – Workflow trace in JSON format.

validate_trace(*data: Dict[str, Any]*)

Perform syntax validation against the schema, and semantic validation.

Parameters **data** (*Dict[str, Any]*) – Workflow trace in JSON format.

7.3.2 wfcommons.trace.trace

class wfcommons.trace.trace.**Trace**(*input_trace: str*, *schema_file: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

Bases: object

Representation of one execution of one workflow on a set of machines

```
Trace(input_trace = 'trace.json')
```

Parameters

- **input_trace** (*str*) – The JSON trace.
- **schema_file** (*str*) – The path to the JSON schema that defines the trace. If no schema file is provided, it will look for a local copy of the WfCommons schema, and if not available it will fetch the latest schema from the WfCommons schema GitHub repository.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

draw(*output: Optional[str] = None*, *extension: str = 'pdf'*) → None

Produce an image or a pdf file representing the trace.

Parameters

- **output** (*str*) – Name of the output file.
- **extension** – Type of the file extension (pdf, png, or svg).

leaves() → List[str]

Get the leaves of the workflow (i.e., the tasks without any successors).

Returns List of leaves

Return type List[str]

roots() → List[str]

Get the roots of the workflow (i.e., the tasks without any predecessors).

Returns List of roots

Return type List[str]

write_dot(*output: Optional[str] = None*) → None

Write a dot file of the trace.

Parameters **output** (*str*) – The output dot file name (optional).

7.3.3 wfcommons.trace.trace_analyzer

class wfcommons.trace.trace_analyzer.**TraceAnalyzer**(*logger: Optional[logging.Logger] = None*)

Bases: object

Set of tools for analyzing collections of traces.

Parameters **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

append_trace(*trace:* wfcommons.trace.trace.Trace) → None

Append a workflow trace object to the trace analyzer.

```
trace = Trace(input_trace = 'trace.json', schema = 'schema.json')
trace_analyzer = TraceAnalyzer()
trace_analyzer.append_trace(trace)
```

Parameters **trace** (Trace) – A workflow trace object.

build_summary(*tasks_list: List[str]*, *include_raw_data: Optional[bool] = True*) → Dict[str, Dict[str, Any]]

Analyzes appended traces and produce a summary of the analysis per task prefix.

```
workflow_tasks = ['sG1IterDecon', 'wrapper_siftSTFByMisfit']
traces_summary = trace_analyzer.build_summary(workflow_tasks, include_raw_
↪data=False)
```

Parameters

- **tasks_list** (*List[str]*) – List of workflow tasks prefix (e.g., mProject, sol2sanger, add_replace)
- **include_raw_data** (*bool*) – Whether to include the raw data in the trace summary.

Returns A summary of the analysis of traces in the form of a dictionary in which keys are task prefixes.

Return type Dict[str, Dict[str, Any]]

generate_all_fit_plots(*outfile_prefix: Optional[str] = None*) → None

Produce fit plots as images for each entry of the summary analysis. For entries in which there are no distribution (i.e., constant value), no plot will be generated.

Parameters **outfile_prefix** (*str*) – Prefix to be attached to each generated plot file name (optional).

generate_fit_plots(*trace_element:* wfcommons.trace.trace_analyzer.TraceElement, *outfile_prefix: Optional[str] = None*) → None

Produce fit plots as images for each entry of a trace element generated by the summary analysis. For entries in which there are no distribution (i.e., constant value), no plot will be generated.

Parameters

- **trace_element** (TraceElement) – Workflow element for which the fit plots will be generated.
- **outfile_prefix** (*str*) – Prefix to be attached to each generated plot file name (optional).

class wfcommons.trace.trace_analyzer.**TraceElement**(*value*)

Bases: *wfcommons.utils.NoValue*

An enumeration.

INPUT = ('input', 'Input File Size (bytes)')

OUTPUT = ('output', 'Input File Size (bytes)')

RUNTIME = ('runtime', 'Runtime (s)')

wfcommons.trace.trace_analyzer.**_append_file_to_dict**(*extension: str*, *dict_obj: Dict[str, Any]*, *file_size: int*) → None

Add a file size to a file type extension dictionary.

Parameters

- **extension** (*str*) – File type extension.
- **dict_obj** (*Dict[str, Any]*) – Dictionary of file type extensions.
- **file_size** (*int*) – File size in bytes.

wfcommons.trace.trace_analyzer.**_best_fit_distribution_for_file**(*dict_obj*, *include_raw_data*) → None

Find the best fit distribution for a file.

