WorkflowHub

Release 0.4-dev

Oct 12, 2020

Quickstart

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

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

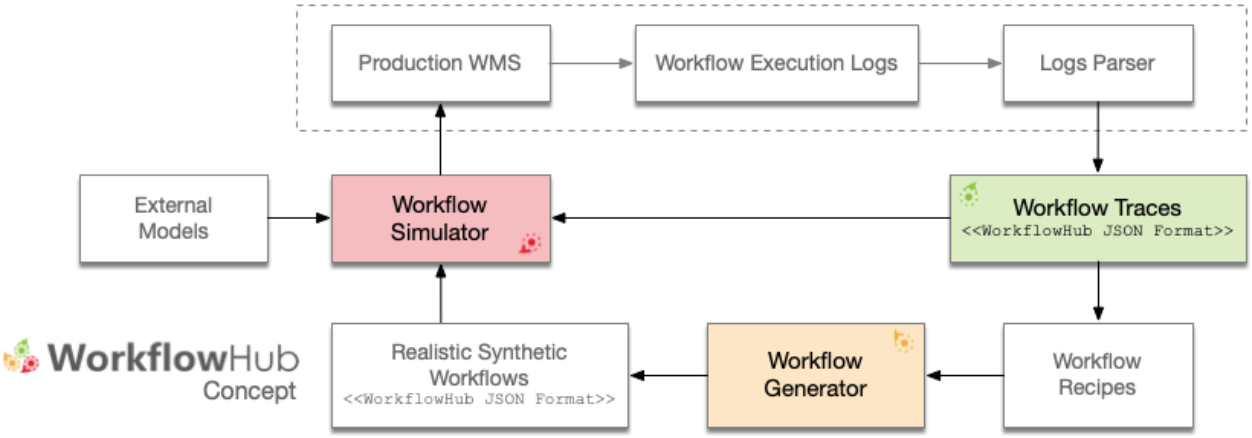

Fig. 1: The WorkflowHub conceptual architecture.

CHAPTER 1

Support

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

You can also reach the WorkflowHub team via our support email: support@workflowhub.org.

1.1 Installation

WorkflowHub is available on PyPI. WorkflowHub requires Python3.5+ and has been tested on Linux and macOS.

1.1.1 Requirements

Graphviz

WorkflowHub 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 `brew` package manager:

```
$ brew install graphviz
```

1.1.2 Installation using pip

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

```
$ python3 -m pip install workflowhub
```

1.1.3 Retrieving the latest unstable version

If you want to use the latest WorkflowHub 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 WorkflowHub's GitHub repository:

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

1.2 The WorkflowHub Project

The WorkflowHub project is a community framework for enabling scientific workflow research and development by providing foundational tools for analyzing workflow execution traces, and generating synthetic, yet realistic, workflow traces 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 WorkflowHub project:

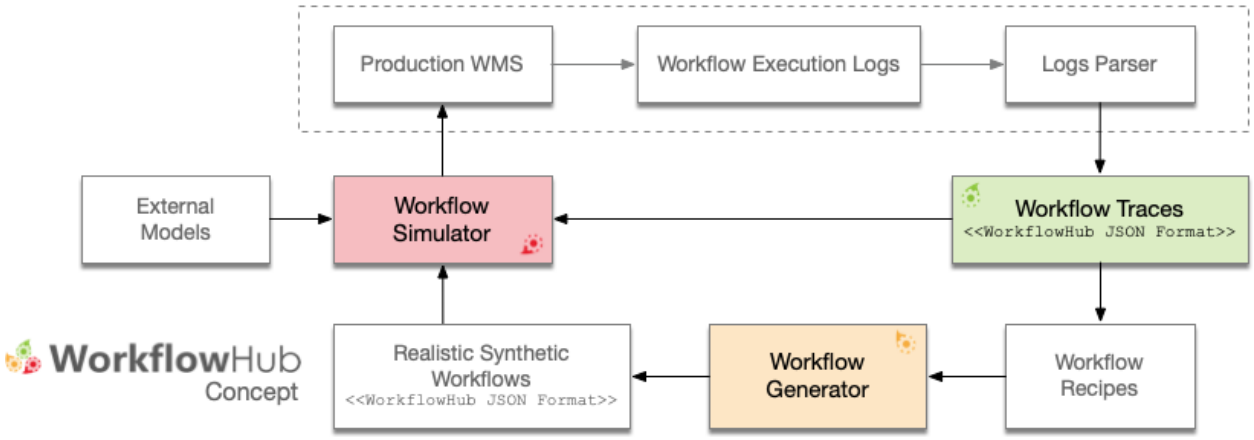

Fig. 1: The WorkflowHub conceptual architecture.