Parameters

- **dict_obj** (*Dict[str, Any]*) – Dictionary of file type extensions.
- **include_raw_data** (*bool*) –

wfcommons.trace.trace_analyzer.**_generate_fit_plots**(*el: Dict*, *title: str*, *xlabel: str*, *outfile: str*, *font_size: Optional[int] = None*, *logger: Optional[logging.Logger] = None*) → None

Produce a fit plot as an image for an entry of a trace element generated by the summary analysis.

Parameters

- **el** (*Dict*) – Entry of a trace element generated by the summary analysis.
- **title** (*str*) – Plot title.
- **xlabel** (*str*) – X-axis label.
- **outfile** (*Optional[int]*) – Plot file name.
- **font_size** – Size of the font.
- **logger** (*Logger*) – The logger where to log information/warning or errors.

wfcommons.trace.trace_analyzer.**_json_format_distribution_fit**(*dist_tuple: Tuple*) → Dict[str, Any]

Format the best fit distribution data into a dictionary

Parameters **dist_tuple** (*Tuple*) – Tuple containing best fit distribution name and parameters.

Returns

Return type Dict[str, Any]

7.3.4 wfcommons.trace.logs.abstract_logs_parser

class wfcommons.trace.logs.abstract_logs_parser.**LogsParser**(*wms_name: str*, *wms_url: Optional[str] = None*, *description: Optional[str] = None*, *logger: Optional[logging.Logger] = None*)

Bases: abc.ABC

An abstract class of logs parser for creating workflow traces.

Parameters

- **wms_name** (*str*) – Name of the workflow system.

- **wms_url** (*str*) – URL for the workflow system.
- **description** (*str*) – Workflow trace description.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

_abc_impl = <_abc_data object>

abstract build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*

Create workflow trace based on the workflow execution logs.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A workflow trace object.

Return type *Workflow*

7.3.5 wfcommons.trace.logs.makeflow

class wfcommons.trace.logs.makeflow.**MakeflowLogsParser**(*execution_dir: str, resource_monitor_logs_dir: str, description: Optional[str] = None, logger: Optional[logging.Logger] = None*)

Bases: wfcommons.trace.logs.abstract_logs_parser.LogsParser

Parse Makeflow submit directory to generate workflow trace.

Parameters

- **execution_dir** (*str*) – Makeflow workflow execution directory (contains .mf and .makeflowlog files).
- **resource_monitor_logs_dir** (*str*) – Resource Monitor log files directory.
- **description** (*str*) – Workflow trace description.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

_abc_impl = <_abc_data object>

_create_files(*files_list: List[str], link:* wfcommons.common.file.FileLink, *task_name: str*)

Create a list of files objects.

Parameters

- **files_list** – list of file names.
- **link** – Link type for the files in the list.
- **task_name** – Task name.

Rtype files_list List[str]

Rtype link FileLink

Rtype task_name str

Returns List of file objects.

Return type List[*File*]

_parse_makeflow_log_file()

Parse the makeflow log file and update workflow task information.

_parse_resource_monitor_logs()
Parse the log files produced by resource monitor

_parse_workflow_file()
Parse the makeflow workflow file and build the workflow structure.

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*
Create workflow trace based on the workflow execution logs.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A workflow trace object.

Return type *Workflow*

7.3.6 wfcommons.trace.logs.pegasus

class wfcommons.trace.logs.pegasus.**PegasusLogsParser**(*submit_dir: str*, *description: Optional[str] = None*, *ignore_auxiliary: Optional[bool] = True*, *legacy: Optional[bool] = False*, *logger: Optional[logging.Logger] = None*)
Bases: wfcommons.trace.logs.abstract_logs_parser.LogsParser

Parse Pegasus submit directory to generate workflow trace.

Parameters

- **submit_dir** (*str*) – Pegasus submit directory.
- **legacy** (*bool*) – Whether the submit directory is from a Pegasus 4.x version.
- **description** (*str*) – Workflow trace description.
- **ignore_auxiliary** (*bool*) – Ignore auxiliary jobs.
- **logger** (*Logger*) – The logger where to log information/warning or errors (optional).

_abc_impl = <_abc_data object>

_fetch_all_files(*extension: str*, *file_name: str = ''*)
Fetch all files from the directory and its hierarchy

Parameters

- **extension** (*str*) – file extension to be searched for
- **file_name** (*str*) – file_name to be searched

Returns List of file names that match

Return type List[str]

_parse_braindump()
Parse the Pegasus braindump.txt file

_parse_dag()
Parse the DAG file.

_parse_dax()
Parse the DAX file.

_parse_job_output(*task*)
Parse the kickstart job output file (e.g., .out.000).

Parameters **task** (Task) – Task object.

_parse_job_output_latest(*task*, *output_file*)
Parse the kickstart job output file in YAML format (e.g., .out.000).

Parameters

- **task** (Task) – Task object.
- **output_file** (*str*) – Output file name.

_parse_job_output_legacy(*task*, *output_file*)
Parse the kickstart job output file in XML format (e.g., .out.000).

Parameters

- **task** (Task) – Task object.
- **output_file** (*str*) – Output file name.

_parse_meta_file(*task_name*)
Parse the Pegasus meta file (generated from pegasus-integrity)

Parameters **task_name** (*str*) – Task file name.

_parse_workflow()
Parse the Workflow file.

build_workflow(*workflow_name: Optional[str] = None*) → *wfcommons.common.workflow.Workflow*
Create workflow trace based on the workflow execution logs.

Parameters **workflow_name** (*str*) – The workflow name.

Returns A workflow trace object.

Return type *Workflow*

PYTHON MODULE INDEX

W

wfcommons.common.file, 17
wfcommons.common.machine, 19
wfcommons.common.task, 18
wfcommons.common.workflow, 20
wfcommons.generator.generator, 23
wfcommons.generator.workflow.abstract_recipe, 45
wfcommons.generator.workflow.blast_recipe, 24
wfcommons.generator.workflow.bwa_recipe, 26
wfcommons.generator.workflow.cycles_recipe, 28
wfcommons.generator.workflow.epigenomics_recipe, 30
wfcommons.generator.workflow.genome_recipe, 32
wfcommons.generator.workflow.montage_recipe, 34
wfcommons.generator.workflow.seismology_recipe, 37
wfcommons.generator.workflow.soykb_recipe, 39
wfcommons.generator.workflow.srasearch_recipe, 41
wfcommons.trace.logs.makeflow, 22
wfcommons.trace.logs.pegasus, 23
wfcommons.trace.schema, 68
wfcommons.trace.trace, 20
wfcommons.trace.trace_analyzer, 21
wfcommons.utils, 43

INDEX

Symbols

_abc_impl (*wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe attribute*), 45
_generate_file() (*wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe method*), 45
_generate_files() (*wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe method*), 46
_generate_task() (*wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe method*), 46
_generate_task_name() (*wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe method*), 46
_get_files_by_task_and_link() (*wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe method*), 46
_load_schema() (*wfcommons.trace.schema.SchemaValidator method*), 68
_semantic_validation() (*wfcommons.trace.schema.SchemaValidator method*), 68
_syntax_validation() (*wfcommons.trace.schema.SchemaValidator method*), 68
_workflow_recipe() (*wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe method*), 46

A

append_trace() (*wfcommons.trace.trace_analyzer.TraceAnalyzer method*), 21
as_dict() (*wfcommons.common.file.File method*), 17
as_dict() (*wfcommons.common.machine.Machine method*), 19
as_dict() (*wfcommons.common.task.Task method*), 18
AUXILIARY (*wfcommons.common.task.TaskType attribute*), 18

B

beta_distribution() (*in module wfcommons.utils*), 43
bin_size (*wfcommons.generator.workflow.epigenomics_recipe.EpigenomicsRecipe attribute*), 31
BLASTRecipe (*class in wfcommons.generator.workflow.blast_recipe*), 24
build_summary() (*wfcommons.trace.trace_analyzer.TraceAnalyzer method*), 21
build_workflow() (*wfcommons.generator.generator.WorkflowGenerator method*), 24
build_workflow() (*wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe method*), 47
build_workflow() (*wfcommons.generator.workflow.blast_recipe.BLASTRecipe method*), 24
build_workflow() (*wfcommons.generator.workflow.bwa_recipe.BWARecipe method*), 26
build_workflow() (*wfcommons.generator.workflow.cycles_recipe.CyclesRecipe method*), 28
build_workflow() (*wfcommons.generator.workflow.epigenomics_recipe.EpigenomicsRecipe method*), 31
build_workflow() (*wfcommons.generator.workflow.genome_recipe.GenomeRecipe method*), 33
build_workflow() (*wfcommons.generator.workflow.montage_recipe.MontageRecipe method*), 35
build_workflow() (*wfcommons.generator.workflow.seismology_recipe.SeismologyRecipe method*), 37
build_workflow() (*wfcommons.generator.workflow.soykb_recipe.SoyKBRecipe method*), 39
build_workflow() (*wfcommons.generator.workflow.srasearch_recipe.SRASearchRecipe*