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

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

The *third axis* (**Workflow Simulator**) of the WorkflowHub 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 WorkflowHub JSON format in our project website.

This Python package provides a collection of tools for:

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

1.2.1 The WorkflowHub JSON Format

The WorkflowHub project uses a common format for representing workflow execution traces and generated synthetic workflows traces, so that workflow simulators and simulation frameworks (that provide support for WorkflowHub format) can use such traces interchangeably. This common format uses a JSON specification available in the WorkflowHub JSON schema GitHub repository. The current version of the WorkflowHub Python package uses the schema version `1.0`. The schema GitHub repository provides detailed explanation of the WorkflowHub JSON format (including required fields), and also a validator script for verifying the compatibility of traces.

1.3 Analyzing Traces

Workflow execution traces 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 WorkflowHub project targets the analysis of actual workflow execution traces (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 traces that resemble the structure and distribution of the original workflow executions.

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

1.3.1 Workflow Execution Traces

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

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

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

- *draw()*: produces an image or a pdf file representing the trace.
- *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 trace.

1.3.2 The Trace Analyzer

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

Note: Although any workflow execution trace represented as a *Trace* object (i.e., compatible with *The WorkflowHub JSON Format*) can be appended to the *TraceAnalyzer*, we strongly recommend that only traces 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 traces for analysis. The *build_summary()* method processes all appended traces. 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 traces 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 traces*.

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.

1.3.3 Examples

The following example shows the analysis of a set of traces, 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 workflowhub import Trace, TraceAnalyzer
from os import listdir
from os.path import isfile, join
```

(continues on next page)

(continued from previous page)

```
# obtaining list of trace files in the folder
TRACES_PATH = "/Path/to/some/trace/folder/"
trace_files = [f for f in listdir(TRACES_PATH) if isfile(join(TRACES_PATH, f))]

# creating the trace analyzer object
analyzer = TraceAnalyzer()

# appending trace files to the trace analyzer
for trace_file in trace_files:
    trace = Trace(input_trace=TRACES_PATH + trace_file)
    analyzer.append_trace(trace)

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

# building the trace summary
traces_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')
```

1.4 Generating Workflows

The second axis of the WorkflowHub project targets the generation of realistic synthetic workflow traces with a variety of characteristics. The *WorkflowGenerator* class uses recipes of workflows (as described in *Analyzing Traces*) 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 WorkflowHub JSON format, which is already supported by simulation frameworks such as WRENCH.

1.4.1 Workflow Recipes

The WorkflowHub package provides a number of *workflow recipes* for generating realistic synthetic workflow traces. 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 workflowhub.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 workflowhub.generator import EpigenomicsRecipe, GenomeRecipe

```

(continues on next page)

(continued from previous page)

```
# 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:

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

1.4.2 The Workflow Generator

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

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

The build methods use the workflow recipe for generating realistic synthetic workflow traces, 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 traces (see *Analyzing Traces*).

Each generated trace 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 trace into files:

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

1.4.3 Examples

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

```
from workflowhub import WorkflowGenerator
from workflowhub.generator import SeismologyRecipe

# creating a Seismology workflow recipe based on the number
# of pair of signals to estimate earthquake STFs
```

(continues on next page)

(continued from previous page)

```
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 trace of the Seismology workflow
workflow = generator.build_workflow()