method), 41
build_workflow() (*wfcommons.trace.logs.makeflow.MakeflowLogsParser method*), 22
build_workflow() (*wfcommons.trace.logs.pegasus.PegasusLogsParser method*), 23
build_workflows() (*wfcommons.generator.generator.WorkflowGenerator method*), 24
BWARecipe (*class in wfcommons.generator.workflow.bwa_recipe*), 26

C

COMPUTE (*wfcommons.common.task.TaskType attribute*), 18
CyclesRecipe (*class in wfcommons.generator.workflow.cycles_recipe*), 28

D

dataset (*wfcommons.generator.workflow.montage_recipe.MontageRecipe attribute*), 36
degree (*wfcommons.generator.workflow.montage_recipe.MontageRecipe attribute*), 36
draw() (*wfcommons.trace.trace.Trace method*), 20
DSS (*wfcommons.generator.workflow.montage_recipe.MontageDataset attribute*), 34

E

EpigenomicsRecipe (*class in wfcommons.generator.workflow.epigenomics_recipe*), 30

F

File (*class in wfcommons.common.file*), 17
FileLink (*class in wfcommons.common.file*), 17
from_degree() (*wfcommons.generator.workflow.montage_recipe.MontageRecipe class method*), 36
from_num_accession() (*wfcommons.generator.workflow.srasearch_recipe.SRASearchRecipe class method*), 41
from_num_chromosomes() (*wfcommons.generator.workflow.genome_recipe.GenomeRecipe class method*), 33
from_num_pairs() (*wfcommons.generator.workflow.seismology_recipe.SeismologyRecipe class method*), 38
from_num_subsample() (*wfcommons.generator.workflow.blast_recipe.BLASTRecipe class method*), 25
from_num_subsample() (*wfcommons.generator.workflow.bwa_recipe.BWARecipe class method*), 26
from_num_tasks() (*wfcommons.generator.workflow.abstract_recipe.WorkflowRecipe class method*), 47
from_num_tasks() (*wfcommons.generator.workflow.blast_recipe.BLASTRecipe class method*), 25
from_num_tasks() (*wfcommons.generator.workflow.bwa_recipe.BWARecipe class method*), 27
from_num_tasks() (*wfcommons.generator.workflow.cycles_recipe.CyclesRecipe class method*), 28
from_num_tasks() (*wfcommons.generator.workflow.epigenomics_recipe.EpigenomicsRecipe class method*), 31
from_num_tasks() (*wfcommons.generator.workflow.genome_recipe.GenomeRecipe class method*), 33
from_num_tasks() (*wfcommons.generator.workflow.montage_recipe.MontageRecipe class method*), 36
from_num_tasks() (*wfcommons.generator.workflow.seismology_recipe.SeismologyRecipe class method*), 38
from_num_tasks() (*wfcommons.generator.workflow.soykb_recipe.SoyKBRecipe class method*), 39
from_num_tasks() (*wfcommons.generator.workflow.srasearch_recipe.SRASearchRecipe class method*), 42
from_points_and_crops() (*wfcommons.generator.workflow.cycles_recipe.CyclesRecipe class method*), 29
from_sequences() (*wfcommons.generator.workflow.epigenomics_recipe.EpigenomicsRecipe class method*), 31
from_sequences() (*wfcommons.generator.workflow.soykb_recipe.SoyKBRecipe class method*), 40

G

generate_all_fit_plots() (*wfcommons.trace.trace_analyzer.TraceAnalyzer method*), 22
generate_fit_plots() (*wfcommons.trace.trace_analyzer.TraceAnalyzer method*), 22
generate_rvs() (*in module wfcommons.utils*), 43
GenomeRecipe (*class in wfcommons.generator.workflow.genome_recipe*), 32