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

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

```
from workflowhub import WorkflowGenerator
from workflowhub.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 traces of the Cycles workflow
workflows_list = generator.build_workflows(num_workflows=10)

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

1.5 User API Reference

The user API reference targets users who want to use WorkflowHub 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*.

1.5.1 workflowhub.common

workflowhub.common.file

class workflowhub.common.file.**File**(*name: str*, *size: int*, *link: workflowhub.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.

class workflowhub.common.file.**FileLink**
Bases: *workflowhub.utils.NoValue*

Type of file link.

INPUT = 'input'

OUTPUT = 'output'

workflowhub.common.task

class workflowhub.common.task.**Task**(*name: str, task_type: workflowhub.common.task.TaskType, runtime: float, cores: int, machine: Optional[workflowhub.common.machine.Machine], args: List[str], avg_cpu: Optional[float], bytes_read: Optional[int], bytes_written: Optional[int], memory: Optional[int], energy: Optional[int], avg_power: Optional[float], priority: Optional[int], files: List[workflowhub.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.

class workflowhub.common.task.**TaskType**
Bases: *workflowhub.utils.NoValue*

Task type.

COMPUTE = 'compute'

workflowhub.common.machine

class workflowhub.common.machine.**Machine**(*name: str, cpu: Dict[str, Union[int, str]], system: Optional[workflowhub.common.machine.MachineSystem], architecture: Optional[str], memory: Optional[int], release: Optional[str], hashcode: Optional[str], 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** (*Optional[Logger]*) – The logger where to log information/warning or errors.

class workflowhub.common.machine.**MachineSystem**

Bases: *workflowhub.utils.NoValue*

Machine system type.

LINUX = 'linux'

MACOS = 'macos'

WINDOWS = 'windows'

workflowhub.common.workflow

class workflowhub.common.workflow.**Workflow**(*name: str, makespan: Optional[int]*)

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.

- **makespan** (*int*) – Workflow makespan in seconds.

write_dot (*dot_filename: 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.

1.5.2 workflowhub.trace

workflowhub.trace.trace

class workflowhub.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 WorkflowHub schema, and if not available it will fetch the latest schema from the WorkflowHub 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).

workflowhub.trace.trace_analyzer

class workflowhub.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: workflowhub.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: workflowhub.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 workflowhub.trace.trace_analyzer.**TraceElement**
Bases: *workflowhub.utils.NoValue*

An enumeration.

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

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

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

1.5.3 workflowhub.generator

workflowhub.generator.generator

class workflowhub.generator.generator.**WorkflowGenerator**(*workflow_recipe: workflowhub.generator.workflow.abstract_recipe.Workflo 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*) → workflowhub.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[workflowhub.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*]

workflowhub.generator.workflow.cycles_recipe

class workflowhub.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*)

Bases: *workflowhub.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.

build_workflow(*workflow_name: Optional[str] = None*) → workflowhub.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*) → workflowhub.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).

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*) → workflowhub.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.

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

Return type *CyclesRecipe*

workflowhub.generator.workflow.epigenomics_recipe

class workflowhub.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*)

Bases: *workflowhub.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.

build_workflow(*workflow_name: str = None*) → workflowhub.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*) → workflowhub.generator.workflow.epigenomics_recipe.EpigenomicsR
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).

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*) → workflowhub.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.

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*

workflowhub.generator.workflow.genome_recipe

class workflowhub.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*)
Bases: *workflowhub.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.

build_workflow(*workflow_name: str = None*) → workflowhub.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*) → workflowhub.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.

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*) → workflowhub.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).

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

Return type *GenomeRecipe*

workflowhub.generator.workflow.montage_recipe

class workflowhub.generator.workflow.montage_recipe.**MontageDataset**
Bases: *workflowhub.utils.NoValue*

An enumeration of Montage datasets.

DSS = 'dss'

TWOMASS = '2mass'

class workflowhub.generator.workflow.montage_recipe.**MontageRecipe**(*dataset: Optional[workflowhub.generator.workfl = <Montage-Dataset.DSS>, num_bands: Optional[int] = 1, degree: Optional[float] = 0.5, data_footprint: Optional[int] = 0, num_tasks: Optional[int] = 133*)

Bases: *workflowhub.generator.workflow.abstract_recipe.WorkflowRecipe*, workflowhub.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.

build_workflow(*workflow_name: str = None*) → workflowhub.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: workflowhub.generator.workflow.montage_recipe.MontageDataset, num_bands: int, degree: float*) → workflowhub.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).

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*) → workflowhub.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).

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

Return type *MontageRecipe*

workflowhub.generator.workflow.seismology_recipe

class workflowhub.generator.workflow.seismology_recipe.**SeismologyRecipe**(*num_pairs: Optional[int] = 2*, *data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*)

Bases: *workflowhub.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.

build_workflow(*workflow_name: Optional[str] = None*) → workflowhub.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*) → workflowhub.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).

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*) → workflowhub.generator.workflow.seismology_recipe.SeismologyReci

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).

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

Return type *SeismologyRecipe*

workflowhub.generator.workflow.soykb_recipe

class workflowhub.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*)

Bases: *workflowhub.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.

build_workflow(*workflow_name: Optional[str] = None*) → workflowhub.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*) → workflowhub.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).

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*) → workflowhub.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].

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

Return type *SoyKBRecipe*

1.6 Developer API Reference

The developer API reference targets developers and researchers who want to contribute to the WorkflowHub 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.

1.6.1 workflowhub.utils

class workflowhub.utils.**NoValue**
Bases: enum.Enum

An enumeration.

workflowhub.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

workflowhub.utils.**generate_rvs**(*distribution: Dict[KT, VT], 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

workflowhub.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