I

INPUT (*wfcommons.common.file.FileLink attribute*), 17
INPUT (*wfcommons.trace.trace_analyzer.TraceElement attribute*), 22

L

leaves() (*wfcommons.trace.trace.Trace method*), 21
LINUX (*wfcommons.common.machine.MachineSystem attribute*), 19

M

Machine (*class in wfcommons.common.machine*), 19
MachineSystem (*class in wfcommons.common.machine*), 19
MACOS (*wfcommons.common.machine.MachineSystem attribute*), 19
MakeflowLogsParser (*class in wfcommons.trace.logs.makeflow*), 22
module
    wfcommons.common.file, 17
    wfcommons.common.machine, 19
    wfcommons.common.task, 18
    wfcommons.common.workflow, 20
    wfcommons.generator.generator, 23
    wfcommons.generator.workflow.abstract_recipe, 45
    wfcommons.generator.workflow.blast_recipe, 24
    wfcommons.generator.workflow.bwa_recipe, 26
    wfcommons.generator.workflow.cycles_recipe, 28
    wfcommons.generator.workflow.epigenomics_recipe, 30
    wfcommons.generator.workflow.genome_recipe, 32
    wfcommons.generator.workflow.montage_recipe, 34
    wfcommons.generator.workflow.seismology_recipe, 37
    wfcommons.generator.workflow.soykb_recipe, 39
    wfcommons.generator.workflow.srasearch_recipe, 41
    wfcommons.trace.logs.makeflow, 22
    wfcommons.trace.logs.pegasus, 23
    wfcommons.trace.schema, 68
    wfcommons.trace.trace, 20
    wfcommons.trace.trace_analyzer, 21
    wfcommons.utils, 43
MontageDataset (*class in wfcommons.generator.workflow.montage_recipe*), 34
MontageRecipe (*class in wfcommons.generator.workflow.montage_recipe*), 34

N

ncr() (*in module wfcommons.utils*), 43
NoValue (*class in wfcommons.utils*), 43
num_accession (*wfcommons.generator.workflow.srasearch_recipe.SRASearchRecipe attribute*), 42
num_bands (*wfcommons.generator.workflow.montage_recipe.MontageRec attribute*), 37
num_chromosomes (*wfcommons.generator.workflow.genome_recipe.GenomeRecipe attribute*), 34
num_crops (*wfcommons.generator.workflow.cycles_recipe.CyclesRecipe attribute*), 29
num_lines (*wfcommons.generator.workflow.epigenomics_recipe.Epigeno attribute*), 32
num_pairs (*wfcommons.generator.workflow.seismology_recipe.Seismolog attribute*), 38
num_params (*wfcommons.generator.workflow.cycles_recipe.CyclesRecipe attribute*), 29
num_points (*wfcommons.generator.workflow.cycles_recipe.CyclesRecipe attribute*), 29
num_populations (*wfcommons.generator.workflow.genome_recipe.GenomeRecipe attribute*), 34
num_sequence_files (*wfcommons.generator.workflow.epigenomics_recipe.EpigenomicsRecipe attribute*), 32
num_sequences (*wfcommons.generator.workflow.genome_recipe.GenomeRecipe attribute*), 34
num_subsample (*wfcommons.generator.workflow.blast_recipe.BLASTRecipe attribute*), 25
num_subsample (*wfcommons.generator.workflow.bwa_recipe.BWARecipe attribute*), 27

O

OUTPUT (*wfcommons.common.file.FileLink attribute*), 17
OUTPUT (*wfcommons.trace.trace_analyzer.TraceElement attribute*), 22

P

PegasusLogsParser (*class in wfcommons.trace.logs.pegasus*), 23

R

read_json() (*in module wfcommons.utils*), 44

roots() (*wfcommons.trace.trace.Trace method*), 21
RUNTIME (*wfcommons.trace.trace_analyzer.TraceElement attribute*), 22

S

SchemaValidator (*class in wfcommons.trace.schema*), 68
SeismologyRecipe (*class in wfcommons.generator.workflow.seismology_recipe*), 37
SoyKBRecipe (*class in wfcommons.generator.workflow.soykb_recipe*), 39
SRASearchRecipe (*class in wfcommons.generator.workflow.srasearch_recipe*), 41

T

Task (*class in wfcommons.common.task*), 18
tasks_files (*wfcommons.generator.workflow.blast_recipe.BLASTRecipe attribute*), 25
tasks_files (*wfcommons.generator.workflow.bwa_recipe.BWARecipe attribute*), 27
tasks_files (*wfcommons.generator.workflow.cycles_recipe.CyclesRecipe attribute*), 29
tasks_files (*wfcommons.generator.workflow.epigenomics_recipe.EpigenomicsRecipe attribute*), 32
tasks_files (*wfcommons.generator.workflow.genome_recipe.GenomeRecipe attribute*), 34
tasks_files (*wfcommons.generator.workflow.montage_recipe.MontageRecipe attribute*), 37
tasks_files (*wfcommons.generator.workflow.seismology_recipe.SeismologyRecipe attribute*), 38
tasks_files (*wfcommons.generator.workflow.soykb_recipe.SoyKBRecipe attribute*), 40
tasks_files (*wfcommons.generator.workflow.srasearch_recipe.SRASearchRecipe attribute*), 42
TaskType (*class in wfcommons.common.task*), 18
Trace (*class in wfcommons.trace.trace*), 20
TraceAnalyzer (*class in wfcommons.trace.trace_analyzer*), 21
TraceElement (*class in wfcommons.trace.trace_analyzer*), 22
TRANSFER (*wfcommons.common.task.TaskType attribute*), 18
TWOMASS (*wfcommons.generator.workflow.montage_recipe.MontageDatas attribute*), 34

V

validate_trace() (*wfcommons.trace.schema.SchemaValidator method*), 68

W

wfcommons.common.file
    module, 17
wfcommons.common.machine
    module, 19
wfcommons.common.task
    module, 18
wfcommons.common.workflow
    module, 20
wfcommons.generator.generator
    module, 23
wfcommons.generator.workflow.abstract_recipe
    module, 45
wfcommons.generator.workflow.blast_recipe
    module, 24
wfcommons.generator.workflow.bwa_recipe
    module, 26
wfcommons.generator.workflow.cycles_recipe
    module, 28
wfcommons.generator.workflow.epigenomics_recipe
    module, 30
wfcommons.generator.workflow.genome_recipe
    module, 32
wfcommons.generator.workflow.montage_recipe
    module, 34
wfcommons.generator.workflow.seismology_recipe
    module, 37
wfcommons.generator.workflow.soykb_recipe
    module, 39
wfcommons.generator.workflow.srasearch_recipe
    module, 41
wfcommons.trace.logs.makeflow
    module, 22
wfcommons.trace.logs.pegasus
    module, 23
wfcommons.trace.schema
    module, 68
wfcommons.trace.trace
    module, 20
wfcommons.trace.trace_analyzer
    module, 21
wfcommons.utils
    module, 43
WINDOWS (*wfcommons.common.machine.MachineSystem attribute*), 19
Workflow (*class in wfcommons.common.workflow*), 20

WorkflowGenerator (*class in wfcommons.generator.generator*), 23
WorkflowRecipe (*class in wfcommons.generator.workflow.abstract_recipe*), 45
workflows (*wfcommons.generator.workflow.blast_recipe.BLASTRecipe attribute*), 25
workflows (*wfcommons.generator.workflow.bwa_recipe.BWARecipe attribute*), 27
workflows (*wfcommons.generator.workflow.cycles_recipe.CyclesRecipe attribute*), 29
workflows (*wfcommons.generator.workflow.epigenomics_recipe.EpigenomicsRecipe attribute*), 32
workflows (*wfcommons.generator.workflow.genome_recipe.GenomeRecipe attribute*), 34
workflows (*wfcommons.generator.workflow.montage_recipe.MontageRecipe attribute*), 37
workflows (*wfcommons.generator.workflow.seismology_recipe.SeismologyRecipe attribute*), 38
workflows (*wfcommons.generator.workflow.soykb_recipe.SoyKBRecipe attribute*), 40
workflows (*wfcommons.generator.workflow.srasearch_recipe.SRASearchRecipe attribute*), 42
write_dot() (*wfcommons.common.workflow.Workflow method*), 20
write_dot() (*wfcommons.trace.trace.Trace method*), 21
write_json() (*wfcommons.common.workflow.Workflow method*), 20