workflowhub.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]

1.6.2 workflowhub.generator

workflowhub.generator.generator

class workflowhub.generator.generator.**WorkflowGenerator**(*workflow_recipe: workflowhub.generator.workflow.abstract_recipe.Workflo*
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*) → workflowhub.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[workflowhub.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*]

workflowhub.generator.workflow.abstract_recipe

class workflowhub.generator.workflow.abstract_recipe.**WorkflowRecipe**(*name: str, data_footprint: Optional[int], num_tasks: Optional[int], 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.
- **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: workflowhub.common.file.FileLink*) → workflowhub.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: workflowhub.common.file.FileLink, files_recipe: Optional[Dict[workflowhub.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[workflowhub.common.file.File]] = None*, *files_recipe: Optional[Dict[workflowhub.common.file.FileLink, Dict[str, int]]] = None*) → workflowhub.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: workflowhub.common.file.FileLink*) → List[workflowhub.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*]

_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]

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

Generate a synthetic workflow trace.

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

Returns A synthetic workflow trace object.

Return type *Workflow*

classmethod from_num_tasks(*num_tasks: int*) → workflowhub.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.

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

Return type *WorkflowRecipe*

workflowhub.generator.workflow.cycles_recipe

class workflowhub.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*)

Bases: *workflowhub.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.

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict[KT, VT]
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*) → workflowhub.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*) → workflowhub.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).

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*) → workflowhub.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.

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

Return type *CyclesRecipe*

workflowhub.generator.workflow.epigenomics_recipe

class workflowhub.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*)

Bases: *workflowhub.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.

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict[KT, VT]

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: str = None*) → workflowhub.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*) → workflowhub.generator.workflow.epigenomics_recipe.EpigenomicsR
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).

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*) → workflowhub.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.

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*

workflowhub.generator.workflow.genome_recipe

class workflowhub.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*)
Bases: *workflowhub.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.

_abc_impl = <_abc_data object>

_get_populations_files_recipe(*index: int*) → Dict[workflowhub.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[KT, VT]

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: str = None*) → workflowhub.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*) → workflowhub.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.

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*) → workflowhub.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).

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

Return type *GenomeRecipe*

workflowhub.generator.workflow.montage_recipe

class workflowhub.generator.workflow.montage_recipe.**MontageDataset**
Bases: *workflowhub.utils.NoValue*

An enumeration of Montage datasets.

DSS = 'dss'

TWOMASS = '2mass'

class workflowhub.generator.workflow.montage_recipe.**MontageRecipe**(*dataset: Optional[workflowhub.generator.workfl = <MontageDataset.DSS>, num_bands: Optional[int] = 1, degree: Optional[float] = 0.5, data_footprint: Optional[int] = 0, num_tasks: Optional[int] = 133*)
Bases: *workflowhub.generator.workflow.abstract_recipe.WorkflowRecipe*, workflowhub.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.

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict[KT, VT]
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: str = None*) → workflowhub.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: workflowhub.generator.workflow.montage_recipe.MontageDataset*, *num_bands: int*, *degree: float*) → workflowhub.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).

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*) → workflowhub.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).

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

Return type *MontageRecipe*

class workflowhub.generator.workflow.montage_recipe.**_MontagetaskRatios**
Bases: object

An auxiliary class for generating Montage tasks.

_get_max_num_tasks(*task_name: str*, *degree: float*, *dataset: workflowhub.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: workflowhub.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: workflowhub.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

workflowhub.generator.workflow.seismology_recipe

class workflowhub.generator.workflow.seismology_recipe.**SeismologyRecipe**(*num_pairs: Optional[int] = 2*, *data_footprint: Optional[int] = 0*, *num_tasks: Optional[int] = 3*)

Bases: *workflowhub.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.

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict[KT, VT]
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*) → workflowhub.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*) → workflowhub.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).

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*) → workflowhub.generator.workflow.seismology_recipe.SeismologyReci
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).

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

Return type *SeismologyRecipe*

workflowhub.generator.workflow.soykb_recipe

class workflowhub.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*)
Bases: *workflowhub.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.

_abc_impl = <_abc_data object>

_workflow_recipe() → Dict[KT, VT]
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*) → workflowhub.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*) → workflowhub.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).

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*) → workflowhub.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].

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

Return type *SoyKBRecipe*

1.6.3 workflowhub.trace

workflowhub.trace.schema

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

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

workflowhub.trace.trace

class workflowhub.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 WorkflowHub schema, and if not available it will fetch the latest schema from the WorkflowHub 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).

workflowhub.trace.trace_analyzer

class workflowhub.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: workflowhub.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: workflowhub.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 workflowhub.trace.trace_analyzer.**TraceElement**
Bases: *workflowhub.utils.NoValue*

An enumeration.

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

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

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

workflowhub.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.

workflowhub.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*) –

workflowhub.trace.trace_analyzer.**_generate_fit_plots**(*el: Dict[KT, VT], 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.

workflowhub.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]

Python Module Index

W

Index

Symbols

A

B

C

D